

João Nuno Gonçalves Costa Cavaleiro Correia

Evolutionary Computation for Classifier Assessment and Improvement

Doctoral thesis submitted to the Doctoral Program in Information Science and Technology,
supervised by Assistant Professor Fernando Jorge Penousal Martins Machado,
and presented to the Department of Informatics Engineering of the
Faculty of Sciences and Technology of the University of Coimbra.

May 2018



UNIVERSIDADE DE COIMBRA

Evolutionary Computation for Classifier Assessment and Improvement

A thesis submitted to the University of Coimbra
in partial fulfilment of the requirements for the
Doctoral Program in Information Science and Technology

by

João Nuno Gonçalves Costa Cavaleiro Correia
jncor@dei.uc.pt

Department of Informatics Engineering
Faculty of Sciences and Technology
UNIVERSITY OF COIMBRA
Coimbra, May 2018

Financial support by Fundação para a Ciência e a Tecnologia,
SFRH/BD/90968/2012.

Evolutionary Computation
for Classifier Assessment and Improvement

©2018 João Nuno Gonçalves Costa Cavaleiro Correia



This dissertation was prepared under the supervision of

Fernando Jorge Penousal Martins Machado
Assistant Professor of the Department of Informatics Engineering
of the Faculty of Sciences and Technology
of the University of Coimbra

“Where there’s a will, there’s a way.”

TB, nani o towazu Inês.

Aos meus Pais.

ABSTRACT

Typical Machine Learning (ML) approaches rely on a dataset and a model to solve problems. For most problems, optimisation of ML approaches is crucial to attain competitive performances. Most of the effort goes towards optimising the model by exploring new algorithms and tuning the parameters. Nevertheless, the dataset is also a key part for ML performance. Gathering, constructing and optimising a representative dataset is a hard task and a time-consuming endeavour, with no well-established guidelines to follow. In this thesis, we attest the use of Evolutionary Computation (EC) to assess and improve classifiers via synthesis of new instances.

An analysis of the state of the art on dataset construction is performed. The quality of the dataset is tied to the availability of data, which in most cases is hard to control. A throughout analysis is made about Instance Selection (IS) and Instance Generation (IG), which sheds light on relevant points for the development of our framework.

The Evolutionary FramEwork for Classifier assessmenT and ImproVemEnt (EFFECTIVE) is introduced and explored. The key parts of the framework are identified: the Classifier System (CS) module, which holds the ML model that is going to be assessed and improved; the EC module responsible for generating the new instances using the CS module for fitness assignment; and the Supervisor, a module responsible for managing the instances that are generated. The approach comes together in an iterative process of automatic assessment and improvement of classifiers.

In a first phase, EFFECTIVE is tested as a generator, creating instances of a particular class. Without loss of generality, we apply the framework in the domain of image generation. The problem that motivated the approach is presented first: frontal face generation. In this case, the framework relies on the combination of an EC engine and a CS module, i. e., a frontal face detector, to generate images of frontal faces. The results were revealing in two different ways. On the one hand, the approach was able to generate images that from a subjective standpoint resemble faces and are classified as such by the classifier. On the other hand, most of the images did not resemble faces, although they were classified as such by the classifier module. Based on the results, we extended the approach to generate other types of object, attaining similar results. We also combined several classifiers to study the evolution of ambiguous images, i. e. images that induce multistable perception. Overall, the results suggest that the framework is viable as a generator of instances and also that these instances are often misclassified by the CS module.

Building on these results, in a second phase, a study of EFFECTIVE for improving the performance of classifiers is performed. The core idea is to use the evolved instances that are generated by the EC engine to augment the training dataset. In this

phase, the framework uses the Supervisor module to select and filter the instances that will be added to the dataset. The retraining of the classifier with these instances completes an iteration of the framework. We tested this pipeline in a face detection problem evolving instances to: (i) expand the negative dataset; (ii) expand the positive dataset; and (iii) expand both datasets in the same iteration. Overall, the results show that: expanding the negative dataset, by adding misclassified instances, reduces the number of false alarms; expanding the positive dataset increases the number of hits; expanding positive and negative datasets allows the simultaneous reduction of false alarms and increase of hits. After demonstrating the adequacy of **EFFECTIVE** in face detection, we tested the framework in a Computational Creativity (**CC**) context to create an image generation system that promotes style change, obtaining results that further demonstrate the potential of the framework.

RESUMO

As abordagens típicas de Aprendizagem de Máquina (AM) dependem de um conjunto de instâncias e de um modelo para resolver problemas. Para a maioria dos problemas, a otimização das abordagens AM é crucial para obter desempenhos competitivos. A maior parte do esforço vai no sentido de otimizar o modelo através da exploração de novos algoritmos e do ajuste de parâmetros. No entanto, o conjunto de instâncias é também parte fundamental no desempenho de abordagens de AM. Reunir, construir e otimizar um conjunto de instâncias representativo é uma tarefa difícil e morosa, sem diretrizes bem estabelecidas. Nesta tese, atestamos o uso de Computação Evolucionária (CE) para avaliação e aperfeiçoamento de classificadores através da síntese de novas instâncias.

Efetuuou-se uma análise do estado da arte sobre construção de conjunto de instâncias. A qualidade do conjunto de instâncias está ligada à disponibilidade de dados que, na maioria dos casos, é difícil de controlar. Uma análise completa é feita sobre a seleção e geração de instâncias, o que esclarece pontos relevantes para o desenvolvimento do nosso sistema.

O **EFFECTIVE** (Sistema Evolucionário para a Avaliação e Melhoria de Classificadores) é introduzido e explorado. Os componentes principais do sistema são: o módulo sistema de classificação (SC), que contém o modelo de AM que será avaliado e melhorado; o módulo de CE responsável por gerar as novas instâncias usando o módulo SC para atribuição da aptidão; e o Supervisor, um módulo responsável por gerir as instâncias geradas. A abordagem consiste num processo iterativo de avaliação automática e aprimoramento de classificadores.

Numa primeira fase, o **EFFECTIVE** é testado como gerador, criando instâncias de uma classe em particular. Sem perda de generalidade, aplicamos o sistema no domínio da geração de imagens. O problema que motivou a abordagem é apresentado em primeiro lugar: geração de imagens de faces frontais. Neste caso, o sistema depende da combinação de um motor de CE e de um módulo SC, i. e., um detector de faces frontais, para gerar imagens de faces frontais. Os resultados foram reveladores de duas maneiras distintas. Por um lado, a abordagem foi capaz de gerar imagens que, de um ponto de vista subjectivo, se assemelham a faces e são classificadas como tal pelo classificador. Por outro lado, a maior parte das imagens não se parecem com faces, muito embora tenham sido classificadas como tal por parte do classificador. Com base nos resultados estendemos a abordagem para gerar outro tipo de objectos, obtendo resultados similares. Também se combinaram vários classificadores num estudo sobre evolução de imagens ambíguas, i. e., imagens que induzem percepção multiestável. De

um modo geral, os resultados sugerem que o sistema é viável como um gerador de instâncias e que essas instâncias são muitas vezes mal classificadas pelo SC.

Com base nos resultados obtidos, numa segunda fase, efectuámos o estudo sobre o **EFFECTIVE** para aprimoramento da performance de classificadores. A ideia central é utilizar as instâncias geradas pelo motor de CE para aumentar o conjunto de dados de treino do classificador. Nesta fase, o sistema usa o módulo Supervisor para seleccionar e filtrar as instâncias que serão adicionadas ao conjunto de treino. O re-treinar do classificador com essas instâncias completa uma iteração do sistema. Testou-se este processo num problema de detecção de faces, evoluindo instâncias para: (i) expandir o conjunto dos negativos; (ii) expandir o conjunto dos positivos; e (iii) expandir ambos os conjuntos na mesma iteração. De um modo geral, os resultados mostram que: expandindo o conjunto dos negativos, adicionando instâncias mal classificadas, reduz o número de falsos alarmes; expandindo o conjunto dos positivos aumenta o número de caras bem detetadas; expandindo o conjunto dos positivos e dos negativos ao mesmo tempo resulta na redução de falsos alarmes e no aumento de caras bem detetadas. Após demonstrar a adequação do **EFFECTIVE** na detecção de faces, testamos o sistema num contexto de Criatividade Computacional para criar um sistema de geração de imagens que promove mudança de estilo, obtendo resultados que demonstram o potencial do sistema.

ACKNOWLEDGMENTS

This thesis counted with important support and encouragement of many people, without whom, certainly, it would not have taken place. I can not find the words to express my feelings and gratitude. With everything to say but with little ability of transmitting it in a proper way, I would like to start with: *Thank you all*.

To my supervisor, Professor Penousal Machado, I am thankful for making all this possible. Thank you for the guidance, the discussions, the encouragement, the advices, the serious moments and the more relaxed ones. Thank you for helping me to not overcomplicate things, and guiding me on the multiple crossroad moments during my research.

I would like to thank the CISUC-DEI and CMS group. Particularly, I am thankful for being part of the CDV lab. I think the positive working environment played a huge roll for me to reach this point. I am thankful to all the current and former members that I have had the pleasure to meet, talk, collaborate and share a space with for all these years: Tiago Martins, Catarina Maças, Filipe Assunção, Nuno Lourenço, Ana Cláudia, António Leitão, António Cruz, Professor Pedro Martins, Professor Amílcar Cardoso, João Cunha, Bruna Sousa, Adriano Vinhas and Manuel Levi. I am also grateful to Professor Ernesto Costa, for triggering my curiosity during the classes into the research of Artificial Intelligence; and for the formal and informal moments. I would also like to thank Professor Bernardete Ribeiro for introducing and further my interest and knowledge in the areas of Pattern Recognition and Machine Learning.

In particular, I would like to thank Tiago Martins for the discussions, the laughs, the collaborations, and most of all for his friendship. This thesis came to fruition also because of his help, patient (and courage) in the final moments of the review process of this thesis.

To all my friends that cheered me into fulfil this quest despite my absence.

To all my family for the support, for being there for me, for helping me on this journey. Specially to my parents for the unconditional support, for the opportunities that have given to me since my early years, and for letting me freely pursuit all my life goals. Also to my brother, for supporting and rooting for me all these years.

To Inês, for the unconditional support, encouragement, assertiveness and love. A special thanks to Iuris, Lex, Nux, Flava, and Fox, that although they do not understand most of what went down, they cheered me up in the most difficult times. Without them none of this makes sense. Thank you for putting up with me and for continuing to collaborate in this phase of my life.

João Nuno Correia
Coimbra, May 2018

CONTENTS

1	INTRODUCTION	1
1.1	The Hypothesis	1
1.2	Objectives	3
1.3	Contributions	4
1.4	Outline	5
2	STATE OF THE ART	7
2.1	Overview and Definitions	7
2.1.1	Machine Learning Concepts and Models	8
2.2	Instance Gathering	22
2.3	Instance Selection	24
2.3.1	Filter	26
2.3.2	Wrapper	31
2.3.3	Evolutionary and Bio-Inspired	34
2.4	Instance Generation	36
2.4.1	Modification of Instances	36
2.4.2	Generative Machine Learning	38
2.4.3	Evolutionary	49
3	EVOLUTIONARY FRAMEWORK FOR CLASSIFIER ASSESSMENT AND IMPROVEMENT	53
3.1	Overview	53
3.2	Classifier System Module	56
3.3	Evolutionary Computation Module	56
3.4	Supervisor Module	57
3.5	Setup and Parametrisation	59
4	EVOLVING IMAGES OF A PARTICULAR TYPE	61
4.1	Instantiation	61
4.1.1	Classifier System(s)	62
4.1.2	Evolutionary Engine(s)	67
4.2	Evolution of Faces	69
4.2.1	Experimental Setup	69
4.2.2	Experimental Results	72
4.2.3	Summary	79
4.3	Evolution of Figurative Images	81
4.3.1	Experimental Setup	82
4.3.2	Summary	90
4.4	Evolution of Ambiguous Images	90
4.4.1	Experimental Setup	91

4.4.2	Experimental Results	93
4.4.3	Summary	99
4.5	Evolution of Photorealistic Faces	99
4.5.1	Annotation Tool	100
4.5.2	Evolutionary Engine and Fitness Assignment	100
4.5.3	Experimental Setup	102
4.5.4	Experimental Results	105
4.5.5	Summary	112
4.6	Summary	112
5	TOWARDS THE IMPROVEMENT OF CLASSIFIERS' PERFORMANCE	115
5.1	Instantiation	116
5.2	Face Detector Training and Evaluation	116
5.3	Proof of Concept	121
5.3.1	Single Run	124
5.3.2	Aggregation of Multiple Runs	126
5.4	Summary	129
6	ASSESSING AND IMPROVING CLASSIFIERS' PERFORMANCE	131
6.1	Expanding the Negative Dataset	131
6.1.1	Framework Changes	131
6.1.2	Experimental Setup	133
6.1.3	Experimental Results	133
6.1.4	Supervision and Classifier Performance	140
6.1.5	Adversarial versus Non-Adversarial Augmentation	146
6.2	Expanding the Positive Dataset	149
6.2.1	Framework Changes	150
6.2.2	Experimental Setup	150
6.2.3	Experimental Results	151
6.3	Expanding the Positive and the Negative Datasets	158
6.3.1	Framework Changes	158
6.3.2	Experimental Setup	159
6.3.3	Experimental Results	160
6.4	Summary	169
7	EFFECTIVE FOR STYLE CHANGE	171
7.1	Instantiation	172
7.1.1	Classifier System	175
7.1.2	Initial Datasets	178
7.1.3	Evolutionary Engine	179
7.2	The Experimental Results	184
7.2.1	Analysis of the Numeric Results Concerning Evolution	185
7.2.2	Analysis of the Visual Results	188
7.2.3	Classifier's Training Results	208

7.3 Summary	211
8 CONCLUSIONS AND FUTURE WORK	215
BIBLIOGRAPHY	221

ACRONYMS

AA	Artificial Artist
AI	Artificial Intelligence
AL	Active Learning
ACO	Ant Colony Optimization
AJS	Aesthetic Judgement System
ANN	Artificial Neural Network
ADASYN	ADaptive SYNthetic sampling
Ant-IS	Ant Instance Selection
BD	Big Data
CA	Co-Evolutionary Algorithm
CC	Computational Creativity
CS	Classifier System
CV	Computer Vision
CNN	Convolutional Neural Network
CART	Classification And Regression Tree
CNNR	Condensed Nearest Neighbour Rule
CPPN	Compositional Pattern Producing Network
CJA	Chen and Jozwik Algorithm
DA	Data Augmentation
DL	Deep Learning
DT	Decision Tree
DNN	Deep Neural Network
DROP	Decremental Reduction Optimization Procedure

DCGAN	Deep Convolutional Generative Adversarial Network
DBN	Deep Belief Network
DRAW	Deep Recurrent Attentive Writer
DECORATE	Diverse Ensemble Creation by Oppositional Relabelling of Artificial Training Examples
EA	Evolutionary Algorithm
EC	Evolutionary Computation
EM	Earth Mover's
EHC	Editing through Homogeneous Clusters
ENN	Edited Nearest Neighbour
ERHC	Editing and Reduction through Homogeneous Clusters
ENRBF	Edited Normalised Radial Basis Function
ELICIT	EvoLutIionary Computation vIsualizaTion
EFFECTIVE	Evolutionary FramEwork for Classifier assessmenT and ImproVemEnt
FD	Face Detection
FE	Feature Extractor
FS	Feature Selection
GA	Genetic Algorithm
GP	Genetic Programming
GPU	Graphics Processing Unit
GAN	Generative Adversarial Network
GMCA	Generalised-Modified Chang Algorithm
IG	Instance Generation
IS	Instance Selection
KNN	K-Nearest Neighbour
LBP	Local Binary Pattern

LLE	Local Linear Embedding
LSTM	Long Short Term Memory network
ML	Machine Learning
MLP	Multi-Layer Perceptron Network
MSE	Mean Square Error
MNIST	Modified National Institute of Standards and Technology
NCC	Normalised Cross-Correlation
NEvAr	Neuro Evolutionary Art
NORBERT	geNeral purpOse expReSSion Based Evolutionary aRt Tool
NEATER	filteriNg of ovEr-sampled dAtA using non-cooperaTive gamE theoRy
OCR	Optical Character Recognition
OSC	Object Selection by Clustering
PR	Pattern Recognition
PCA	Principal Component Analysis
POP	Pattern by Ordered Projection
POC-NN	Pair Opposite Class-Nearest Neighbour
PSR	Prototype Selection by Relevance
SSGA	Steady-State Genetic Algorithm
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
SNoW	Sparse Network of Winnows
SMOTE	Synthetic Minority Over-sampling Technique
SV-kNNC	Support Vector k-Nearest Neighbour Clustering
RF	Random Forest
RBM	Restricted Boltzmann Machine
RHC	Reduction through Homogeneous Clusters

RNGE	Relative Neighbourhood Graph Editing
RNN	Recurrent Neural Network
RSP	Reduction by Space Partitioning
RENN	Repeated Edited Nearest Neighbour
RMSE	Root Mean Square Error
ReLU	Rectified Linear Unit
VAE	Variational AutoEncoder
WP	Weighting Prototypes
XML	eXtensible Markup Language

LIST OF FIGURES

Figure 2.1	Description of a dataset.	8
Figure 2.2	Artificial Neural Network overview.	11
Figure 2.3	Sparse Network of Winnows classifier overview.	11
Figure 2.4	Example of different decision boundaries.	12
Figure 2.5	Maximised margin using a Support Vector Machine.	13
Figure 2.6	Mapping from the input space to feature space.	14
Figure 2.7	Adaboost training.	14
Figure 2.8	Example of the distribution of instances when using a stage classifier.	15
Figure 2.9	Convolutional Neural Network example.	16
Figure 2.10	Recurrent Neural Network overview.	17
Figure 2.11	Expanded Recurrent Neural Network example.	17
Figure 2.12	Long Short Term Memory network overview.	18
Figure 2.13	Autoencoder overview.	19
Figure 2.14	Generative Adversarial Network overview.	20
Figure 2.15	Restricted Boltzmann Machine overview.	21
Figure 2.16	Deep Belief Network overview.	22
Figure 2.17	Filter and wrapper Instance Selection methods overview.	26
Figure 2.18	Mapping of instances using Voronoi Editing, Gabriel Editing and Relative Neighbourhood Graph Editing.	27
Figure 2.19	Boundary data example.	28
Figure 2.20	Instance Generation via modification of instances.	36
Figure 2.21	The Support Vector Machine boundary for a face dataset.	37

Figure 2.22	Examples of elastic distortions applied to an image.	39
Figure 2.23	Instances generated by the interweaving algorithm.	41
Figure 2.24	Real text image generator process and outputs.	42
Figure 2.25	Images generated with a Variational AutoEncoder on CIFAR-10.	42
Figure 2.26	Images generated with a Generative Adversarial Network.	43
Figure 2.27	Comparison of instances generated by different generative Machine Learning approaches.	44
Figure 2.28	Examples of Deep Convolutional Generative Adversarial Network arithmetic.	45
Figure 2.29	Overview of the Variational AutoEncoder and Deep Recurrent Attentive Writer networks.	46
Figure 2.30	The Deep Recurrent Attentive Writer network generation process.	47
Figure 2.31	Instances generated with Deep Recurrent Attentive Writer networks.	48
Figure 2.32	Synthetic images generated using Pixel Recurrent Neural Network.	49
Figure 2.33	Overview of the process of image generation using an Evolutionary Algorithm and a Deep Neural Network.	50
Figure 3.1	The Evolutionary FramEwork for Classifier assessmentT and ImproVemEnt.	54
Figure 3.2	Evolutionary Computation run overview.	57
Figure 4.1	EFFECTIVE workflow for this chapter.	62
Figure 4.2	Sample of sub-windows extracted using Sung and Poggio's (1995) sampling algorithm.	63
Figure 4.3	Overview of the object detection process with a Cascade Classifier.	64
Figure 4.4	Haar features (image from Viola and Jones, 2001).	65
Figure 4.5	Local Binary Pattern features (image from Ahonen et al., 2006).	65
Figure 4.6	Output of the detection's grouping algorithm.	66
Figure 4.7	Example of an individual with the expression-tree $\cos(x + y)$.	67
Figure 4.8	Genotype and phenotype of an individual using eXploit-Faces engine.	68
Figure 4.9	Evolutionary Computation run overview.	71
Figure 4.10	Evolution of the average and maximum fitness when using C1, C2, and C3 to assign fitness.	73
Figure 4.11	Examples of evolved images identified as faces by the classifiers that do not resemble faces from a human perspective.	74
Figure 4.12	Fittest individual per generation.	75
Figure 4.13	Genotype and rendering of the fittest individual.	76

Figure 4.14	Examples of some of the most interesting images that have been evolved, considered to be faces by the classifier. 77
Figure 4.15	Examples of pareidolia phenomenon in everyday objects. 78
Figure 4.16	Evolution of the average and maximum fitness using <i>LBP1</i> . 78
Figure 4.17	Evolution of the average and maximum fitness using <i>LBP2</i> . 79
Figure 4.18	Example of images detected as faces by Local Binary Pattern (<i>LBP</i>) classifiers that resemble faces. 80
Figure 4.19	Example of images detected as faces by <i>LBP</i> classifiers which do not resemble faces. 80
Figure 4.20	Example of images detected as faces by <i>LBP</i> classifiers which are mainly blurred images. 81
Figure 4.21	Negatives instances used for training the cascade classifiers. 83
Figure 4.22	Instances used to train a cascade classifier for leaf detection. 84
Figure 4.23	Evolved images identified as objects by the classifiers that do not resemble the corresponding objects from a human perspective. 86
Figure 4.24	Images evolved using a detector of lips to assign fitness. 87
Figure 4.25	Images evolved using a detector of breasts to assign fitness. 88
Figure 4.26	Images evolved using a detector of leaves to assign fitness. 89
Figure 4.27	Well-known examples of ambiguous images. 91
Figure 4.28	Sample of the binarised flowers dataset. 93
Figure 4.29	Fitness progression of the best individual across generations and of the percentage of best individuals in which a flower was detected. 94
Figure 4.30	Examples of evolved images containing flowers (top row) and faces (bottom row). 95
Figure 4.31	Examples of images containing non overlapping faces and flowers. 95
Figure 4.32	Fitness of the best individual and percentage of the best individuals containing an overlap between a face and a flower. 97
Figure 4.33	Evolved images that are computationally ambiguous, but fail to induce, in our opinion, multistable interpretation in humans. 98
Figure 4.34	Evolved images considered ambiguous by both humans and computers. 98
Figure 4.35	Screenshot of the annotation tool. 101
Figure 4.36	Examples of composite faces. 103
Figure 4.37	The positive instances used to train the classifier <i>face200</i> . 104
Figure 4.38	The positive instances used to train the classifier <i>face500</i> . 104
Figure 4.39	Progression of the fitness of the best individual across generations in <i>setup200</i> . 106
Figure 4.40	Progression of the average detections in <i>setup200</i> . 108

Figure 4.41	Progression of the detection of the best individual across generations in <i>setup200</i> . 109
Figure 4.42	Fittest individual in the last generation in <i>setup200</i> . 110
Figure 4.43	Examples of faces curated by <i>face500</i> in different runs when using <i>face200</i> to guide evolution. 110
Figure 4.44	Fittest individual in the last generation for 12 different runs when using <i>face500</i> to guide evolution. 111
Figure 4.45	Examples of faces curated by <i>face200</i> in different runs when using <i>face500</i> to guide evolution. 111
Figure 4.46	Examples of faces evolved in different runs. 113
Figure 5.1	Overview of the framework workflow for the face detection experiments. 116
Figure 5.2	Sample of images of the positive training dataset with their corresponding cropped image corresponding to the object to detect. 118
Figure 5.3	All positive instances of the training dataset. 119
Figure 5.4	Sample of images of the training dataset considered as negative instances. 120
Figure 5.5	Three different object detection outcomes. 121
Figure 5.6	Instances from the test datasets. 122
Figure 5.7	All positive instances of the Feret dataset. 123
Figure 5.8	All positive instances of the CMU-MIT dataset. 124
Figure 5.9	Evolution of fitness across generations. Results are averages of 30 independent runs. 125
Figure 5.10	Evolved images classified as faces by the classifier. 125
Figure 6.1	Sample of images generated in the starting population. 134
Figure 6.2	Evolution of the average and maximum fitness across generations for the initial framework iteration for all Supervisor setups. 136
Figure 6.3	Evolution of the average and maximum fitness across generations for the last framework iteration for all Supervisor setups. 137
Figure 6.4	Sample of images generated by the framework that were added to the training dataset on the first iteration. 140
Figure 6.5	Samples of images generated that were added to the training dataset in the last iteration from each corresponding setup. 141
Figure 6.6	Evolution of the average and maximum fitness across generations for the initial framework iteration, considering the valid the EC runs. 152

Figure 6.7	Evolution of the average and maximum fitness across generations for the last framework iteration, considering the valid the EC runs. 153
Figure 6.8	Instances collected by the Supervisor of the last population of each classifier in the first iteration, not considered to be faces. 155
Figure 6.9	Instances collected by the supervisor representative of the last population of each classifier in the last iteration, not considered to be faces by the classifier. 155
Figure 6.10	Evolution of the average and maximum fitness of <i>Abstract</i> across generations for the initial framework iteration. 161
Figure 6.11	Examples collected by the supervisor of the <i>Abstract</i> EC engine. 162
Figure 6.12	Evolution of the average and maximum fitness of <i>PhotoRealistic</i> across generations for the initial framework iteration. 162
Figure 6.13	Examples collected by the supervisor of the <i>PhotoRealistic</i> EC engine. 163
Figure 6.14	Examples collected by the supervisor of the <i>Abstract</i> EC engine. 163
Figure 6.15	Evolution of the average and maximum fitness of the <i>PhotoRealistic</i> across generations for the last framework iteration, considering the valid EC runs. 164
Figure 6.16	Evolution of the average and maximum fitness of the <i>Abstract</i> across generations for the last framework iteration, considering the valid EC runs. 164
Figure 6.17	Examples collected by the supervisor of the <i>PhotoRealistic</i> . 165
Figure 7.1	Schematic of the EFFECTIVE Framework instantiation. 174
Figure 7.2	Samples of the internal dataset. 179
Figure 7.3	Images generated from trial runs of this instantiation using geNeral purpOse expReSSion Based Evolutionary aRt Tool (NORBERT). 180
Figure 7.4	Flow of the proposed hybrid algorithm. 181
Figure 7.5	Novelty computation for an individual. 183
Figure 7.6	Evolution of the fitness of the best individual of each generations. 187
Figure 7.7	Fittest individual from each population of a typical evolutionary run of the first iteration. 189
Figure 7.8	Samples of the images classified as external, generated throughout the course single typical evolutionary run of the first iteration. 190
Figure 7.9	Fittest individuals of the last generation of each of the 30 seeds of the first iteration. 191

Figure 7.10	Images classified as external, generated throughout the course of the evolutionary runs of the first iteration. 193
Figure 7.11	Images classified as external, generated throughout the course of the evolutionary runs of the second iteration. 195
Figure 7.12	Images classified as external, generated throughout the course of the evolutionary runs of the third iteration. 197
Figure 7.13	Images classified as external, generated throughout the course of the evolutionary runs of the fourth iteration. 198
Figure 7.14	Images classified as external, generated throughout the course of the evolutionary runs of the fifth iteration. 199
Figure 7.15	Images classified as external, generated throughout the course of the evolutionary runs of the sixth iteration. 200
Figure 7.16	Images classified as external, generated throughout the course of the evolutionary runs of the seventh iteration. 202
Figure 7.17	Images classified as external, generated throughout the course of the evolutionary runs of the eighth iteration. 203
Figure 7.18	Images classified as external, generated throughout the course of the evolutionary runs of the ninth iteration. 204
Figure 7.19	Images classified as external, generated throughout the course of the evolutionary runs of the tenth iteration. 205
Figure 7.20	Images classified as external, generated throughout the course of the evolutionary runs of the eleventh iteration. 206
Figure 7.21	Images classified as external, generated throughout the course of the evolutionary runs of the twelfth iteration. 207
Figure 7.22	Images classified as external, generated throughout the course of the evolutionary runs of the thirteenth iteration. 209

LIST OF TABLES

Table 4.1	Parameters of the Genetic Programming engine. 70
Table 4.2	Haar cascade classifier training parameters. 82
Table 4.3	Parameters of the Evolutionary Computation engine. 85
Table 4.4	Parameters of the Genetic Programming (GP) engine. 92
Table 4.5	Detection parameters. 92
Table 4.6	Training parameters. 105
Table 4.7	Detection parameters. 105
Table 4.8	Evolutionary Computation engine parameters. 107

Table 5.1	Haar training parameters. 117
Table 5.2	Evolutionary Computation run detection parameters. 117
Table 5.3	Performance tool parameters. 120
Table 5.4	Parameters of the Genetic Programming engine. 124
Table 5.5	Results attained by the initial classifier by all the framework classifiers in three independent test datasets. 126
Table 5.6	Results attained by the initial, the external classifiers and by all the framework classifiers with different Supervisors in three independent test datasets. 127
Table 6.1	Framework parameters. 133
Table 6.2	Genetic Programming engine parameters. 134
Table 6.3	Number of EC runs necessary to find 30 valid runs, average and median number of candidates per EC run, and average and median number of candidates per valid EC run. 138
Table 6.4	Negative dataset's size along the iterations for all setups. 142
Table 6.5	Framework's classifiers performance. 145
Table 6.6	Statistical significant differences among Framework's setups. 146
Table 6.7	Control's classifiers performance. 147
Table 6.8	Comparison of statistical differences between the framework's classifiers and control's classifiers. 149
Table 6.9	Framework parameters. 150
Table 6.10	Evolutionary Computation parameters. 150
Table 6.11	Number of EC runs necessary to find 30 valid runs, average and median number of candidate instances per EC run, and average and median number of detections per valid EC run. 154
Table 6.12	Positive dataset's size along the iterations for all setups. 156
Table 6.13	Framework's classifiers performance. 157
Table 6.14	Framework parameters. 159
Table 6.15	<i>Abstract</i> engine parameters. 159
Table 6.16	<i>PhotoRealistic</i> engine parameters. 160
Table 6.17	Number of EC runs necessary to find 30 valid runs, average and median number of candidate instances per EC run, and average and median number of detections per valid EC run. 166
Table 6.18	<i>Abstract</i> and <i>PhotoRealistic</i> dataset's size along the iterations for all setups. 167
Table 6.19	Framework's classifiers performance. 168
Table 7.1	Global parameters of the framework. 175
Table 7.2	The proposed feature taxonomy is composed of five categories: colour, complexity, composition, salience and texture. 177
Table 7.3	Parameters related to the Artificial Neural Networks and their training. 178

Table 7.4	Parameters of the GP engine.	184
Table 7.5	Statistics regarding evolutionary process across iterations.	185
Table 7.6	Statistics regarding the training of the classifiers of each iteration.	212

LIST OF ALGORITHMS

Algorithm 2.1	Boosting selection algorithm.	25
Algorithm 3.1	The Evolutionary Framework for Classifier assessment and Improvement algorithm.	55

INTRODUCTION

Machine Learning (ML) research deals with the question of how to construct computer programs that automatically improve with experience. Broadly defining, we can say that a computer program is said to learn from experience E with respect to some class of task T and performance measure P , if its performance at task T , as measured by P , improves with experience E (Mitchell, 1997). For example, when the performance of a speech recognition machine improves after hearing several samples of a person's speech, we feel quite justified in that case to say that the machine has learned. The tasks associated with ML include classification, regression, detection, recognition, among others. Usually, ML involves two important components: a training dataset and a model, encoded, e. g., by a function or a set of rules. Most of the effort in ML research goes to the model part, creating new theories, approaches, algorithms, operators, parameters, which attempt to maximise the performance while making use of training data. Nevertheless, designing and optimising a training dataset is a complex and time-consuming process and also a crucial one for the success of ML algorithms (Bishop, 2006; Sung and Poggio, 1995; Wang et al., 2005b).

Although there are some aspects that we should account for, guidelines or rules are not always applicable to create a training dataset. For instance, in a face detection problem, an approach to create a face dataset could be to search images that contain faces with a certain diversity: male, female, race, illumination, pose, rotation, among others. However, regarding the non-face dataset, any image that does not contain a face can be considered as a non-face image. Therefore, it is hard to create a suitable non-face dataset. These issues pertaining the design of a dataset can lead to severe shortcomings that limit the performance of classifiers and its applicability.

1.1 THE HYPOTHESIS

Evolutionary Computation (EC) is a field of Artificial Intelligence (AI), inspired by the Darwinian (Darwin, 1859) principles of the natural process of evolution, for problem solving purposes. These computational models are usually referred to as Evolutionary Algorithms (EAs). Even though there are different types of EAs, they all share some key aspects: (i) Selection – individuals that are better adapted to their environment have a higher probability of surviving and reproducing; (ii) Inheritance – the descendants inherit characteristics from the progenitors; (iii) Variation – attained by the recombination of the genetic code of the progenitors and by the introduction of random changes in the genetic code, i. e. mutations. In practical terms, if we define a problem as a triple

of: input, model and output; **EC** is usually applied to one (or more) of these types of problems: (i) optimization – the objective is to find the inputs used by a defined model that produce a desired output or a description of it; (ii) modelling – we want to discover a model that maps the inputs to the outputs; (iii) simulation – we know the inputs and the model but we want to explore the outputs (Back et al., 1997; Eiben and Smith, 2003). In the **ML** context, **EAs** have been used to evolve solutions for classifiers parameters, thresholds, features, feature selection for classification, the classifier itself, among others.

In a previous work (Machado et al., 2012b), we introduced a framework that combines a Genetic Programming (**GP**) image generation system (Machado and Cardoso, 2002) with a state of the art Face Detection (**FD**) approach (Viola and Jones, 2001). The **GP** engine evolved images that were incorrectly identified as faces by the **FD**. Later, these images were added to the negative dataset, and the classifier was retrained. The experimental results obtained by the classifiers trained with a negative dataset augmented through the addition of images evolved in a single evolutionary run, indicated that statistically significant performance improvements could be accomplished.

The working hypothesis is that **EC** can be used to assess and improve the performance of classifiers by evolving new training instances. Without loss of generality, we resort to image classification problems to illustrate how this may be achieved. In simple terms, the framework we propose proceeds as follows:

1. Selection of a starting dataset;
2. N framework iterations start; C classifiers from the Classifier System (**CS**) module are trained based on the available instances; each classifier can be trained with a different order of training instances, different parameters, or with different **ML** algorithms;
3. For each C , E independent **EC** runs are executed to generate instances of a pre-defined class; each C classifies each evolved instance; the fitness of the instances depends on the results of the classification task;
4. The **EC** runs stop when a termination criterion is met (e. g. achieving a predefined number of generations or attain a certain fitness value);
5. A Supervisor selects and filters instances gathered from all **EC** runs, updating the training dataset;
6. The process is repeated from step 2 until the termination criterion is met (e. g. attaining a predefined number of framework iterations, a number of instances, or a certain performance value).

The **EC** runs tend to evolve images that do not resemble the target class and are classified incorrectly. Thus, the evolved images explore shortcomings of the classifier.

The proposed framework could arguably allow to: (i) dynamically assess the classifier; (ii) and improve its performance by automatically adjusting the dataset with new generated instances. This way, several research questions arise:

- **How to create a suitable fitness function?** The framework relies on an automatic fitness assignment scheme based on values from the **CS**. In some classification tasks, such as **FD**, the outputs are binary and, consequently, insufficient to provide suitable information to create a smooth fitness landscape and properly guide evolution. How to design a proper fitness function capable of guiding evolution and exploit the **CS**'s shortcomings?
- **How should the Supervisor select and filter individuals per **EC** run?** An **EC** run can yield a considerable number of individuals to be added to the dataset. How to select suitable individuals? After selecting the suitable ones, should we accept all of them? Should we create a limit per **EC** run? A limit per framework iteration? Should we discard equal or similar individuals? How to determine the similarity between examples (e. g. images)? How should we merge the individuals from multiple **EC** runs?
- **How to compensate the unbalance that will be created?** The proposed framework generates instances for different classes, but it can generate more for one class than for the other, which can lead to class unbalance. Is there a way to overcome this potential problem?
- **What are the necessary conditions for the framework to succeed?** The approach improves the performance of the **CS** by evolving instances that are misclassified, adding them to the dataset, and retraining the classifier. An immediate condition for the success of this process is that the **EC** run has to be able to evolve instances that are misclassified by the **CS**. However, how can we guarantee that those examples will improve the performance? What other components, such as the Supervisor module or even the **CS**, can interfere?

1.2 OBJECTIVES

To answer the research questions we attend to several objectives:

- Acquire knowledge regarding the state of the art in instance gathering, selection and generation, in order to support the development of the framework and its analysis;
- Propose and develop a framework that uses **EC** to assess and improve a classifier performance via synthesis of new instances;
- Develop fitness functions suitable for the **EC** module of the framework;

- Provide methods and models to manage the instances that are generated by the framework;
- Assess the viability of the framework as an Instance Generation (IG) approach;
- Assess whether the framework is able to generate instances that improve the classifier's performance;
- Evaluate the framework with test datasets;
- Assess the generalisation of the framework by instantiating and applying it to different classification scenarios with different classifiers.

The creation of novel ML approaches and evolutionary paradigms is outside the scope of this thesis. Exhausting every possible parameter for each approach chosen is also out of the scope. Additionally, the starting point of this research is image classification tasks. We will resort to image generation engines based on GP or Genetic Algorithms (GAs), which will be integrated with image classifiers adapted, namely concerning fitness assignment, to pursue the thesis' goals.

1.3 CONTRIBUTIONS

We highlight the following contributions resulting from the work done in the context of this thesis:

Literature review: A review of the state of the art on the topics of dataset construction, instance gathering, Instance Selection (IS), IG for supervised ML approaches (Chapter 2).

An evolutionary framework capable of assessing and improving classifiers: Creation of a fully functional framework suitable to fulfil the objectives of this thesis (Chapter 3). The framework uses EC to exploit vulnerabilities of the classifier, generating instances that are incorrectly classified. These instances are used to expand the dataset in order to improve the performance of the classifiers (Chapters 5, 6 and 7).

Automatic fitness assignment schemes: For the process to be automatic, with no user in the loop, the fitness of the individuals needs to be assigned automatically. For each problem addressed in this thesis (see Chapters 4, 5, 6 and 7) automatic fitness functions were developed.

Evolutionary engines: Different EC engines, described in Section 4.1.2, were developed and used in several works presented along Chapters 4, 5, 6 and 7. We started by using a well-established GP engine, the Neuro Evolutionary Art (NEvAr), for the evolution of populations of images. Afterwards, we implemented a new one, geNeral purpOse expREssion Based Evolutionary aRt Tool (NORBERT), with different features such as novelty search and hybrid tournament selection (Chapter 7). A photorealistic

face generator based on a conventional [GA](#), named X-faces (eXploit faces), was also developed.

Image annotator: The freely available solutions for image annotation for the purpose object detection are scarce or have some limitations. A tool to assist the annotation of the face dataset in Section [4.5](#) was created to provide extra functionalities and interface improvements in comparison to the available tools.

Application to different classification scenarios: The approach was tested in object detection, and distinguishing paintings from computer generated images.

Sampling algorithms: The process of selecting a good representative set from a collection of instances is an open research topic in [ML](#). In the proposed approach, the selection of instances is also important, as such, sampling algorithms allowed us to evaluate and cluster the instances more efficiently. We defined different sampling strategies and tested them in Chapters [4](#), [5](#), [6](#) and [7](#).

Synthesis of novel examples: Since this approach synthesises instances by using [EAs](#), the entire instance distribution is converted to a search space that can be explored. Nevertheless, as concluded in Chapter [4](#), when the [EA](#) finds solutions suitable for the problem, the tendency is to generate several similar instances. We used the Supervisor module to deal this issue (Chapters [4](#) and [5](#), [6](#)). In Chapter [7](#) we resorted to a different approach: an archive algorithm was used with a multi-objective tournament selection, which provided a way to switch between fitness search and novelty search (see Chapter [7](#)).

1.4 OUTLINE

We start with the state of the art in Chapter [2](#). Starting with concepts and definitions concerning [ML](#) and datasets. Next, we address aspects concerning dataset construction. Then, we survey techniques that can improve the dataset, such as [IS](#) and [IG](#).

In Chapter [3](#) we present our framework, named Evolutionary FramEwork for Classifier assessmenT and ImproVemEnt ([EFFECTIVE](#)). Every aspect related to its definition, algorithm, parameters and pipeline are described in this Chapter.

Preliminary tests were conducted to test its viability as a generator of instances. These are analysed at the beginning of Chapter [4](#). The Chapter continues with several follow-up works, where we use [EFFECTIVE](#) to generate images of a particular type.

Based on the results attained in Chapter [4](#) and information from Chapter [2](#), tests are performed to verify if the framework could be used to improve a classifier's performance iteratively and automatically. This is covered in Chapter [5](#), where we analyse the contribution of the instances generated to the improvement of classifiers for [FD](#).

In Chapter [6](#) we further explore the framework by instantiating [EFFECTIVE](#) in a set of three experiments. First, we explore the generation of false positives to expand the negative dataset. Then we explore the generation of false negatives to expand the positive dataset. In a last set of experiments we expand the positive and negative

datasets simultaneously. For all the instantiations we analyse the performance of the classifiers in three different test datasets.

In Chapter 7, we deploy the framework in a different classification scenario, in a Computational Creativity (CC) context, where a classifier is trained to distinguish from instances generated by an EC engine and real paintings. The framework is used to generate instances and improve the dataset during multiple iterations, leading to the generation of instances of different styles on each iteration.

Finally, in Chapter 8 we perform a summary of the work done, highlighting the main contributions, drawing overall conclusions and indicating future work.

Typical **ML** approaches involve two components of the utmost importance: a dataset and a model. Most of the effort in **ML** research concerns the model. However, designing and optimising a training dataset is a complex and time-consuming process, which is crucial to the success of **ML** algorithms. In this thesis, it is essential to analyse the construction of a dataset and understand the gathering, selection and generation processes. Since it is a vast topic in **ML**, this Chapter focuses on supervised classification problems. In Section 2.1 we provide an overview of the definitions and mathematical notation related to datasets used throughout this thesis. We also introduce concepts and models of **ML** approaches mentioned in this survey of the state of the art, in Section 2.1.1. The following Sections concern the construction of datasets: Instance Gathering (Section 2.2), **IS** (Section 2.3) and **IG** (Section 2.4).

2.1 OVERVIEW AND DEFINITIONS

As we analyse the state of the art, we noticed that there is no universally accepted terminology. As such, we introduce the terms and definitions that we use in this thesis.

A dataset consists of a collection of observations that relate to a particular subject. In **ML**, dataset is the most frequent word to describe a set of observations also named as instances (Mitchell, 1997). Conversely, an instance is a single observation which can be learned or used by a model, and instances are grouped as a set to form the dataset (Provost and Kohavi, 1998). Furthermore, instances are often described as a vector of features. A feature is considered a prominent description or aspect of a particular observation of a given domain (categorical/discrete or continuous/quantitative). A model, function, or classifier, is the structure and corresponding interpretation that partially or entirely summarises a dataset for description or prediction purposes.

Based on the literature, the aforementioned terms and concepts are usually framed with different mathematical notations. We consider that a standard supervised learning problem is defined by a dataset D of the form $D = \{ \langle \mathbf{x}_0, \mathbf{y}_0 \rangle, \dots, \langle \mathbf{x}_m, \mathbf{y}_m \rangle \}$, i. e., a set of pairs instance/class, $(\langle \mathbf{x}_m, \mathbf{y}_m \rangle)$, where \mathbf{x}_m is an instance m and \mathbf{y}_m is the output of a function c , i. e. $\mathbf{y}_m = c(\mathbf{x}_m)$, for the input instance \mathbf{x}_m . The output \mathbf{y}_m maps or corresponds to a certain finite set Y of K classes ($Y \in \{0, \dots, K\}$). \mathbf{x}_m is a vector of the form $\{x_{m0}, x_{m1}, \dots, x_{mn}\}$ whose components are discrete or real-valued numbers that hold information. These are also called the features of \mathbf{x}_m , where the notation x_{mn} refers to the n^{th} feature of m^{th} instance x from the dataset D . Figure 2.1 illustrates these definitions.

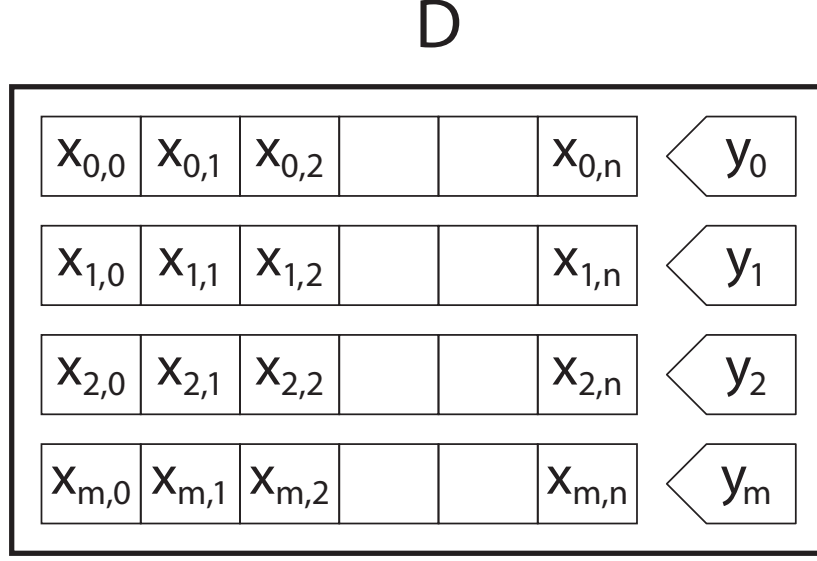


Figure 2.1: Description of a dataset D , a set of instances x_m composed of $x_{m,n}$ features and the corresponding y_m class.

2.1.1 Machine Learning Concepts and Models

This Section describes concepts and models related to **ML** approaches of the state of the art presented in the next Sections. We start by covering the definitions of supervised and unsupervised learning, and discriminative and generative approaches. Then, we present an overview of different **ML** models that are mentioned in the next Sections.

In supervised learning, as defined in Section 2.1, we have x instances as our dataset D , each labelled with an y class. The objective is to learn a function that map x to y . In unsupervised learning we only have the dataset, i. e. the instances have no class associated. Unsupervised learning approaches learn the hidden structure of the dataset.

In supervised learning, we can have discriminative and generative **ML** approaches (Y. Ng and I. Jordan, 2002). A discriminative approach learns $P(y|x)$, the posterior probability of an instance x belonging to class y . Based on the observation of a dataset D it creates a decision boundary that separates the different instances x based on their class y . To perform a prediction for a new instance, it checks on which side of the boundary it falls and assigns the class accordingly. On the other hand, generative approaches model $p(x|y)$ by learning the joint probability $P(x, y)$ and then make predictions by using the Bayes rule. These algorithms model each class, and to classify a new instance, it checks to which modelled class it relates best. Ultimately, the objective is to train models that create $x \sim p_{\text{model}}(x)$ where $p_{\text{model}} \approx p_{\text{data}}$. An example follows: if y indicate whether an instance is a dog (1) or a cat (0), then $P(x|y = 1)$ models the distribution of a dog's features and $P(x|y = 0)$ models the distribution of a cat's features. This feature

space is called the latent space, typically denoted as Z , composed of latent variables (z), variables that we have to infer rather than directly observe. Learn based on latent features can be beneficial. The latent features can be general features for other learning tasks or be incorporated in another model. Some examples include Autoencoders, Variational AutoEncoders (VAEs), Generative Adversarial Networks (GANs) and Deep Belief Networks (DBNs). After modelling $P(y)$, i. e. the class prior or the odds of belonging to class y , we can use the Bayes rule using the following formula to calculate the posterior probability: $\arg \max_y P(y|x) = \arg \max_y P(x|y).P(y)/P(x)$.

For every problem that involves a dataset, there are different ML approaches that can be used. From approaches such as K-Nearest Neighbour (KNN) or Classification And Regression Tree (CART), which are simple and computationally inexpensive for low dimensionality problems (Mitchell, 1997). If we have a high dimensionality problem, i. e. a lot of features (x_{mn}) with several instances (x) to be processed, we can resort to Random Forest (RF), Support Vector Machine (SVM) or Multi-Layer Perceptron Network (MLP). In more complex scenarios where we need to understand the representation dynamically, and when there are large amounts of instances to process, we may need to resort to Deep Learning (DL) approaches.

The KNN is a lazy learner, an approach that generalises based on the training dataset at the prediction time. Given a new input x and the instances in D it identifies the k nearest instances of x regardless of the class, using a pre-defined distance function. After collecting all k nearest neighbours, it checks what is the class from that collection that is in majority and assigns that class to the new input. As we will see in Section 2.3 this classifier is used in many IS approaches.

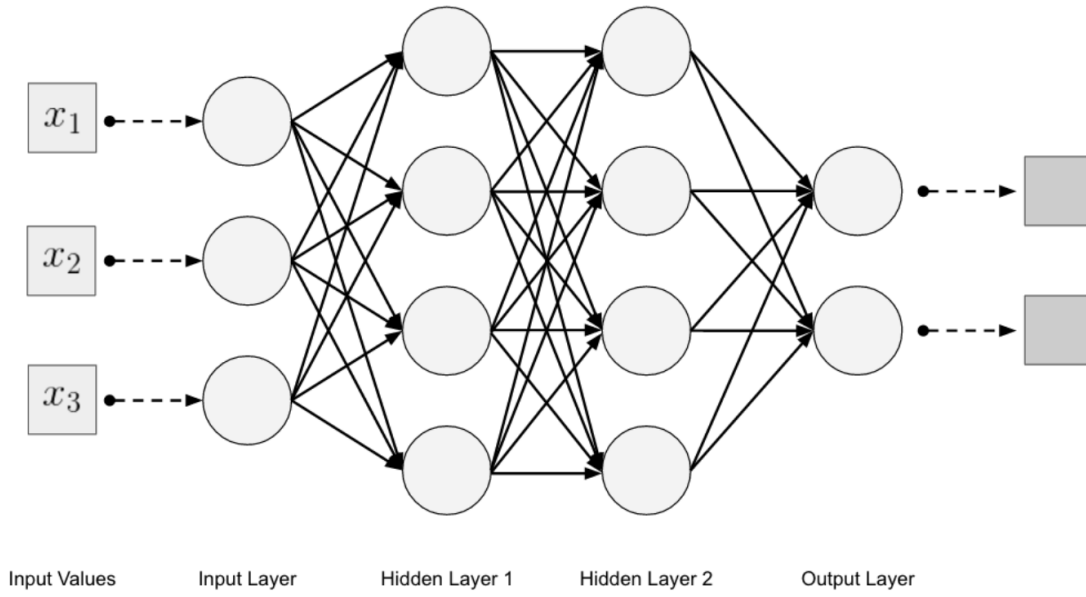
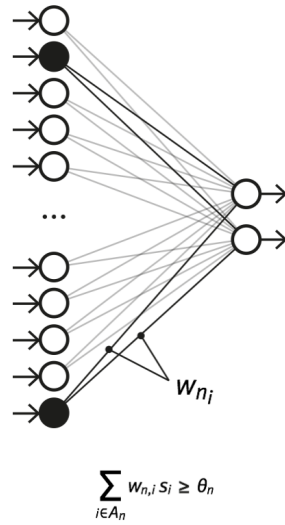
RFs are a collection of CARTs (Breiman, 2001), which are Decision Trees (DTs) algorithms that can be used for classification or regression. Based on the features (x_{mn}) of the dataset instances, a tree is built where each node is a decision based on the value of a given feature (e. g. $x_{0,0} < 0.5$). In classification, a leaf-node represents a class. In order to construct the optimal decision tree we need to compute and minimise a cost function (e. g. the Gini impurity formula: $\text{Gini}(t) = 1 - \sum_{i=0}^{Y-1} P_i^2$), where Y is the set of classes and P_i is the probability of an instance with label i being chosen. This index indicates how many instances are split by the decision. The closest to 0, the best it splits the data. We can construct the decision tree by recursively splitting it according to selected attributes as conditions. Nevertheless, if we only use a single DT for the whole dataset, the model can overfit. RF creates random subsets of features to build smaller DTs and combines them in a committee of subtrees which can, e. g., rule the decision based on the majority voting of the class.

Artificial Neural Networks (ANNs) are another popular model. Before defining an ANN we need to define the basic unit, i. e., the perceptron. A perceptron takes an input vector x , multiplies by some weights w , sums the result and applies a non-linear function to obtain an output (Van Der Malsburg, 1986). This process, known as forward pass can be defined by $\text{output} = a(\sum_{i=0}^N (x_i * w_i) + b)$ which can be redefined

in the form of matrixes as $\text{output} = a(\mathbf{WX} + \mathbf{b})$ with $\mathbf{X} = \{x_0, x_1, \dots, x_n\}$ and $\mathbf{W} = \{w_0, w_1, \dots, w_n\}$. The non-linearity is made by an activation function a . The activation function is important because most of the real-world problems datasets are non-linear and in order to make a decision boundary on such data it becomes necessary to be able to operate in non-linear spaces. Typical activation functions include the *Sigmoid*, *Tanh* and Rectified Linear Unit (**ReLU**).

We can combine multiple perceptrons to form a multi-output perceptron which has several perceptrons connected, where the inputs are linked to the outputs directly. If we stack layers of multi-output perceptrons, we create a **MLP** which has an input layer, at least a hidden layer and an output layer (e. g. see Figure 2.2). The **MLP** falls into the category of feedforward network, where the input instances are fed to the network and are transformed into an output (Mitchell, 1997). In order to train a neural network we need to define a loss function, which in the case of a **MLP** can be defined as: $\text{loss} := J(\beta) = \frac{1}{\#D} \sum_i^D \text{loss}(f(x_i; \beta), y_i)$ with $\beta = W_0, W_1, \dots, W_n$ and x_i, y_i a pair of the training dataset. Thus, we need to minimise the loss function by updating the weights (β) for the objective ($J(\beta)$), which can be accomplished using gradient descent by calculating the loss at each step and moving towards the maximum descent, until it converges. In Stochastic Gradient Descent (**SGD**) the weights are initialised randomly, for each training instance the loss gradient is computed and the weights are updated with the rule $\beta := \beta - \alpha \Delta_\beta J(\beta; x_i, y_i)$ where α is the learning rate (Rumelhart et al., 1986). To calculate the gradient we can use backpropagation which calculates the error between the target and the output using the chain rule. The chain rule provides a way for calculating the derivative of the composition of two or more functions as the product of functions, i. e. $(f \circ g)' = (f' \circ g) \cdot g'$. The goal with backpropagation is to update each of the weights in the network, making the actual output to be closer to the target, and minimising the error for each output neuron and for the network as a whole. Thus, with backpropagation we optimise the weights so that the neural network can learn how to map arbitrary inputs to outputs correctly.

Similar to an **ANN**, a Sparse Network of Winnows (**SNoW**) (Carlson et al., 2004) is a sparse network of linear units with two layers, the input layer and n target nodes. The layers are linked through weighted edges as shown in Figure 2.3. The learning is performed by the Winnow's update rule, which is a multiplicative one which contrasts with the perceptron update rule which is an additive one (Carlson et al., 2004). Each update is performed using a promotion and demotion parameter as follows: (i) let $A_i = i_1, \dots, i_m$ be the set of active features (input nodes) in a given example linked to the target node t , and let s_i be the strength associated with feature i in the example; (ii) if the algorithm predicts negative, i. e., $\sum_{i \in A_t} w_{t,i} s_i < \theta_t$, and the groundtruth label is positive, the weights of active features in the current example are *promoted*: $\forall i \in A_t, w_{t,i} \leftarrow w_{t,i} \cdot \alpha_t^{s_i}$; (iii) if the algorithm predicts positive, i. e., $\sum_{i \in A_t} w_{t,i} s_i \leq \theta_t$, and the groundtruth label is negative, the weights of active features in the current example are *demoted*: $\forall i \in A_t, w_{t,i} \leftarrow w_{t,i} \cdot \beta_t^{s_i}$. To perform prediction, a weighted

Figure 2.2: Overview of an **ANN** with 2 hidden layers.Figure 2.3: Overview of a **SNoW** classifier.

sum of activation functions of the target nodes is used. Thus, the target node with the highest predictive value for a given instance is selected as the final prediction.

In the generalisation of a **ML** approach, a decision boundary is created, which separates the different classes in the instance's space. This decision boundary can be de-

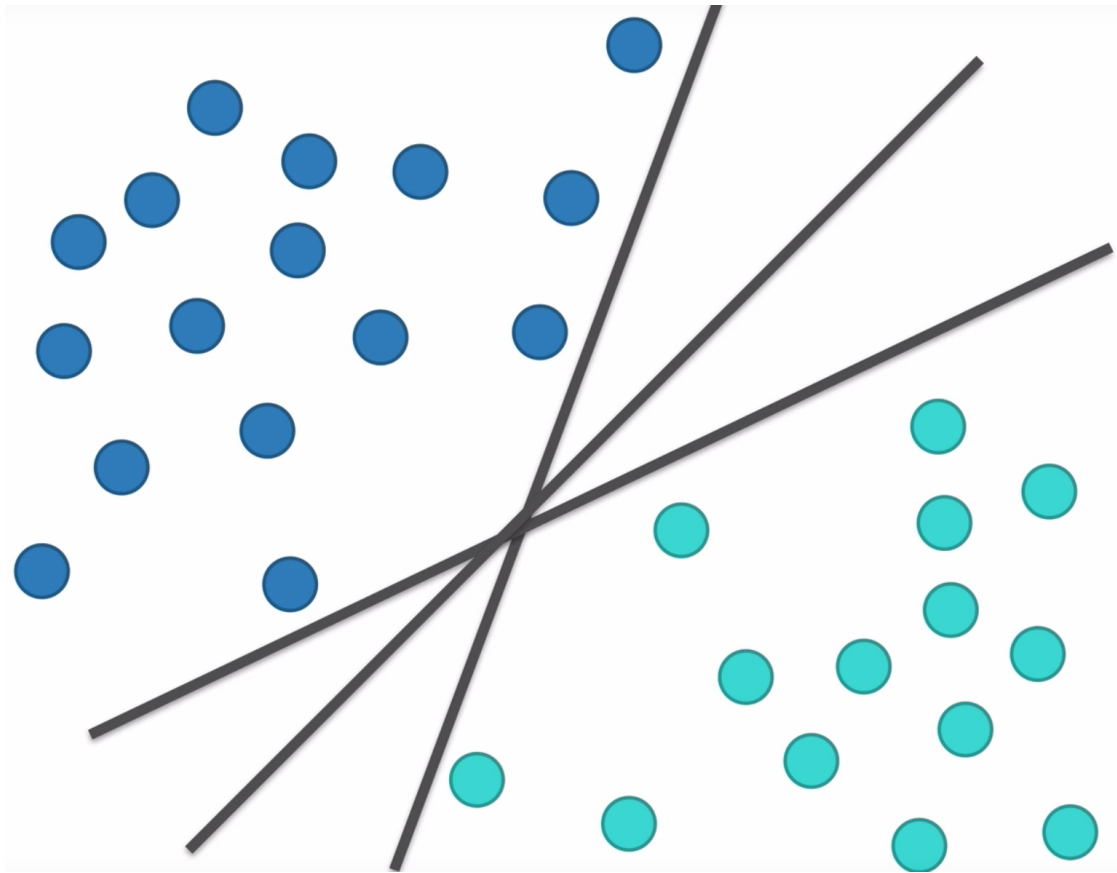


Figure 2.4: Example of different decision boundaries between two different classes represented in blue and green (image from <http://doditsuprianto.blogspot.com/2018/06/support-vector-machine.html>).

defined in various ways, e. g. we can see different decision boundaries in Figure 2.4 in a 2D feature space with two classes. If we maximise and optimise the margin, we are increasing the likelihood of a new instance x to fall into its correct class. Cortes and Vapnik (1995) created an approach that maximises this boundary at training time, the SVM.

An example of such boundary is shown in Figure 2.5. The points that define the margin and are close to the opposing class are called support vectors. We define the decision boundary as the hyperplane in the midpoint between the support vectors, i. e. the middle of margin. So the underlying idea is that the best split between the data is found by maximising the distance between the support vectors and the hyperplane. The SVMs maximise the margin by solving a constrained optimisation problem. The formulation for the SVM is the following: $w \cdot x + b \geq 0$, where w are the weights, x the instance's vector of features and b a constant. The objective is to find an hyperplane γ that separates the classes and maximises the distance between the support vectors

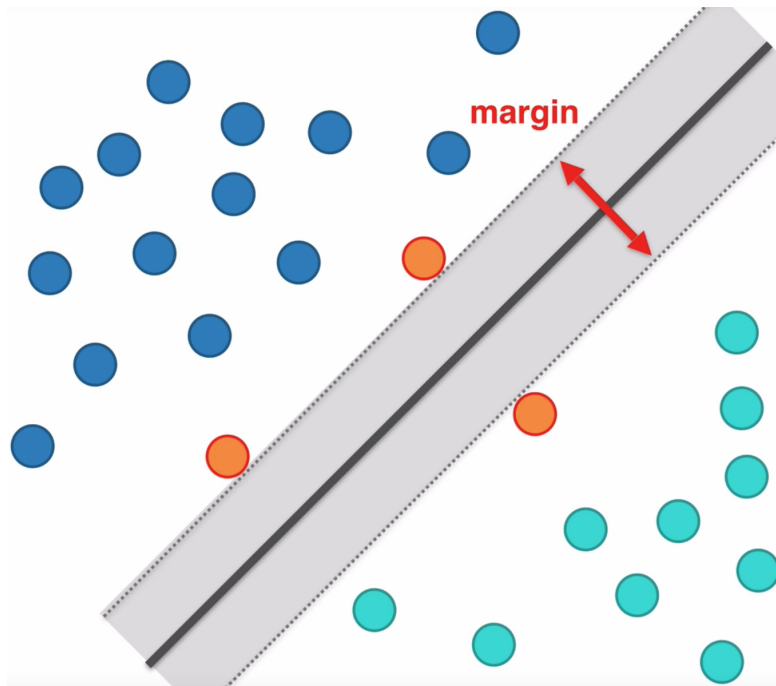


Figure 2.5: Maximised margin using SVM instances of the different classes are in blue and green. Orange points are the support vectors, the line is the hyperplane and the distance between the support vectors of opposite classes is the margin (image from <http://doditsuprianto.blogspot.com/2018/06/support-vector-machine.html>).

and that hyperplane, i. e., $\max_{w, \gamma} \gamma$ that $\forall i, y_i(w \cdot x_i + b) \geq \gamma$. If the dataset is not linearly separable SVMs address this problem by defining a function that transforms the data into a high dimensional space and finds the hyperplane that separates the classes (see Figure 2.6). Furthermore, to avoid computation in an higher dimensional space, it resorts to the kernel trick. Using the kernel trick, we take the inputs vectors x in the original space without feature transformation (Cortes and Vapnik, 1995).

Adaboost was created based on the idea that simple learners combined can build a robust classifier (Freund and Schapire, 1997). The algorithm starts by assigning a weight of equal value to all the instances of the dataset. Next, it creates simple classifiers for training and testing on the training dataset (second image of the first row of Figure 2.7). It selects the classifier with the lowest weight error and updates the instances' weights (third image of the first row of Figure 2.7). Instances misclassified have their weights increased whereas the ones correctly classified have their weights diminished. The process iteratively and similarly adds simple classifiers until a termination criterion is met. The final classifier is a weighted combination of the simple classifiers selected along iterations (last image of the second row of Figure 2.7).

The quality of the instances in the dataset impacts this approach and this simple version can underperform if we train with a noisy dataset. One way to overcome this

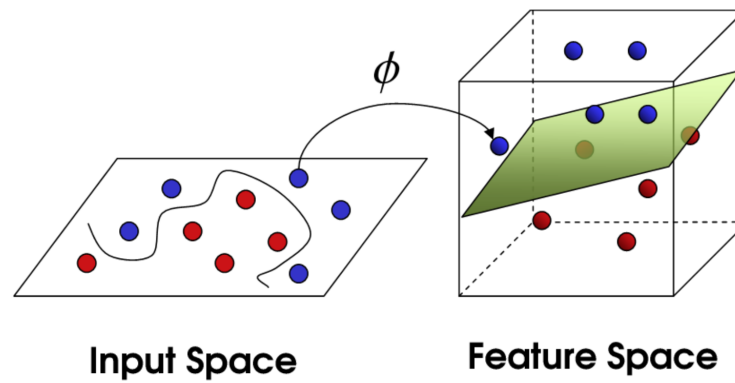


Figure 2.6: Mapping from the input space to the feature space in order to find the optimal hyperplane (image from <http://doditsuprianto.blogspot.com/2018/06/support-vector-machine.html>).

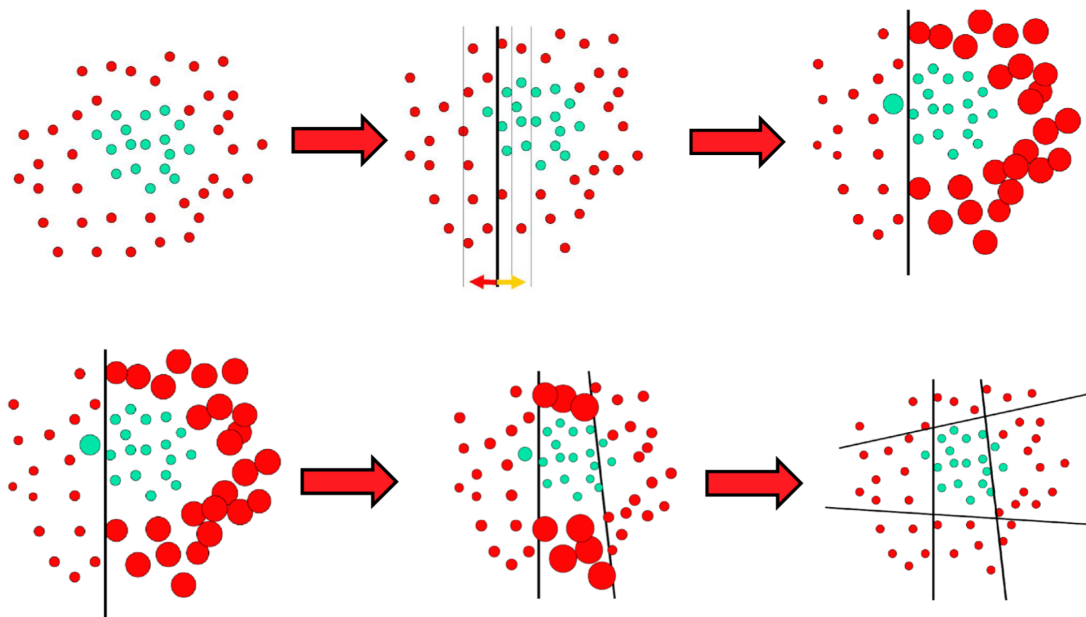


Figure 2.7: Illustration of the training of an Adaboost classifier with linear classifiers. In this case, 2D instances of two different classes are shown with different colours. The process is shown from top left to bottom right.

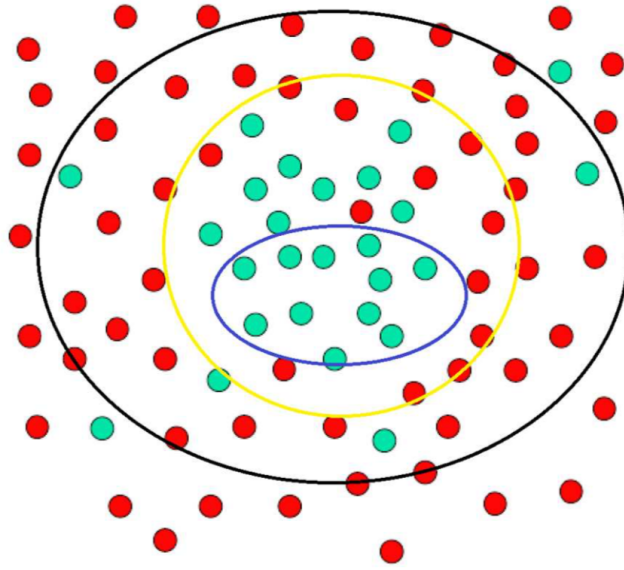


Figure 2.8: Example of the distribution of instances when using a stage classifier. Each circle groups the instances that are used for training at each stage.

is to use decision trees with a stage mechanism where each of the stages is trained with a part of the instances as shown in Figure 2.8 (Breiman, 2001; Viola and Jones, 2001).

Deep Learning (DL) models have recently received a lot of attention, due to technology advancement in terms of computational power, namely Graphics Processing Unit (GPU) power. The deep architectures contrast with shallow models regarding the number of layers it uses, typically more than two (Hinton, 2006). An essential aspect of DL models is the ability to provide mechanisms for feature extraction and learning since shallow approaches usually rely on feature engineering.

Convolutional Neural Networks (CNNs) are a type of ANNs, i. e., Deep Neural Networks (DNNs), that have been used successfully in different classification and recognition tasks (LeCun et al., 1998). The main characteristic of a CNN is the usage of special layers, such as convolutional and pooling layers, which provide feature extraction and dimensionality reduction in training (Fukushima, 1980). The convolution layers perform convolution of matrixes and act as filters applied to the inputs. After the convolution layers, it is usual to apply an activation layer, i. e., layers that apply non-linear activations typically using ReLU. The max-pooling layers extract values, i. e., in this case, the maximum values of a given region of the previous layers, and reduce the dimensionality of the matrix that comes from the previous layer. Dropout is a regularisation layer that adds a probability of certain weights connected to the neurons of being deactivated during training, making the network explore different paths for generalisation. These types of layers function as feature extraction operations. After

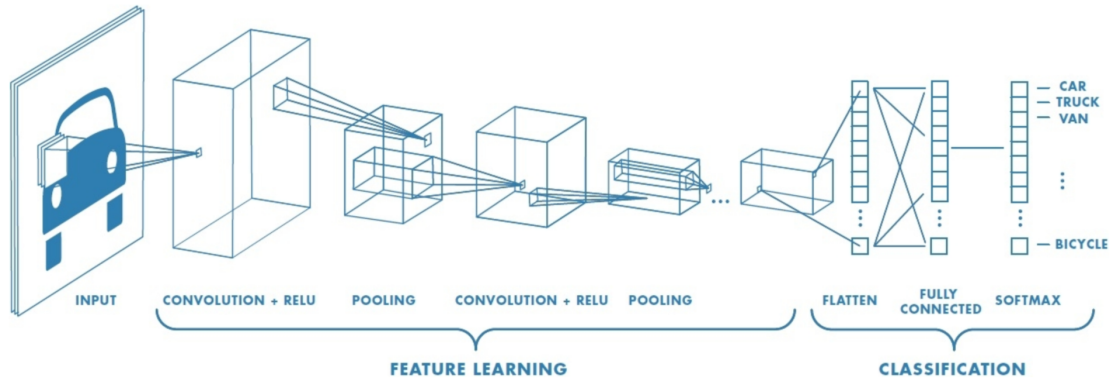


Figure 2.9: Example of a **CNN**. The input instance in this case is an image and the output is the class to which it belongs. The image is processed by the convolutional layers, activation layers and max pooling layers in the feature learning phase, and afterwards fully connected layers perform classification (image from <https://www.mathworks.com/discovery/convolutional-neural-network.html>).

the feature learning layers, the final layers are normally fully connected. Figure 2.9 shows an example of such architecture. It is also a feed-forward network so it can be optimised using backpropagation. In recent years, it has consistently attained good results in several supervised image related classification tasks (Krizhevsky et al., 2012; Szegedy et al., 2015).

Recurrent Neural Networks (**RNNs**) are also feedforward networks that offer a compact, shared parametrisation of a series of conditional distributions. These types of networks are suitable to deal with time series or sequences. As shown in Figure 2.10, each hidden layer is connected to other hidden layers. We can loop over a sequence of hidden layers each with a certain state, meaning that the state of the hidden layer of step $t - 1$ is the input of the hidden layer at time t . Figure 2.11 displays an expansion of the **RNN** that illustrates that behaviour. The parameters are shared among the layers and in that way we can compute a function of n time steps. This architecture also enables the use of backpropagation to train and optimise all the weights. Nevertheless, it has a problem of the vanishing gradient, where the backpropagation of the error in the **RNNs** gets smaller as we increase the number of steps to model (Hochreiter and Schmidhuber, 1997).

To solve the problems of **RNNs**, a new type of networks called Long Short Term Memory networks (**LSTMs**) were introduced by Hochreiter and Schmidhuber (1997). The architecture of the network is different from a **RNN** using different gates to process inputs and deal with changes over time: the input gate, the forget gate and output gate (as shown in Figure 2.12). The input gate protects the unit from irrelevant input events. The forget gate is used to forget previous calculated steps. The output gate controls what should be exposed to the next unit. This way, the network implements a selective

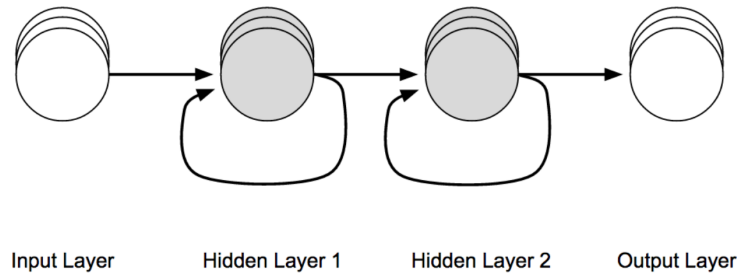


Figure 2.10: Example of a **RNN** where each hidden layer has recurrent connections (image from <https://deeplearning4j.org/lstm.html>).

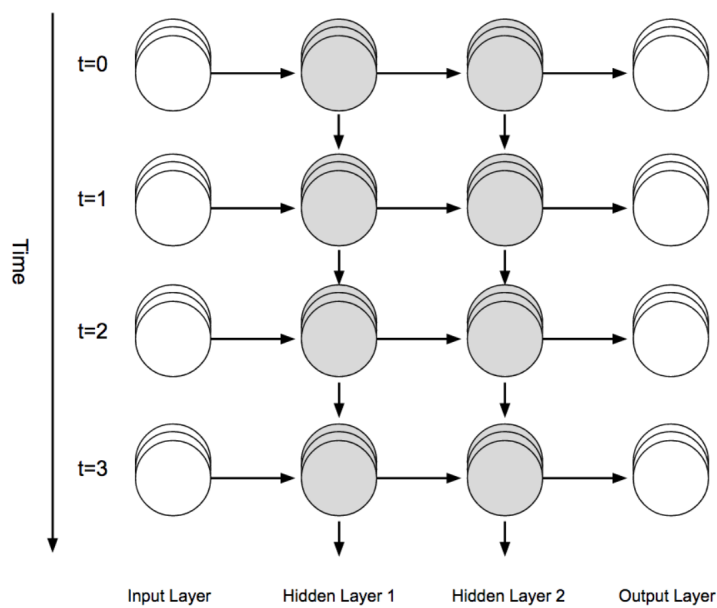


Figure 2.11: Example of an expanded **RNN**. Each hidden layer t is connected with the $t + 1$ hidden layer (image from <https://deeplearning4j.org/lstm.html>).

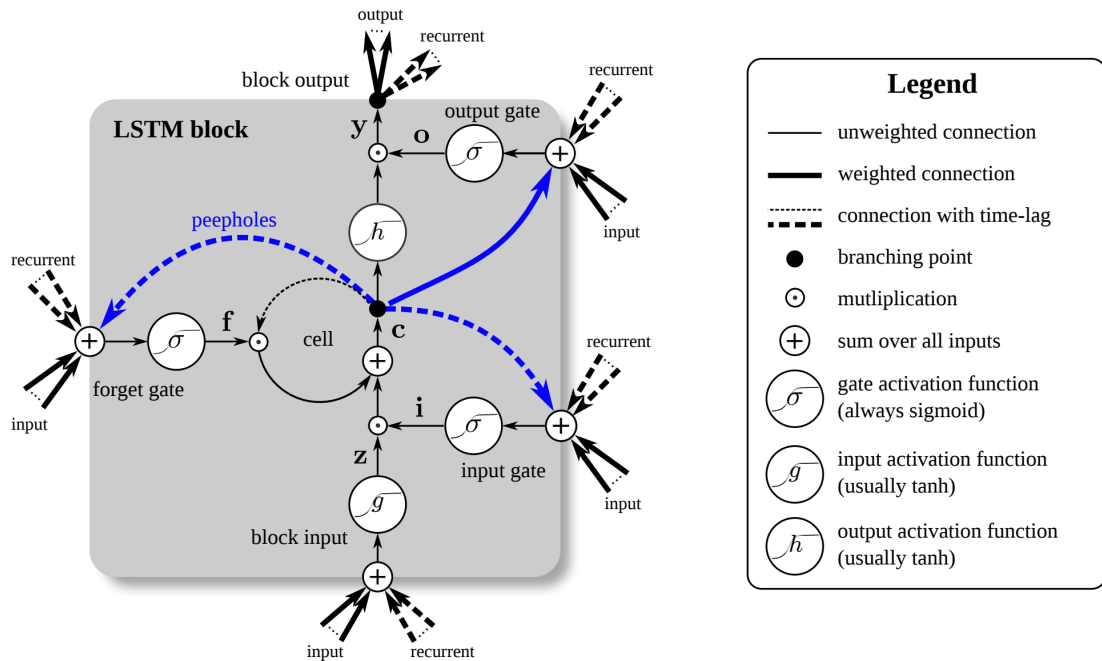


Figure 2.12: **LSTM** overview with the different connections and gates (image from <https://deeplearning4j.org/lstm.html>).

and conditional memory mechanism, capturing dependencies throughout the different time steps.

Hinton and Zemel (1994) introduced the Autoencoders, networks with a structure as depicted in Figure 2.13. It is an unsupervised generative **ML** approach. Hence, it learns a lower-dimensional feature representation from unlabelled instances. The network is composed of two phases: encoding and decoding. In the encoding phase the input instance x is processed and compressed into a lower dimensional space, which is commonly called latent space, denoted as Z . This Z should be of a lower dimension so that the encoder captures the hidden structure of the inputs, i. e., the significant factors of variation in the data. By manipulating the z values of the Z space, in the decoding phase, we can generate an output x' of the same size of the input x . This network is also feed-forward, and the objective is to minimise the error between the input x and x' (e. g., using a least squared error loss function). After training the Autoencoder, the learned latent space can be used by a discriminative model. This can be achieved by discarding the decoder phase and use the latent variables as inputs for a discriminative model. Since a generative model learns to model data first, we can achieve with few examples a strong set of features. In certain cases the strong features can be more suitable for prediction than training a discriminative approach on the same training dataset.

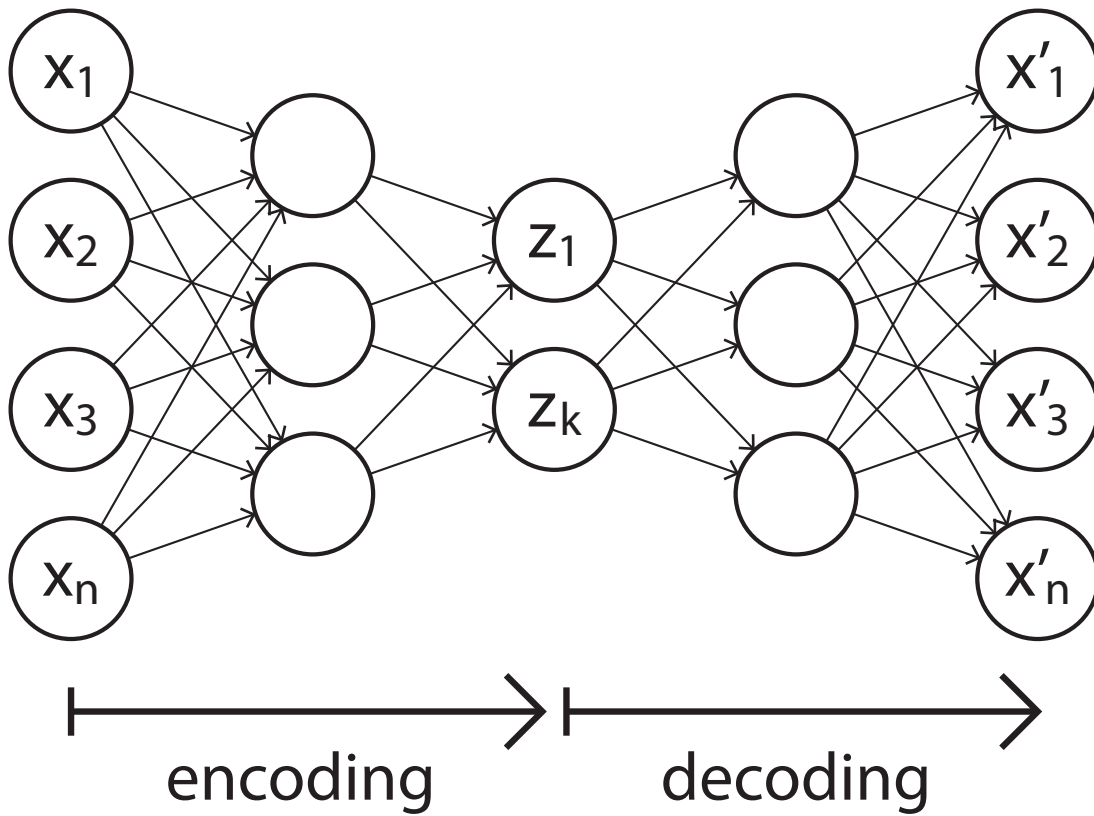
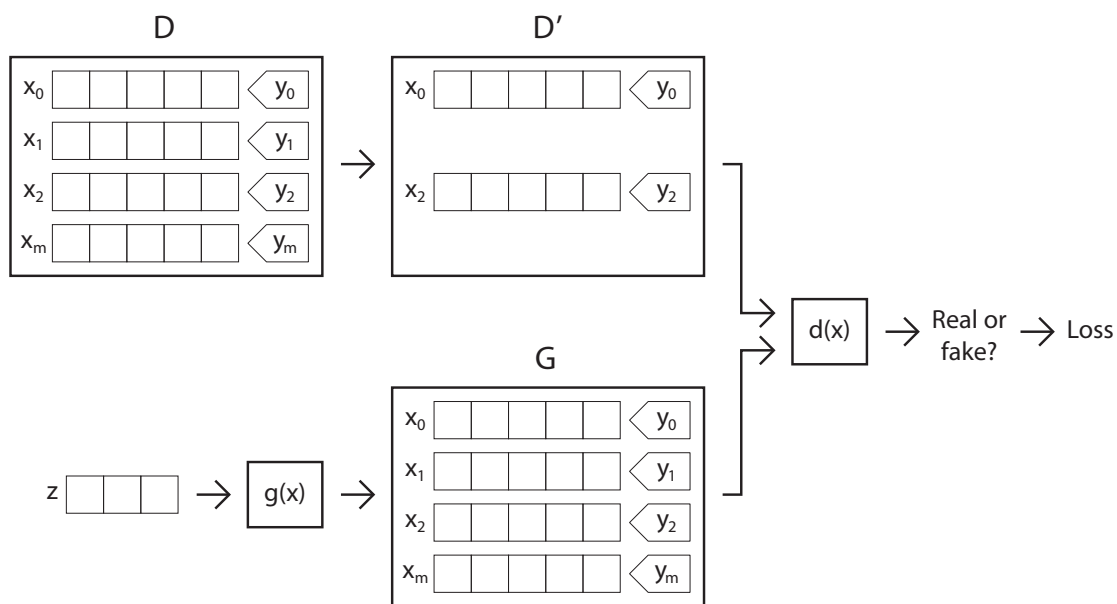


Figure 2.13: Autoencoder overview. The inputs x are encoded to latent variables z and, in a feedforward fashion, decoded to x' output values.

Figure 2.14: **GAN** overview.

Based on the way that an Autoencoder works, researchers posed the question “how can we generate new instances from the training dataset distribution?”. To do it, we need to model the density distribution of the training dataset explicitly. Moreover, the problem is that even if we sample from Z to generate x , we do not know the distribution to which it belongs without modelling: $p_\alpha(x) = \int p_\alpha(z)p_\alpha(x|z)dz$, which is considered intractable. Kingma and Welling (2013) address this modelling problem and present means that allow optimising a lower bound for the density function, providing a way to train a generative model known as **VAE**. The approach is to train two networks, one for encoding and another for decoding. The idea is to reconstruct the sample of the encoder network but following the distribution of the decoder. The encoder models $p(z|x)$ and outputs the mean and the diagonal covariance of $(z|x)$, where z is obtained by a Gaussian distribution and x is an input instance. The latent factors z of the encoder are sampled for the input x . Then, the latent factors pass through the decoder network which outputs the mean and diagonal covariance of $(x|z)$. Based on the decoder output, it is possible to sample x' and see if it maximises the likelihood lower bound. Backpropagation is used to optimise this process (Kingma and Welling, 2013).

In **VAE**, the idea is to explicitly model the density distribution to generate new samples from that distribution. Goodfellow et al. (2014) present **GAN**, an approach that does not need to model the density function explicitly. The overall concept is to train two networks: a generator and discriminator (as shown in Figure 2.14). The generator uses a generative approach, e. g., a decoder network, that samples from a distribution

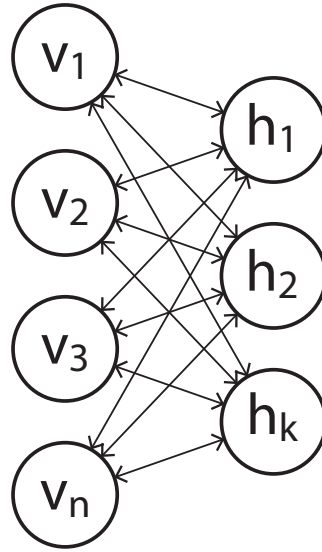


Figure 2.15: Overview of a [RBM](#) with one layer, a symmetrical bipartite and bidirectional graph. v_n are inputs and constitute the visible layer. h_k are the hidden values of the [RBM](#) to which v_s are connected via weighted and bi-directional connections.

Z and generates instances x that are considered fake, i. e., not real instances from the training dataset. The discriminator network follows a discriminative approach as it is trained to distinguish between instances from the training dataset and instances generated by the generator. The whole process comes together as a min-max game, where the discriminator has to correctly distinguish the real instances from the instances generated by the generator, and on the other hand, the generator has to trick the discriminator network, i. e., generating instances that are similar to the distribution of the real dataset. This process of using a model that learns to generate instances that exploit vulnerabilities in another model is also called adversarial learning. The instances generated by this process are called adversarial instances.

The Restricted Boltzmann Machine ([RBM](#)) (David E. Rumelhart, [1986](#)) is a type of unsupervised learning model with two layers: the visible layer and the hidden layer (as shown in Figure [2.15](#)). Each node in the visible layer is connected to every node of the hidden layer. It is restricted because no two nodes in the same layer share a connection. In a forward pass, the hidden layer models the hidden structure of input features and in the backward pass it reconstructs the input. [RBMs](#) can learn high-level features with unsupervised training. These are the building blocks for the Deep Belief Networks ([DBNs](#)).

[DBNs](#) are composed of layers of [RBMs](#) (Bengio et al., [2007](#); Hinton, [2006](#)). Structurally, this means that per each layer of [RBM](#) the hidden layer serves as the visible layer to the next hidden layer (as shown in Figure [2.16](#)) allowing to learn more complex representations of the input dataset. These networks have two phases: pre-training and

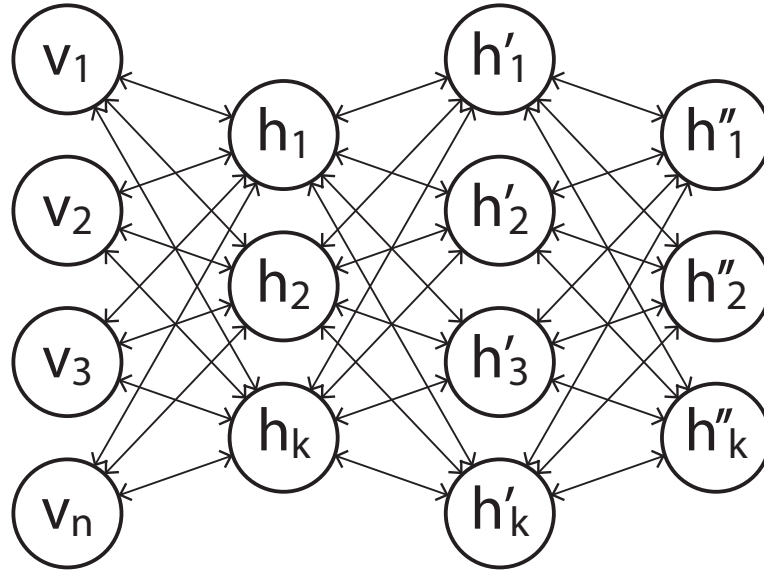


Figure 2.16: Overview of a **DBN** with three hidden layers. The v_n nodes are the visible layer, the input layer. The h , h' and h'' are hidden nodes. This is equivalent of stacking three **RBM**s, where the visible nodes for the h' nodes are the h nodes and the visible nodes of h'' are the h' nodes.

fine-tuning. In pre-training, instances are provided to the **DBN** in an unsupervised way, to learn patterns and model the structure of the data. After pre-training, a discriminator feed-forward network uses the layers of features as weights, and with few labelled examples, it learns how to classify the classes for a given task.

The remainder of the Chapter is divided into three Sections: instance gathering, selection and generation. The idea is to provide some guidelines for instance gathering and then survey instance selection, and instance generation approaches. The set of works presented in the following Sections is organised chronologically and grouped by common aspects. Furthermore, in each Section, we separate evolutionary and bio-inspired approaches from other methods. When a surveyed work fits several Sections, we assign it to a specific one based on its most distinguishable aspect.

2.2 INSTANCE GATHERING

Constructing a dataset depends on the data that we plan to analyse and the problem at hands. In the literature it is possible to find some guidelines about what kind of data is suitable to analyse or to solve a given problem. Most researchers resort to the same datasets for specific problems to compare the performance of the different available approaches. On the other hand, some problems require getting a dataset to train models that can perform well in a real-world scenario, with time and resources

constraints, and with reasonable and reliable performance. Considering all constraints, designing and optimising a training dataset is a complex and time-consuming process, but also a crucial one for the success of ML algorithms (Bishop, 2006; Sung and Poggio, 1995). A well-constructed dataset has an impact on the performance of the classifier. Therefore, when constructing a dataset, there are some aspects to take into account:

1. Correctness – correct labelling of the training instances;
2. Representativeness – include in the dataset, instances that are representative of each assigned class;
3. Completeness – cover the solution space for each class as much as possible.

An obvious recommendation is to do a literature survey about the problem that we have to solve, to ensure that our dataset copes with the characteristics mentioned above. In theory, our dataset should have the specified characteristics, but in practice, it is hard to ensure and control them. Possibly, the aspect that is more controllable is the correctness. That is, assuring that the instances that we gather have their class/label correctly assigned. One must avoid using a biased subset of training examples to train a classifier because the resulting classifier will probably perform poorly on the rest of the input space, i. e., it will likely result in a classifier with low generalisation. Even if we gather the most representative instances, one should include more data and cover as much solution space as possible. The amount of data needed depends on the complexity of the data that we are trying to learn. Thus, we should use a pre-labelled set of data of suitable size. If this is not possible, the next approach would be to gather and label data. For example, for handwriting classification, one could take many samples of handwritten text and have a human labelling them. The obvious downside of this approach is that it is time-consuming and requires human intervention and supervision. Nowadays, one could outsource this task to a *Mechanical Turk-like* service or build a service that produces labelled training examples, as some benchmarks nowadays do (Deng et al., 2009). Ultimately one could try to harness all the data and aim for full completeness, but engaging in such task is time-consuming, impractical and impossible for most problems. Another possibility is the generation of additional instances based on the data and knowledge of the problem that we have, i. e., perform IC. But even if we manage to gather or create all instances or a subset of them, we should try to minimise the number of instances and create an efficient way to learn and generalise from the available data, i. e., perform IS. Summarily, when building a dataset, we should take into account the characteristics mentioned earlier for gathering the instances, and then attempt selection and generation techniques to assess and improve them.

2.3 INSTANCE SELECTION

After gathering the instances for our dataset, we should analyse them, and decide whether we need all or only a subset of them. Just as some features are more useful than others, so may some instances better aid the learning process than others (Blum and Langley, 1997). Most work assumes the presence of a benevolent tutor, us users, who gives informative instances or provides ideal training datasets (Settles, 2010). However, a more robust approach involves letting the learning model select or focus on training instances by itself. Researchers have pointed out at least three reasons for selecting examples used during training. The first reason is when the ML algorithm is computationally intensive, thus making sense to learn from a smaller sample of instances for computational efficiency. The second reason is when the cost of labelling is high (e. g., when labels must be obtained from experts) but many unlabelled examples are available or are easy to create. The third reason is to increase the rate of learning by focusing the attention on the most useful examples, thus aiding search through the space of hypotheses, which has a lot to do with the representativeness aspect of the dataset mentioned in Section 2.2. The notion of representativeness can take two sides: (i) instances that are relevant due to the information and knowledge they provide about the task/problem; and (ii) instances that are relevant for the learning algorithm used for the construction of a model.

IS is based on the principles of sampling from statistics. The underlying idea in statistics is to choose a representative sample and use it to understand something about the whole population. In ML, the baseline idea is to select a part of the available dataset to construct a suitable model for a particular task. With the constraints imposed by computational resources, the central point of IS is the approximation, that is, the focus of IS is to achieve the best possible results with a selection of instances instead of using all the instances available.

An IS algorithm follows the idea of picking a subset of data from the full dataset to create a particular model for a specific task/problem. As such, IS is commonly associated with Feature Selection (FS) approaches (Blum and Langley, 1997). Some approaches combine both IS and FS, or even IS via FS (Olvera-López et al., 2010). Methods of undersampling to change the datasets by sampling from the majority class, or oversampling to repeat instances in the minority class in the training dataset, are straightforward to implement and often explored in problems with unbalanced datasets (Liu et al., 2009; Meyliana and Budiardjo, 2014).

Furthermore, besides sampling, there are links with other areas such as boosting and Active Learning (AL). Most boosting algorithms have some methods that either weight instances or select them from the available dataset to create weak learners that, when combined, create a strong learner (Freund and Schapire, 1997). An early work by Drucker et al. (1994) that investigated boosting observe that the training error curve of the original boosting algorithm has negative slope until it asymptotes to some

Algorithm 2.1 Boosting selection algorithm.

```

1: procedure BOOSTING SELECTION
2:    $w_i \leftarrow 1/N$  ▷ Initialise  $w_i$  based on  $N$  instances of  $D$ 
3:    $S' \leftarrow D, m = 0$ 
4:   while  $m < M$  do ▷  $M$  is the number of rounds
5:      $S \leftarrow \text{instanceSelection}(S')$ 
6:      $V \leftarrow \text{updateSelected}(S)$  ▷ Register selected instances updating a set of
       votes  $V$ 
7:      $w_i \leftarrow \text{update}(w_i, \text{KNN}(S))$  ▷ Update  $w_i$  using KNN over  $S$ 
8:      $S' \leftarrow \text{sample}(D, w_i)$  ▷ Sample  $S'$  from  $D$  based on  $w_i$ 
9:      $m \leftarrow m + 1$ 
10:  end while
11:   $D' \leftarrow \text{instanceSelection}(D, V)$  ▷ Perform instance selection based on the
       votes  $V$ 
12:  return  $D'$  ▷ Return the updated dataset
13: end procedure

```

value of training error. In their work, the authors make three conjectures. Firstly, the capacity of the learning algorithm increases with the increase of the training dataset size until it reaches an asymptotic value. Secondly, good algorithms will have the same negative slope characteristic, meaning that the difference between the test and training error rate should converge to zero as we increase the size of the training dataset. It is desirable that the training error curve has this negative slope so that the test error rate follows the training error rate to some small value. The same negative slope is observed in: additive algorithms, e. g. cascade architectures, that build networks incrementally; and algorithms that use queries, filters, hierarchical structures that in general adapt to the structure of the learning classifier. These factors contributed to research on the topic. For instance, García-Pedrajas and De Haro-García (2014) created a generalised version of boosting combined with an **IS** algorithm, as depicted in Algorithm 2.1.

In **AL**, the key idea is that a **ML** algorithm can achieve better results if it can pick the set of instances from which it learns (Settles, 2010). Thus, an **AL** approach requires a starting set of labelled instances to train a model and an oracle (e. g., a user that annotates data). The **AL** approach performs queries to the oracle, typically in the form of unlabelled instances. Based on the query results, the knowledge base is updated, the model is retrained, and the next set of queries are generated until a termination criterion is met (e. g., when a specific value of performance is achieved). The approach relates with **IS** since selecting the starting set of instances is an essential part of the algorithm which can minimise the number of queries to be performed to achieve a particular performance value or generate a better model using fewer resources (Fu et al., 2013; Settles, 2010).

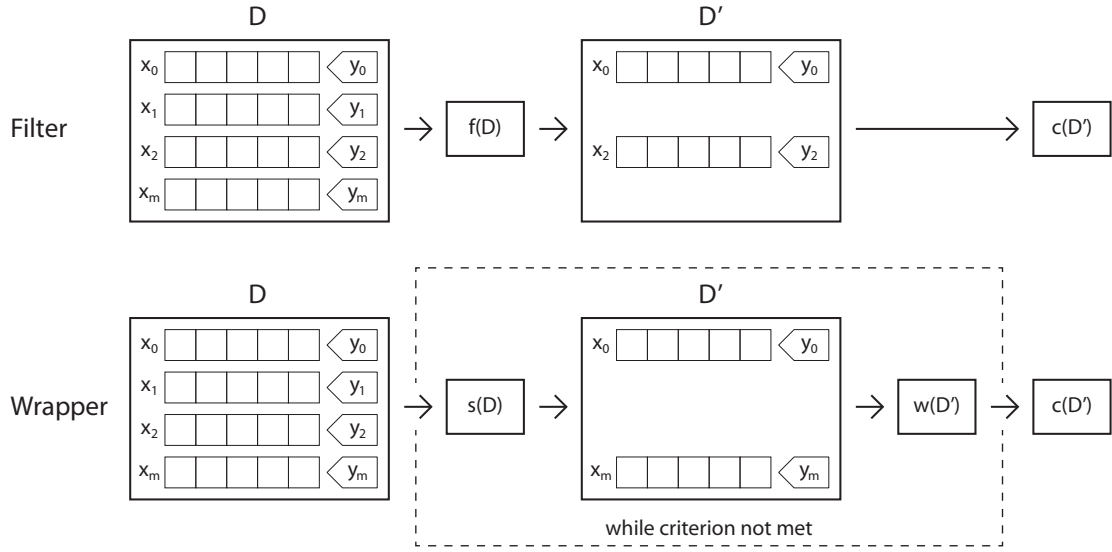


Figure 2.17: Overview of filter and wrapper [IS](#) methods. D is the starting dataset and $f(x)$ is the filter function of the filter [IS](#) method. In the wrapper [IS](#) method, $s(x)$ is the function that selects a subset of D , i. e. D' , which is evaluated by a model in the function $w(x)$. In both methods, the resulting subset D' is used in the model c as the training dataset.

Similar to the categorisation found in feature selection, we can distinguish two major classes of algorithms for [IS](#) filter and wrapper. In filter [IS](#), the instances of a given dataset D_x are selected without the intervention of the model to be trained, typically using statistical methods or other functions ($f(D_x)$). In wrapper [IS](#), the selection of the instances to create the subset is considered as a search problem with at least one evaluator ($w(D_x)$), usually, the model to be trained, searching iteratively for a subset until some criterion is met, e. g., number of iterations, the performance achieved by the model, or size of the subset. An overview is given in Figure [2.17](#). In the next Subsections, we survey filter and wrapper methods.

2.3.1 Filter

Filter methods are less common in the [ML](#) literature than wrappers but have received a lot of attention over the years (Blum and Langley, [1997](#); Jankowski and Grochowski, [2004](#); Olvera-López et al., [2010](#)).

A simple filter approach is random sampling, which consists of the random selection of a pre-defined number of instances of the whole dataset as the training dataset. This method is frequently used as a baseline approach for comparison with other [IS](#) approaches.

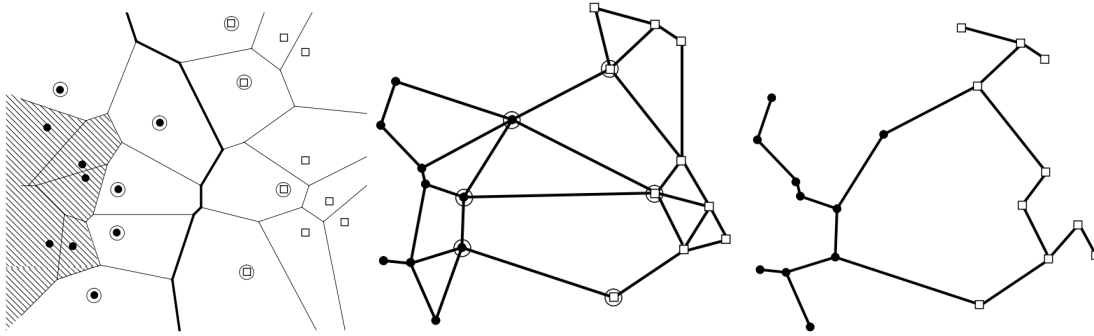


Figure 2.18: Examples of the mapping of instances using Voronoi Editing (left), Gabriel Editing (centre) and Relative Neighbourhood Graph Editing (right). Different instances are represented with dots and squares. In the left and centre images, circled instances are instances that have a neighbour instance of a different class and are therefore selected by the algorithm. In the right image, only linked instances of different classes are selected (images from Bhattacharya et al., 1992).

Most of the filter algorithms focus on selecting boundary instances based on the instances' class. Boundary instances are the instances that are closer to the decision boundary of a given model. These algorithms attain competitive performance when compared with wrapper methods (Olvera-López et al., 2010). One advantage is that, in most cases, the resulting set of instances works for different classifiers. A downside of these algorithms is the scaling up for large datasets, since most require processing each instance of the dataset more than once. Some strategies for subset stratification of the dataset try to minimise this kind of problem rather than using the whole dataset (Cano et al., 2005; García-Pedrajas et al., 2013; García and Herrera, 2009). Nevertheless, there is a trade-off between the classifier's performance and the computational cost associated with the stratification strategy.

The first algorithms based on boundary instances map all the instances and manipulate them to get a clear boundary between instances of different classes. Bhattacharya et al. (1992) use Voronoi diagrams to select relevant instances based on the rule of a 1-Nearest Neighbour where instances that have neighbours of the same class are discarded (see the left image on Figure 2.18). Since the time complexity of building Voronoi diagrams in the worst case scenario is $O(N^{d/2})$, where N is the number of instances of the dataset and d the number of dimensions, the authors concluded that it is computationally expensive for datasets with a high number of instances. Bhattacharya et al. continued working on the topic using graph theory and ended up using Gabriel graphs, resulting on the algorithm Gabriel Editing, which maps the instances as presented in the centre image of Figure 2.18. The idea is similar to the Voronoi diagram based approaches, but it limits the nodes' connections to a circular range. The authors expanded their work and tested the Relative Neighbourhood Graph Editing (RNGE),

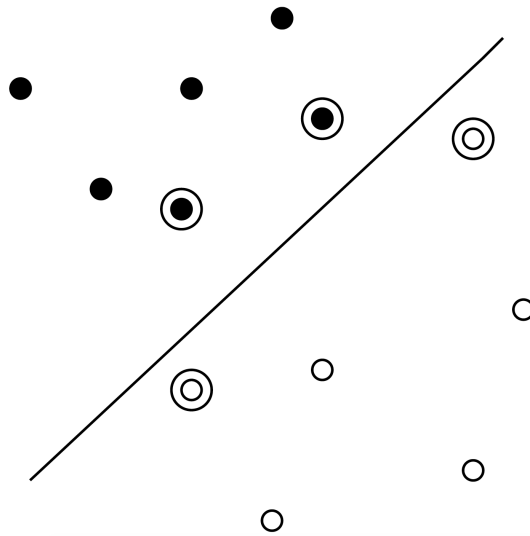


Figure 2.19: Example of boundary data, represented by the double circles (image from Hara and Nakayama, 2000).

a subset of Gabriel graphs where the distance among nodes is relative and instances have a maximum range to be connected (refer to the right image in Figure 2.18). These approaches considerably reduce the number of instances of the final dataset for training, but with the trade-off in performance degradation.

Lewis and Catlett (1994) describe an approach of heterogeneous uncertainty sampling, which filters instances based on the outcome of several probabilistic classifiers trained with small subsets of the instances of the whole dataset. The probabilistic classifiers evaluate the instances and those that have a borderline classification around 50% chance of belonging to either class are selected. The selected instances are used to train another type of classifier, in the particular case of the authors' work, a C4.5 classifier. The authors stated that the classifier trained with the selected subset of instances was more accurate than the one trained with ten times the number of instances using random sampling.

Hara and Nakayama (2000) introduce a method to select boundary instances to train MLPs. The algorithm processes all instances, picking those that are close to an instance that belongs to another class (see Figure 2.19). The analysis of the results revealed that using only boundary data results in low convergence during training. The authors combined the boundary selection with random selection, improving the classifier's performance and reducing the training time of the MLP when compared with the previous version. Riquelme et al. (2003) introduce the Pattern by Ordered Projection (POP) method, which selects boundary instances and discards the interior ones. It is based on the concept of the weakness of instances, defined as the distance, in terms of attributes, between each instance and a boundary instance. Thus, instances farth from the bound-

ary instances have a higher weakness value. A threshold for the maximum value of weakness is defined, and instances with a weakness value higher than the threshold are discarded from the selection. This method leads to the selection of boundary instances. Raicharoen and Lursinsap (2005) also pursue the idea of selecting boundary instances with the Pair Opposite Class-Nearest Neighbour (POC-NN) algorithm. The selection is based on the mean instance of each class. A mean instance is an instance calculated using the mean of every instances' features of the same class. An instance is selected if it belongs to a certain class but is close to the mean instance of another class. With a different rationale, Vishwanathan and Murty (2002) use a linear Multi-category SVM to iteratively perform dataset reduction and outlier elimination to select instances to train a KNN classifier. The approach reduces the size of the dataset by removing boundary instances and removing instances of the class which are far from the boundary. Thus, it promotes the selection of interior instances to form subsets that maximise the decision boundary between classes.

Several authors resort to clustering to aid in the task of selecting instances. Mollineda et al. (2002) introduce the Generalised-Modified Chang Algorithm (GMCA), which merges clusters of instances of the same class that are close to each other, selecting the instances near the centroids of the newly merged clusters. Sánchez (2004) presents the Reduction by Space Partitioning (RSP) algorithms, which constitute a popular set of three abstraction algorithms known as RSP1, RSP2 and RSP3. They are extensions of the Chen and Jozwik Algorithm (CJA) (Chen and Jóźwik, 1996). In CJA, the algorithm is initialised by searching the most pair-wise distant instances of the different classes in D. Then, the algorithm divides D into N subsets, depending on the number of classes. Each subset includes items closer to each of the instances selected in the initialisation. CJA continues by dividing the subset by the largest diameter. It repeats this process until the number of subsets is equal to threshold predefined by the user. Finally, it selects a mean instance (centroid) for each created subset. The class label of each centroid is the most common class in the corresponding subset. In contrast, RSP1 computes as many centroids as the number of different classes in the subset. For each split, RSP1 ensures that each class is represented on the final subset of instances. RSP1 and RSP2 differ in the split of the subsets. RSP2 follows the idea that instances from the same class should be near each other and instances belonging to other classes should be far. The split is performed on the subset with the highest overlapping degree among instances from other classes. The rationale of RSP3 is to promote class homogeneity. A cluster satisfies homogeneity if all its sub-clusters contain only instances which belong to a single class. The algorithm continuously splits the non-homogeneous subsets until they become homogeneous. Being a non-parametric algorithm is an advantage of RSP3 since it automatically determines the size of the dataset with the selected instances. The Object Selection by Clustering (OSC) by Olvera-López et al. (2008) selects boundary instances and a few interior ones. OSC divides D into clusters; the subset is constructed with boundary instances picked from non-homogeneous clusters and

interior ones from homogeneous clusters. Ougiaroglou and Evangelidis (2012) present the Reduction through Homogeneous Clusters (RHC) algorithm, a fast non-parametric algorithm for data reduction. It works by using the k-means algorithm to cluster the training dataset into homogeneous clusters recursively. The subset consists of the centroids of the final clusters. The work by Ougiaroglou and Evangelidis (2012) reports that RHC achieved a better performance than RSP3.

There is also the concept of prototype instances, i. e., instances that are created to represent a group of instances. The work by Chang (1974) is an example of this type of approach. It repeatedly merges the two nearest neighbours of the same class into an average prototype instance, as long as this merge does not increase the error rate of the training dataset. One drawback of Chang's method is that it may yield prototypes that do not characterise well the training set regarding generalisation. Paredes and Vidal (2000) presented the Weighting Prototypes (WP) method that uses gradient descent for computing weights for each instance, where the nearest neighbours have a higher weight, discarding instances that have weights higher than a certain threshold. Salzberg et al. (1995) proposed a novel and more extreme version of the above methods, which focuses on the decision boundary and ignore the training set. To accomplish this, a small set of prototypes is designed to perform the desired decision boundary by synthesising best-case prototypes. Olvera-López et al. (2008) created Prototype Selection by Relevance (PSR), which computes the relevance of each instance based on its average similarity, i. e., the most similar instance, in terms of features, among those of the same class. PSR selects a percentage of relevant instances, pre-determined by the user, and, based on those choices, the algorithm selects the boundary instances. Furthermore, instances that are similar but belong to different classes are also selected. A combination of prototype instances and clustering was explored by Ougiaroglou and Evangelidis (2016), who further developed their approach, RHC to remove noise and mislabelled data, as well as to smooth the decision boundaries between classes. First, the authors propose an editing algorithm called Editing through Homogeneous Clusters (EHC). Then, they extend the idea by introducing a prototype abstraction algorithm that integrates the EHC mechanism, and is capable of creating a small noise-free representative dataset of the initial training instances. The algorithm is called Editing and Reduction through Homogeneous Clusters (ERHC). EHC and ERHC are based on a fast and parameter-free iterative execution of the k-means clustering algorithm, which is used to create homogeneous clusters. Clusters with a single item are removed. ERHC summarises the instances of the remaining clusters by selecting the mean instance per each representative dataset. EHC and ERHC were tested on several datasets and the results show that both are fast algorithms and without underperformance. In the results reported, ERHC selects fewer instances than EHC.

2.3.2 Wrapper

The main idea of an [IS](#) wrapper method is to use a selection criterion based on the accuracy obtained by a model, with focus on the notion that instances that do not contribute to our model should be discarded. This type of algorithms uses an iterative process and are normally composed of an evaluator and a selection strategy. Wrapper methods received a lot of attention from the research community over the years. In this Section we review some of the prominent approaches and explore their connections with other topics such as boosting methods, unbalanced dataset problems and classifiers.

Incremental and lazy learners require that the instances present in the dataset are meaningful and not noisy. Most of the research reported in this Section uses [KNN](#) as its classifier, since its lazy learning methodology makes it easy to set up and scale up with reasonable performance, making it a good classifier to evaluate the generalisation boundary (Toussaint, [2002](#)). The work by Hart ([1968](#)) is one of the first to address [IS](#) with the Condensed Nearest Neighbour Rule ([CNNR](#)), an algorithm that starts the selection of the training dataset with one randomly selected instance per class. Next, the classifier is trained and tested iteratively, the test instances that are misclassified are added to the training dataset. Wilson ([1972](#)) focuses on discarding noisy instances from the training dataset. An instance is discarded when its class is different from the class of majority its neighbour's instances' (based on [KNN](#), with $k = 3$). Later, the author also proposed the Repeated Edited Nearest Neighbour ([RENN](#)) which repeated the process of Edited Nearest Neighbour ([ENN](#)) while changes are made in the selected dataset. The work of Tomek ([1976](#)) builds upon the idea of [ENN](#) with different k s, where the misclassified instances are discarded if selected by the different [KNN](#)s. Edited Normalised Radial Basis Function ([ENRBF](#)) is a more sophisticated extension of [ENN](#) where the instances are discarded based on the probability of given an instance x belonging to the k^{th} class from the training dataset D ($p(y|x, D)$) (Jankowski and Grochowski, [2004](#)). In a similar way Vázquez et al. ([2005](#)), apply [ENN](#) but using the probability of belonging to a class instead of using the [KNN](#) rule.

Wilson and Martinez ([2000](#)) explore the idea of associate and proposes five methods, named Decremental Reduction Optimization Procedure ([DROP](#))1 – 5, that make use of this concept. The associates of an instance x are those instances such that x is one of their k nearest neighbours. [DROP](#)1 discards instance x from D if the associates of x in the subset D' are correctly classified without using x to train the model. [DROP](#)1 has the problem of not being effective in discarding noisy instances. [DROP](#)2 was thought to fix this issue by extending [DROP](#)1 and testing not only the associates of x in subset D' but for the whole dataset D . [DROP](#)3 and 4 use a combination of [ENN](#) and [DROP](#)2. [DROP](#)3 uses a noise-filtering pass using [ENN](#) without constraints. In [DROP](#)4, an instance considered noisy is only removed if it is misclassified by its [KNN](#)s and if the removal

of the instance does not degrade the performance. **DROP5** is based on the same idea as **DROP2**, but discards instances with a different class, to smooth the decision boundaries.

Based on the definitions, the **SVM** (Cortes and Vapnik, 1995) is not only a classifier but also an **IS** method since among the elements in D , only the support vectors (V) are used to discriminate between classes, meaning that $D' = V$ (Olvera-López et al., 2010). Based on this idea, Li et al. (2005) present a wrapper method based on **SVM** that works by performing a two-step selection: in the first step it uses a **SVM** to obtain V and afterwards uses **DROP2** over the dataset V . Another method related to instance selection based on SVM is Support Vector k-Nearest Neighbour Clustering (**SV-kNNC**) proposed by Srisawat et al. (2006), which after applying SVM over D uses k-Means clustering V , retains only instances belonging to homogeneous clusters. When the cluster is not homogeneous, the algorithm preserves the instances from the majority class.

The primary use of **IS** is to tackle the problem of unbalanced data. The most straightforward resampling methods are random oversampling and random undersampling (Drummond and Holte, 2003). The former augments the minority class by exactly duplicating the examples of the minority class, while the latter randomly removes instances of the majority class until the classes are balanced. One of the shortcomings is that random oversampling may make the decision regions of the learner smaller and more specific, thus causing the learner to overfit. Random undersampling can remove some useful information from the data sets. The work by Drummond and Holte (2003) compares this type of approaches and concludes that undersampling produces reasonable sensitivity to misclassification costs and class distribution, whereas oversampling is ineffective, producing little or no change regarding performance. Nevertheless, this is still an active research area, and divergent opinions exist regarding which algorithms work better.

The most commonly accepted approaches for unbalanced datasets are Synthetic Minority Over-sampling Technique (**SMOTE**) by Chawla and Bowyer (2002) and ADaptive SYNthetic sampling (**ADASYN**) by He et al. (2008). **SMOTE** is a standard boosting procedure, improving the prediction of the minority class while not sacrificing the performance of the whole testing dataset. The algorithm generates new synthetic instances via feature interpolation between the minority instances and the nearest neighbours. A clear advantage of **SMOTE** is that it makes the decision regions larger and less specific. **ADASYN** also uses feature interpolation to generate synthetic instances. The difference is that instead of applying a uniform distribution for **IS** like **SMOTE**, **ADASYN** uses a density distribution as a criterion to automatically decide the number of synthetic instances to generate for each minority class. Other researchers explore several improvements and extensions of **SMOTE**. Recent advancements include the work by Han et al. (2005) who presented BorderLine-SMOTE, based on the idea that to achieve better prediction performance, most of the classification algorithms learn the boundary of each class as exactly as possible in the training process (Olvera-López et al.,

(2010; Toussaint, 2002). The authors state that the instances on the boundary and the ones nearby are more prone to be misclassified than the ones far from the boundary, and thus are more important for classification. Based on this analysis, the authors propose and test oversampling borderline instances of the minority class. Alejo et al. (2015) introduced a dynamic oversampling method, a hybrid method that combines SMOTE with a sequential backpropagation algorithm. The oversampling rate is based on the error of the backpropagation Mean Square Error (MSE). The algorithm only selects the training instances necessary to deal with the class unbalance problem while avoiding an increase of the training time. Almogahed and Kakadiaris (2015) worked on an algorithm called filteriNg of ovEr-sampled dAta using non-cooperatiVe game theory (NEATER). The classification problem is formulated as a non-cooperative game, where all the instances are players and the goal is to uniformly and consistently label all of the synthetic data created by oversampling techniques. It uses mainly SMOTE and ADASYN algorithms to generate instances. The authors report that the algorithm does not require any prior assumptions and selects representative synthetic instances while generating a minimal number of noisy instances.

There is a connection among IS, AI and boosting approaches. The work by Sung (1996) explores this connection. The main contributions are two-fold. First, the creation of a general distribution based model to accurately map the instances into feature space, where the model learns from the instances a similarity measure for matching new patterns against the distribution based model. Since the authors worked on object and pattern detection problems that are based on learning, the system's performance depends heavily on the quality of the training instances it receives. For that reason, they worked on the selection of high-quality instances for the learning task. The second contribution was the successful application of an AI formulation for function approximation. Based on the work with three specific approximation function classes, Sung (1996) shows that the active IS strategy learns its target with fewer instances than random sampling. Afterwards, the authors simplify the original AI formulation and show how it leads to a tractable IS paradigm suitable for use in object and pattern detection problems. The work by Freund and Schapire (1997) on boosting is described as a wrapper method, which takes a generic learning algorithm and adjusts the distribution given to it by removing some training data based on the algorithm's behaviour. The basic idea is that, as learning progresses, the booster selects instances from the distribution to keep the accuracy of the learner's current hypothesis near to that of random guessing. The consequence is that the learning process focuses on the distribution of instances that are currently hard to learn. Boosting under default conditions reduces the number of instances required for the algorithm to learn. Drucker et al. (1994) have shown that boosting can improve the accuracy of ANN methods on tasks involving Optical Character Recognition (OCR). Boosting approaches seem especially appropriate for algorithms that use optimisation techniques, e. g., back-propagation, where training is more computationally expensive than testing.

2.3.3 *Evolutionary and Bio-Inspired*

The calculation of a consistent training subset with a minimal cardinality for the **KNN** rule is an NP-hard problem (Hart, 1968). Like many other combinatorial problems, **IS** may require an exhaustive search to obtain an optimal solution. Since we can frame the problem of finding the best group of instances as a search and optimisation problem, approaches that use metaheuristics for **IS** are widespread. Tsai et al. (2013) report several optimisation techniques to find the training subset of small size. Such methods include tabu search, gradient descent and simulated annealing.

One of the first approaches was proposed by Cameron-Jones (1995), who uses hill climbing algorithms to find the instance subset with the best accuracy based on difference heuristics. Gathercole and Ross (1994) use **GP** for **IS**. Nevertheless, the authors argued that **GP** on a difficult problem with a large training dataset, a large population size is needed and a significant number of evaluations must be carried out. The authors worked to reduce the number of such evaluations by stratifying the training dataset on which to carry out the **GP** algorithm, attaining promising results.

Cano et al. (2003) use **EAs** for knowledge discovery data reduction. The authors compare conventional **GAs** with different parameter configurations, with non-evolutionary **IS** algorithms. The results show that the evolutionary instance selection algorithms consistently outperform the non-evolutionary ones. They report the following advantages: better instance reduction rates; higher classification accuracy; and models that are easier to interpret.

Derrac et al. (2012) explore an approach integrating **IS**, instance weighting, and feature weighting, in the scope of a Co-Evolutionary Algorithm (**CA**) model framework. To accomplish these tasks, three populations are defined within a cooperative framework. The first performs **IS**, aiming to select a suitable subset of instances to enhance the classification performance of the **KNN** classifier. To increase the speed of the final classification process, it tries to reduce the size of the subset as much as possible. The second and third populations perform feature weighting and instance weighting, respectively. Both are used to select the best possible weights to further increase the performance of the **KNN** classifier. Their search processes are guided by a Steady-State Genetic Algorithm (**SSGA**) with a crossover operator with multiple descendants. This operator is used to increase the convergence capabilities of the standard **SSGA**, which is a necessary improvement in the global behaviour of the **CA**. The authors compare it with a wide range of evolutionary and non-evolutionary related methods, showing the benefits of their co-evolutionary approach.

Miloud-Aouidate and Baba-Ali (2013) perform **IS** based on Ant Colony Optimization (**ACO**) principles, called Ant Instance Selection (**Ant-IS**) algorithm. The objective is to optimise the number of instances to be selected from a dataset to train a **KNN** with the best accuracy. The experimental results on several validation datasets are compared to other **IS** approaches. The results provide evidence that: (i) **Ant-IS** is competitive with

the approaches based on [KNN](#), and (ii) the datasets created with [Ant-IS](#) offer better classification accuracy than those obtained with other algorithms considered in the study.

Triguero et al. [\(2015\)](#) explore the use of classification techniques in a Big Data [\(BD\)](#) scenario. The authors argue that the huge quantities of instances available for evaluation and learning may limit the applicability of most of the standard techniques. This problem becomes even more difficult when the class distribution is skewed, an issue known as unbalanced big data classification. Evolutionary undersampling techniques have shown to be a promising solution to deal with the class unbalance problem. However, according to Triguero et al. [\(2015\)](#), typically, their practical application is limited to problems with no more than tens of thousands of instances. To overcome this limitation, they propose a parallel model to enable evolutionary undersampling methods to deal with large-scale problems. For this purpose, they resort to Map Reduce schemes that distribute the algorithm's execution in a cluster environment. Furthermore, they developed a windowing approach for class unbalance data to speed up the undersampling process without losing accuracy. They tested their algorithms with several datasets with up to 4 million instances. The overall results reported promising scalability abilities.

2.4 INSTANCE GENERATION

Creating or obtaining representative examples to include in a training dataset is a hard task. **IG** approaches, or more recently referred to as Data Augmentation (**DA**) approaches, address these issues. They are often part of bootstrap sampling techniques, a process for creating a distribution of datasets out of a single dataset with the objective of iteratively improving a classifier's performance (Davison and Hinkley, 1997; Sammut and Webb, 2010). The term **DA** refers to methods for constructing iterative optimisation processes or sampling algorithms, with the generation and introduction of unobserved data or latent variables (Dyk and Meng, 2001). Historically, for deterministic algorithms, **DA** was popularised in the general statistical community by the seminal article by Dempster et al. (1977), regarding the usage of Earth Mover's (**EM**) algorithm for maximising a likelihood function or, more generally, a posterior density. For stochastic algorithms, **DA** was popularised in the statistical literature by Tanner and Wong's (2009) **DA** algorithm for posterior sampling.

Transposing the idea of **DA** to **ML**, the general approach is to generate instances that can be used for training the model and changing its behaviour, changing the shape of the decision boundary, or simply improving the structure that models the distribution of data. There are two main types of approaches for **IG**: (i) adding noise or variations to the existing examples; and (ii) creating a model capable of generating new instances. We survey approaches that focus on the domain of **IG** in general and, in particular, image generation approaches since it is a dominant topic in this thesis. Similarly to the previous Section, the approaches that use evolutionary techniques are analysed in a separate Subsection.

2.4.1 Modification of Instances

The core idea of this kind of **IG** approach is to modify existing instances from the dataset, creating new ones using a model or function to do so. The overall process is illustrated in Figure 2.20, where an instance is modified on one of its features, leading to a new instance. Obviously, one may modify more than one feature.

Sung and Poggio (1995) present a commonly used method for bootstrapping in image classification problems that implies the synthesis of instances. In a face detection problem, the authors present a bootstrapping method for augmenting the non-face instances composed of the following steps: (i) selecting a starting set of instances of

$$\mathbf{x}_m = \{x_{m0}, \mathbf{x}_{m1}, \dots, x_{mn}\} \longrightarrow f(\mathbf{x}) \longrightarrow \mathbf{x}'_m = \{x_{m0}, \mathbf{x}'_{m1}, \dots, x_{mn}\}$$

Figure 2.20: **IG** using a function $f(\mathbf{x})$ to modify, e. g., the feature x_{m1} to create a new instance.

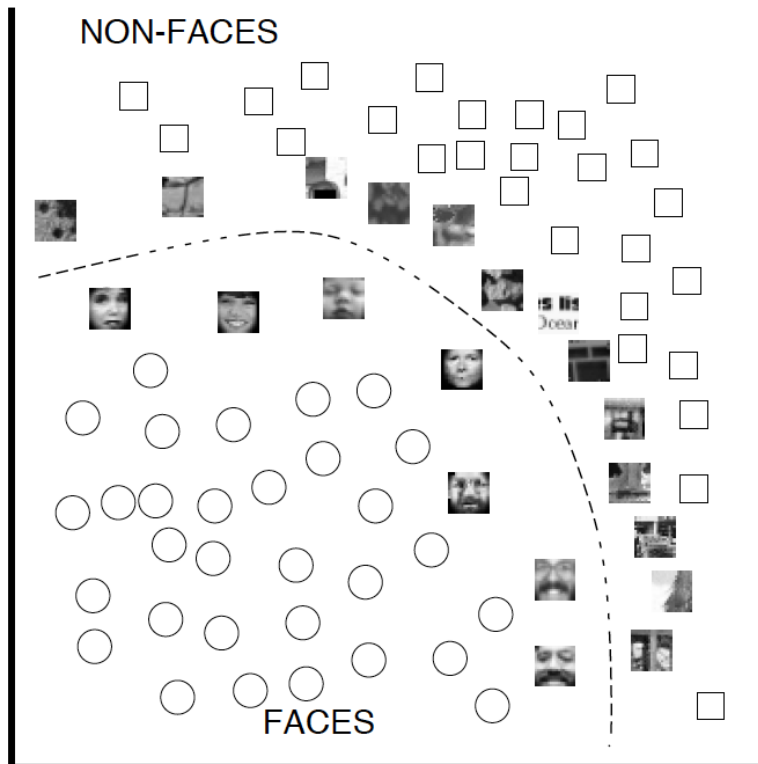


Figure 2.21: The [SVM](#) boundary for faces (circles) and non-faces instances (squares) (image from Osuna et al., [1997](#)).

non-face images; (ii) training a classifier (a [MLP](#)) with the current dataset of images (face and non-face); (iii) randomly selecting parts of non-face images, classifying them, and adding the misclassified image parts to the current dataset of images; (iv) repeating this process from step (ii) until a certain criterion is met (e.g. performance degrades or a predetermined number of iterations or samples added is reached). These new images are considered as variations of the existing ones. To augment the positive set of images, the authors introduce variations to the existing instances of faces by rotating, changing contrast and lightning, and mirroring the images. Due to their value for improving the performance of the classifiers, these techniques became popular in face detection approaches (Rowley et al., [1998a](#); Viola and Jones, [2001](#); Yang et al., [2000](#), [2002](#)). Osuna et al. ([1997](#)) use the same approach as Sung and Poggio ([1995](#)) to train [SVM](#) to distinguish faces from non-faces. The authors analyse the [SVM](#)'s decision boundary, concluding that some of the images of the non-face dataset near the decision boundary resemble faces and that these images are arguably similar to the face instances near the boundary (as shown in Figure [2.21](#)).

Niyogi et al. ([1998](#)) use prior knowledge to create new instances, thereby expanding a training dataset. At first, they perform an overview of the supervised learning formu-

lation, analysing the complexity of the learning problem. They discuss the complexity of these issues and why in the absence of prior knowledge a vast number of instances may be required to achieve adequate performances. The authors describe specific non-trivial transformations based on prior knowledge, which allow them to generate new instances for pattern recognition problems. In the context of face recognition, they create different virtual views from a single instance by using prior knowledge about linear object classes, 3D views, 2D projections and rotations (Beymer and Poggio, 1995). A model for face recognition is trained using 1 real view and 14 virtual samples per person, achieving 85% correct classification rate. Nevertheless, it does not achieve the performance of a model trained with 15 real views, which attained 99% correct classification rate. Both systems perform better than the initial system, with only 1 real view, which attained 32%, showing the potential of synthesised instances to improve performance.

In a speech recognition problem, Stahl et al. (2001) propose a method for synthetically generating more training instances, covering the maximum types of instances and thus improving performance on test data. The authors state that the mismatch between the training and test set recording conditions is an issue and that one solution is to enlarge the speech collections by conducting records in many different environments and conditions. Since it is hard to do cover all the possible conditions, they propose an approach to generate training data synthetically by filtering clean speech with impulse responses and adding noise signals from the target domain. The authors experiment several types of variations to create new instances for training their models. The results attained in a test dataset suggest that the new instances significantly improve the performance.

Simard et al. (2003) propose a simple technique for vastly expanding the training set based on elastic distortions (see Figure 2.22), creating different training instances for the Modified National Institute of Standards and Technology (MNIST) dataset. They argue that simple distortions such as translations, rotations, and skewing can be generated by applying affine displacement fields to images. Their model trained with instances generated with affine distortions significantly improved their results on the MNIST test dataset.

2.4.2 *Generative Machine Learning*

As analysed in Section 2.1, in generative approaches, a ML model is used to model a dataset distribution. These models can be used to generate new instances that can be used to improve ML approaches. Other types of approaches use additional models to generate new instances and add them to the training dataset to improve the performance of another model.

Melville and Mooney (2004) present a new meta-learner called Diverse Ensemble Creation by Oppositional Relabelling of Artificial Training Examples (DECORATE),

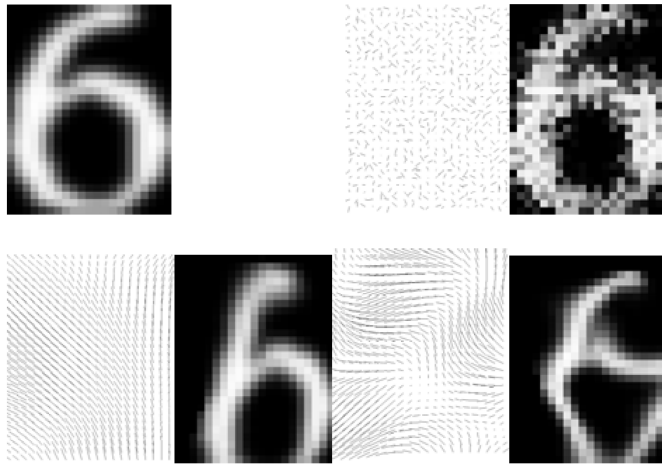


Figure 2.22: On the top left, the original image. On the right and bottom, the pairs of displacement fields applied to the original image and the resulting images (image from Simard et al., 2003).

which uses strong learners to build an effective and diverse committee. The approach relies on adding different randomly constructed instances to the training set when building new committee members. Melville and Mooney (2004) assume that the features are independent, and generate new instances by randomly picking instances from an approximation of the training dataset distribution. In other words, for a numeric attribute, a new feature is defined with values from the Gaussian distribution defined by mean and standard deviation of the feature values in the training dataset. In case of a nominal feature, the authors compute the probability of occurrence of each distinct value in its domain and generate values based on this distribution. Classes are assigned based on the predictions of the current ensemble. Thus, a prediction from the ensemble is made. If the probability of belonging to a class is 0, a small constant value is assigned. The classes are then selected, so that the probability of selection is inversely proportional to the current ensemble's predictions. The authors state that this process directly increases the diversity of the committee when a new classifier is trained on the augmented data. Furthermore, they argue that ensembles of classifiers are often more accurate than their component classifiers if the errors made by the ensemble members are uncorrelated, suggesting that their opposite labelling method enforces that. The approach was compared to state of the art approaches, such as boosting, bagging and RFs, obtaining results that are statistically significantly better in 15 representative datasets from the UCI repository¹ (Frank and Asuncion, 2010).

Wang et al. (2005b) propose a manifold-based method to select a face detection training dataset. They apply a subsampling algorithm based on manifold called Isomap (Isometric feature mapping (Tenenbaum et al., 2000)). Isomap maps the face dataset

¹ UCI Machine Learning Repository — <http://archive.ics.uci.edu/ml/>

into a 2-dimensional space that the authors use to calculate distances among the instances. The subsampling of the dataset is performed by excluding instances that are closer to each other more than a given threshold. This is done by calculating the gaps between the instances in the manifold space and creating synthetic instances to fill them. The synthetic instances are variations of the ones surrounding the gaps. These instances are generated through a method that the authors named interweaving, using the following algorithm: (i) apply Principal Component Analysis (PCA) and compute the coefficients e for all faces in the current dataset; (ii) calculate the nearest instances to the gap, based on a predefined maximum distance; (iii) calculate the weights for the nearest instances based on the distance to the gap; (iv) construct the new instance using a weighted linear combination of the coefficients e of the nearest instances (see, e. g., Figure 2.23). This approach allowed Wang et al. to develop classifiers that perform better in the benchmark dataset (MIT-CMU (Rowley et al., 1998a)) than random sampling methods similar to the ones adopted by Sung and Poggio (1995).

Chen et al. (2007) also used a manifold with the objective of creating a dataset to train a robust face detector. Building on Wang et al.'s work (2005), the authors use Isomap to estimate the distance between instances. Next, the authors resort to Local Linear Embedding (LLE) – a non-linear dimensionality reduction algorithm – to compute weights of the instances in the low dimensional space. Using the distance and weights information, a threshold is defined, and neighbouring instances below the threshold are removed from the dataset. Possible gaps in the low dimensional space are filled through the generation of new instances by using the same method by Wang et al. (2005b). The combination of Isomap and LLE is applied to the positive and negative datasets, and the resulting datasets are used to train an AdaBoost classifier. After using the AdaBoost classifier on a large dataset, they collect the false positives and use them to train a one-class SVM. The final detector is the combination of the Adaboost classifier and the SVM. Their results in the MIT-CMU dataset outperformed state of the art approaches at that time (93.5% correct detection rate). The work by Wang and Chen indicates that, in this specific context, generating a large set of positives and negatives instances based on a starting dataset can significantly improve the performance of classifiers.

Sapp et al. (2008) worked with synthetic instances to gather a large dataset for real image object recognition. The idea consists in capturing images of a few example objects in a green screen, and then synthesise a new, larger dataset by perturbing the foreground, background and shadow of the images using a probabilistic model. To learn the probabilistic model they used the groundtruth information about the foreground, the background, the object and the texture. The key insight of this approach is that it can model the true distribution of an object class roughly as well as real data, by using synthetic data derived from real images.

Similarly, Jaderberg et al. (2014) create a real text image generator. Jaderberg et al. (2014) argue that the available datasets are very limited regarding the existing text








































Synthetic virtual sample:  (in the white complexion)						
Original samples						
weight	0.1172	0.1109	0.0883	0.0816	0.0803	0.0800
Original samples						
weight	0.0780	0.0776	0.0758	0.0734	0.0690	0.0675
Synthetic virtual sample:  (in the black complexion)						
Original samples						
weight	0.1179	0.1132	0.0951	0.0850	0.0838	0.0770
Original samples						
weight	0.0760	0.0716	0.0712	0.0703	0.0692	0.0690
Synthetic virtual sample:  (a female)						
Original samples						
weight	0.0887	0.0874	0.0874	0.0847	0.0843	0.0828
Original samples						
weight	0.0828	0.0825	0.0811	0.0803	0.0796	0.0779

Figure 2.23: Instances generated by the interweaving algorithm. (image from Wang et al., 2005b)

vocabulary, i. e., the text that appears in natural images. Therefore, to overcome this issue they developed an offline process to synthesise instances. It randomly renders typefaces (fonts) under different randomly parameterised conditions. These include border and shadow rendering, base colouring, projective distortion, and adding noise. At the final step, the font is blended into an existing image of the starting dataset. The whole process is shown in Figure 2.24. The results obtained are highly realistic and can replace real-world instances of training datasets in a problem of text recognition in natural images, i. e., scene text recognition. The authors conducted experiments where they replaced the real instances with the synthetic data to train different CNNs and obtained state of the art performance in several datasets such as ICDAR 2003, ICDAR 2013 benchmark, Street View Text, and IIT5k.

The following works described in this Section are relevant as IG approaches but they were not used to improve the performance of classifiers.

Both VAEs and GANs (defined in Subsection 2.1.1) are able to generate instances (see Figures 2.25 and 2.26). Larsen et al. (2015) perform a study comparing these approaches to synthesise new instances evaluating only the generation process. It also proposes two more approaches, a combination of both, VAE-GAN and a VAE that uses a GAN to measure similarity of the outputs and improve the VAE training process (VAE_{Dis1}). In this work, the idea is to learn a generative model for face images conditioned on facial attributes. At test time, the authors use the different models to generate face images by retrieval from chosen attribute configurations. A separately trained regressor model predicts the attributes from the generated images. They argue that a good generative model should be able to produce visual attributes that are correctly

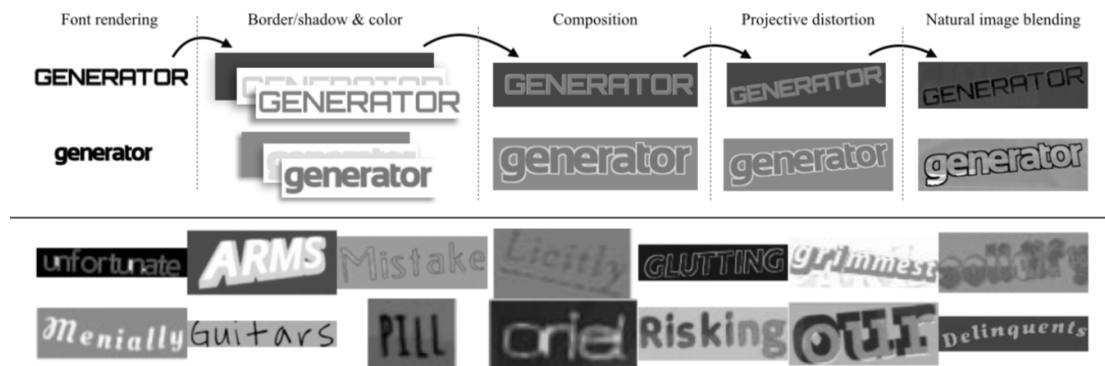


Figure 2.24: On the top row, the text generation process. It involves text generation, font rendering, creating and colouring the image layers, applying projective distortions, and finally image blending. On the bottom, some randomly sampled data created by the synthetic text engine (image from Jaderberg et al., 2014).



Figure 2.25: Images generated with a VAE on CIFAR-10 (image from Goodfellow, 2016).

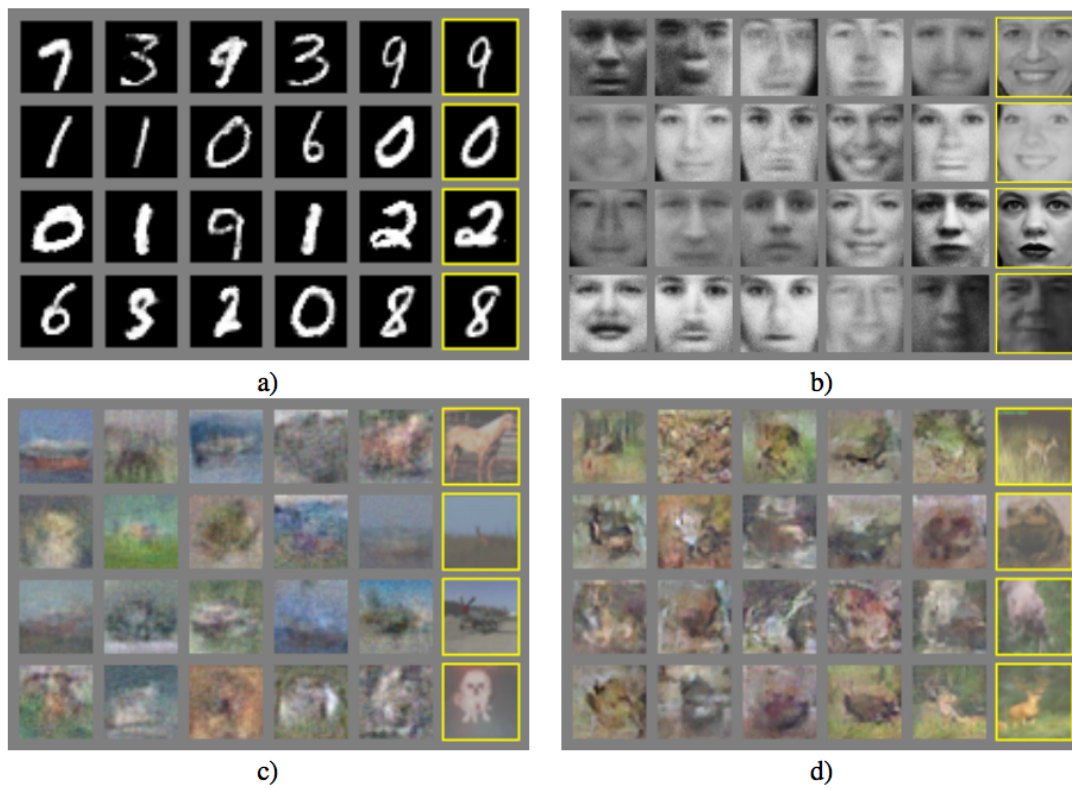


Figure 2.26: Images generated with a GAN. The images from the training dataset are highlighted in yellow (image from Goodfellow et al., 2014).

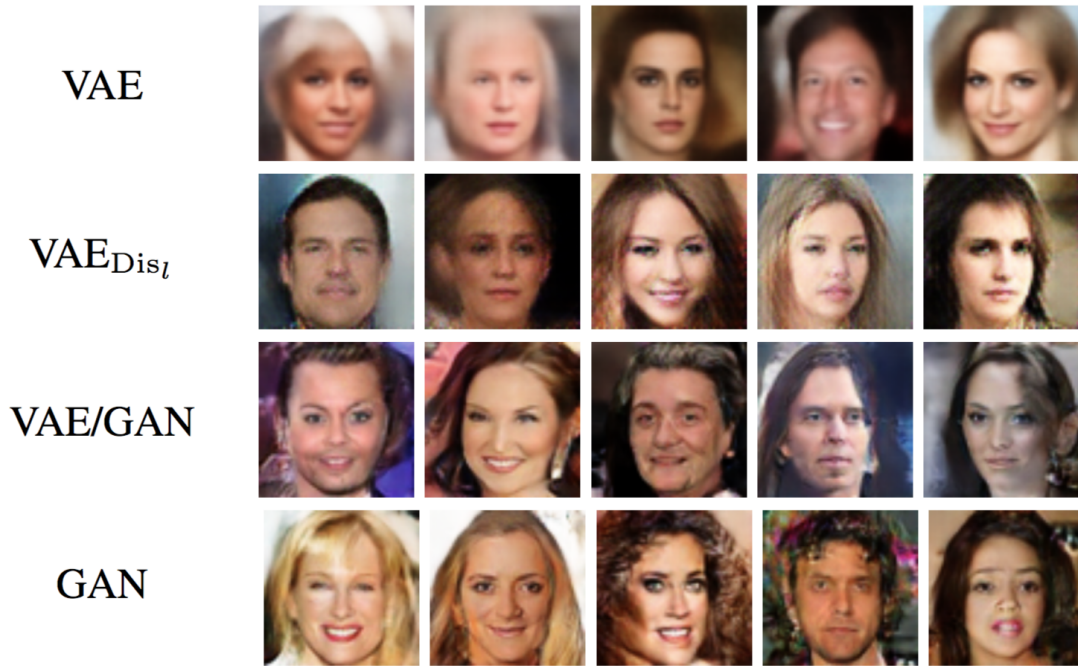


Figure 2.27: Comparison of instances generated using [VAE](#), [VAE_{Dis_l}](#), [VAE/GAN](#) and [GAN](#) (image from Larsen et al., [2015](#)).

recognised by the regression model. Therefore, they proposed a method to evaluate the reconstruction of the instances, arguing that the [VAE/GAN](#) reconstructs the instances with less error (see Figure [2.27](#)).

[GANs](#) are hard to scale when using [CNNs](#). Radford et al. ([2015](#)) introduced the Deep Convolutional Generative Adversarial Network ([DCGAN](#)) as a stable set of architectures for training [GANs](#) that give evidence that they learn good representations of images for supervised learning and generative modelling. The authors altered the architecture of the networks that allowed them to train robust convolutional discriminator and generator networks. Their work revealed that the latent space of these networks could be traversed via the representation space and that vector operations showed consistent and stable generations with semantic relevance. Thus, the generator holds arithmetic properties based on the vectors of the latent space; e. g. averaging the z vectors of the latent space that generates a smiling woman minus the average vector that creates a neutral woman plus the average vector that generates a neutral man is going to generate a vector z that creates smiling men as shown in Figure [2.28](#).

Gregor et al. ([2015](#)) argue that most approaches for image generation aim to generate entire, complex images in one step. The authors suggest that when a person is drawing, a sequential and iterative process takes place, refining and evaluating the drawing at each step until the drawing is finished. The authors aim to replicate such process

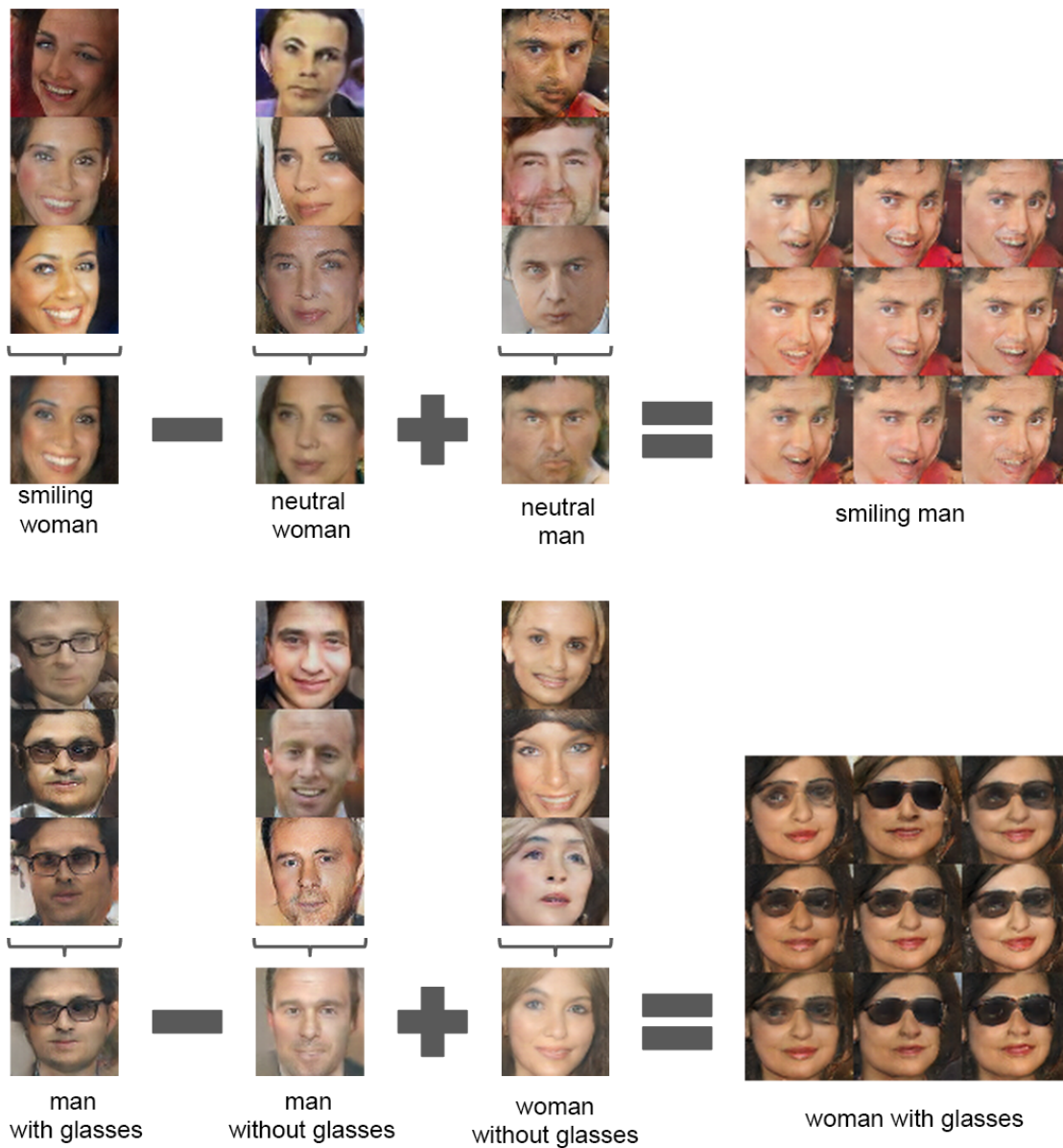


Figure 2.28: Examples of DCGAN arithmetic using latent space vectors. Averaging z vectors of a type of instances and performing arithmetic operations with them yield a new average vector, and by sampling from it we get a different type of instances (image from Radford et al., 2015).

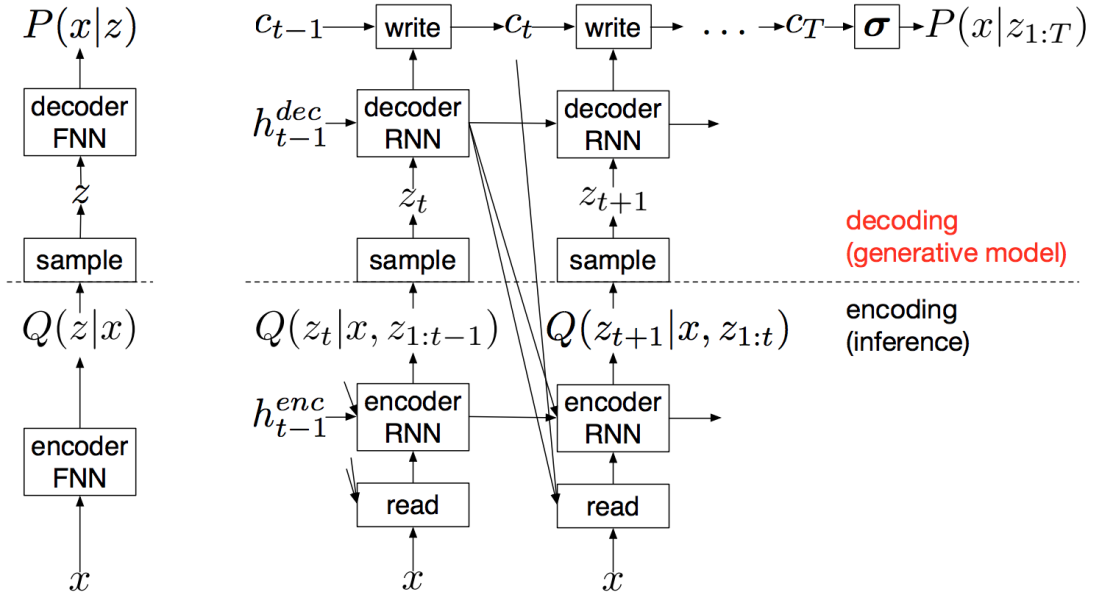


Figure 2.29: On the left a [VAE](#) and on the right a [DRAW](#) network (image from Gregor et al., 2015).

by introducing the Deep Recurrent Attentive Writer ([DRAW](#)) network. Using [DRAW](#), each part of the scene is generated independently. The network consists of pairs of [RNNs](#) (definition in Subsection 2.1.1) encoders and decoders. The encoder network compresses the real images presented during training and a decoder that reconstitutes images to the canvas space. It is considered a type of [VAE](#), extending them with a progressive refinement mechanism and spatial attention algorithm (see Figure 2.29). Based on the [RNN](#) idea of connecting the hidden layers of the t with $t-1$, using [DRAW](#), the hidden layer of the $t+1$ encoder is connected to the $t-1$ layer of the decoder. Furthermore, for each t , the decoded images are successively added to form the final image. Another important characteristic is the existence of canvas matrix, i. e., where the pixels of the final scene are drawn. The canvas matrix (c in Figure 2.29) takes part of the generation process. The network at each time t must decide what part of the image is going to serve as input, where it is going to generate the output and what it is going to generate (depicted as read and write boxes in Figure 2.29). The system is trained end-to-end with stochastic gradient descent, where the loss function is a variational upper bound on the log-likelihood of the data. These two points greatly reduce the complexity of the information that the Autoencoder needs to learn, thereby allowing its generative capabilities to handle larger, more complex distributions, like natural images. To generate an image, we sample from z_t space running the decoder at each time step t , adding each image result until all steps t of the network are processed. This approach improves the generative modelling capabilities by splitting

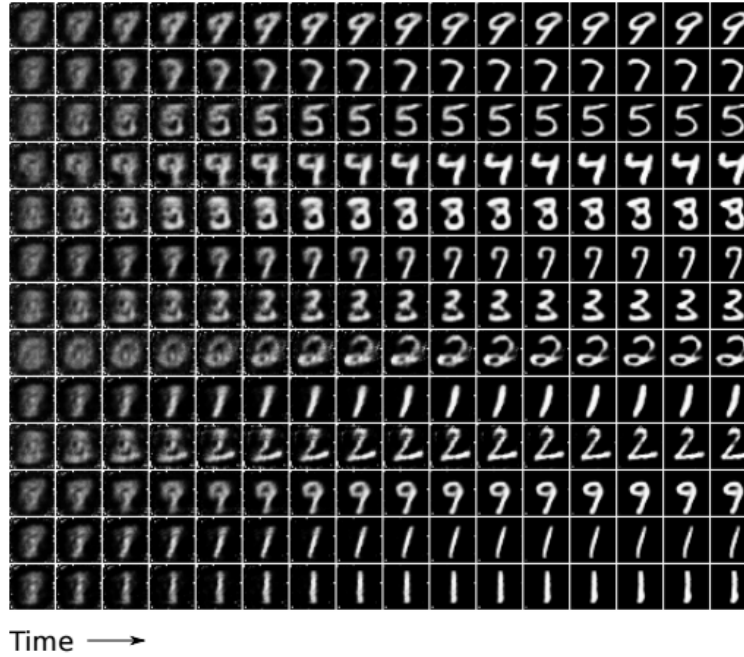


Figure 2.30: Example of the **DRAW** network generation process. It shows the generated images throughout the different time steps (image from Gregor et al., 2015).

the complexity of the task across the temporal and spatial domains, refining the output in each time step as shown in Figure 2.30. The approach was used to generate images for the **MNIST**, **MNIST** with two digits, Street View House Numbers dataset and CIFAR (see Figure 2.31).

Oord et al. (2016) introduced the Pixel **RNN**. It uses autoregressive **LSTMs** to model the explicit density of the dataset. It is based on the chain rule that allows to calculate any member of the joint distribution of a set of random variables using only conditional probabilities. Thus, to define the likelihood of an image x , it computes the probability of the i^{th} pixel given all previous pixels. The approach must generate one pixel first, then, based on the first pixel generated, it generates the second until it produces the whole image. Examples of the generated images are shown in Figure 2.32. The authors also present Pixel **CNN** an alteration to the previous model where **CNNs** are used to model a portion of the image and not only one pixel. Consequently, each previously evaluated pixels are dependent on the modelling of a **CNN** over some region. The results attained are comparable with the **GAN**'s results with a simple way to calculate the likelihood, but the approach is computationally more expensive.



Figure 2.31: Instances generated with **DRAW** networks for different image datasets: on the top left **MNIST**; on the top right the Street View House Numbers; on the bottom left CIFAR-10; and on the bottom right the **MNIST** with two digits. All images, except the **MNIST** with two digits, have in the rightmost column the closest (in terms of **RMSE**) groundtruth instance from the training dataset.

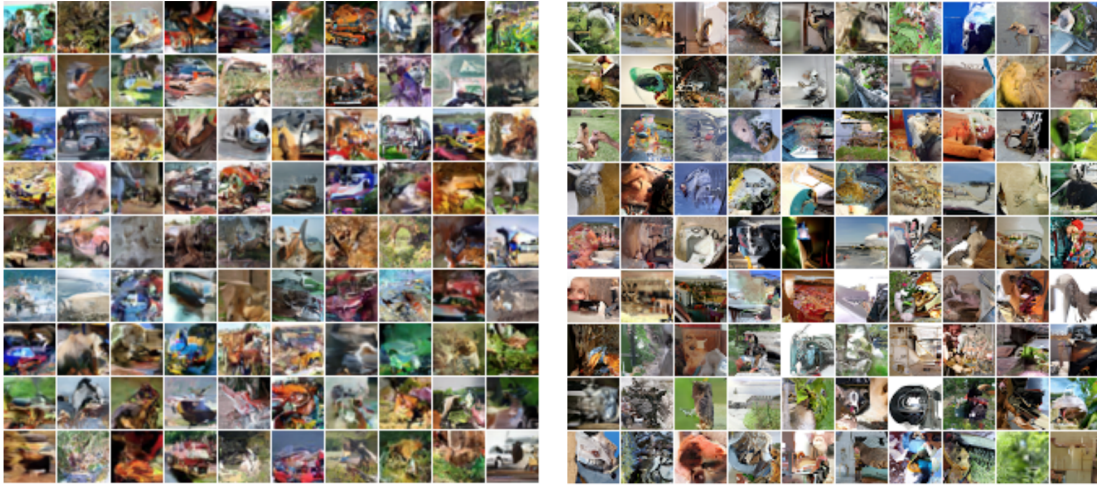


Figure 2.32: Synthetic images generated using Pixel RNN. On the left, images generated with models trained with the CIFAR-10 dataset and, on the right, images generated with models trained with the ImageNet dataset (image from Oord et al., 2016).

2.4.3 Evolutionary

EC has been used in the improvement of ML models; e. g., EAs have been used in ML to evolve hyper-parameters, features, the model's weights, among others. In IG, EAs are involved in the generative process. That is, EAs are used to create, from scratch or based on the parametrisation of a pre-defined model, new instances to improve the ML approach. To the best of our knowledge, while surveying the work done in this particular area, we concluded that few examples use EC for IG.

In the seminal work by Baluja et al. (1994), a system for automatic image generation is presented. The system employs a GP engine with an ANN to automate the production of images based on the user's preference. The training dataset is composed of images generated through interactive evolution. The user assigns a score to each of images of the training dataset. Then, an ANN was trained using backpropagation on the dataset. The GP engine uses the ANN to assign fitness. Several experiments were conducted with different configurations of ANN and datasets. The authors state that some of the evolutionary runs quickly converged to plain images with a high activation but did not match the user's preference. The propensity for EA to find shortcuts on the fitness landscape and exploit shortcomings of the classifier, at that time, was also exposed in other works, e. g., by Spector and Alpern (1994) and Teller and Veloso (1995).

Machado et al. (2007a,b) present an approach for the production of images based on the promotion of an arms race between an ANN and an EC system. The system is comprised of three components: a Feature Extractor (FE), a GP engine, and an ANN. The FE extracts features from the input images, mainly based on complexity metrics. The

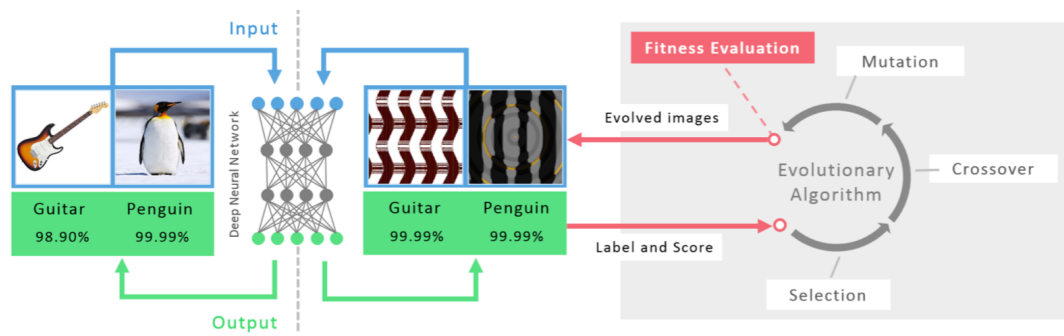


Figure 2.33: Overview of the process of image generation using an EA and a DNN (image from Nguyen et al., 2015).

GP engine, called NEvAr (Machado and Cardoso, 2002), allows the generation of populations of images (see Section 4.1.2.1). The ANN is a MLP that receives the FF features as input and is trained to discriminate among images created by NEvAr (internal) and famous paintings (external). The ANN is used to assign fitness to the images generated along the populations. The system evolves populations of images and tries to find images classified as external by the ANN. After NEvAr finishes the evolutionary run, the images created and classified as external are added to the internal set of images. An ANN is retrained to distinguish between the two sets. The process is repeated for several iterations and based on the experimental results of training, test and validation datasets, the authors concluded that the classifier improved its performance along the iterative process. Furthermore, the evolutionary process also explores different paths, generating different types of images. Among the generated images, there were atypical images that explored shortcomings of the classifier.

Also related with the work by Machado et al. (2007a), more recently, Nguyen et al. (2015) worked on EAs to generate images using DNNs to assign fitness. Figure 2.33 shows an overview of the process and results attained. They tested EAs with two different representations: a direct encoding, where the individual is a pixel based representation with operators that change the values by polynomial mutation operators; and a Compositional Pattern Producing Network (CPPN) that can evolve complex images, based on Picbreeder (Secretan et al., 2008). Although some of the evolved instances are recognisable by humans as being the target object, the authors also conclude that state of the art DNNs for object recognition show high confidence when evaluating *noisy* and unrecognisable images/instances that come from the CPPN. They reported that these images did not significantly improve the performance of classifiers on the Imagenet dataset. This work is related but preceded by this thesis's work and publications, starting with the work in face detection (Machado et al., 2012a,b), object detection (Correia et al., 2013a), production of aesthetically pleasing images (Correia et al., 2013b, 2017) that are going to be discussed in Chapters 5, 6 and 7.

Chen et al. (2004) propose a self-adaptive GA to improve face detection systems. The approach relies on sampling the face training dataset. The initial population of the GA is composed of positive instances of a training dataset, i. e. faces. Therefore, each individual is an image, encoded as an integer vector representing the intensity values of each pixel. The recombination operator is based on an image segmentation process that divides the image into regions (forehead, eye, nose, mouth) and exchanges the segments between individuals. Mutation consists in changing certain aspects of the image, such as illumination, position and angle of the selected segmented parts. The fitness of the individuals is determined by the output of a SNoW classifier (Yang et al., 2000). At each generation, the individuals that were classified as faces are added to the training dataset, and the SNoW classifier is retrained with the augmented set. Adding more variations to the positive dataset led to performance improvement when compared with the initial classifier. Furthermore, this approach is combined with the approach by Chen et al. (2007) (already described in the previous Section) into a single framework for IC by Chen et al. (2009). It uses the described self-adaptive GA to generate positive instances. The negative instances are generated using the method introduced by Sung and Poggio (1995) (see Subsection 2.4.1). The resampling is performed by using Isomap and LLE from the work by Chen et al. (2007). A SNoW classifier is trained with the resulting dataset, which obtains a 90.7% correct detection rate with no false alarms on the MIT-CMU dataset. This work is relevant for this thesis for a couple of reasons. First, it shares the same classification problem that will be described in Chapter 5. Second, it uses a classifier to assign fitness in the evolutionary process similar to the approach that we propose in Chapter 3. Last, it is an approach that evolves photorealistic faces, which is related to the work of Section 4.5.

EVOLUTIONARY FRAMEWORK FOR CLASSIFIER ASSESSMENT AND IMPROVEMENT

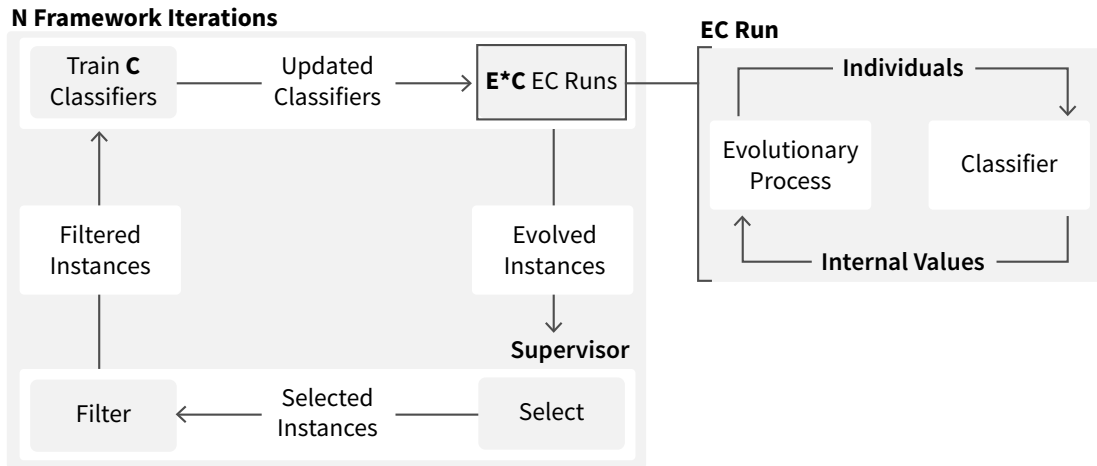
Based on the survey presented in the previous Chapter we propose a framework that uses **EC** to generate instances to improve the dataset and consequently the performance of classifiers. We started with the idea for a generative framework that evolved into an **IS** and **DA** framework. We saw an opportunity to explore the potential of **EC** to create new instances of a particular class, i. e., a generative approach, that combines **EC** with a classifier (Machado et al., 2012b). After the first experiments, we observed that our approach was generating instances from the target class, which explored shortcomings of the classifier. We thought that these instances could be included in the training dataset and that training with these new instances should minimise the shortcomings of the classifier and this way improve its performance. We were also motivated by the fact that, at that time, most **IS** approaches relied on the starting base dataset to generate new instances. With our approach, we rely on the quality of the classifier.

Note that, at the start of our work (Machado et al., 2012a,b), some of the works mentioned in the state of the art were not published: Goodfellow et al. (2014), Nguyen et al. (2015), and Radford et al. (2015). We were motivated by the ideas by Romero et al. (2003), Machado et al., 2007a and preliminary results (described in Section 4.2), which led us to create and explore the framework proposed in this Chapter. The framework built during this thesis makes use of new and existing ideas and algorithms, expanding upon the current state of the art by: (i) using **EC** to generate new dataset instances, i. e., perform **IG**; (ii) using classifiers to assign fitness to the instances generated in the evolutionary process; (iii) resorting to sampling algorithms to select suitable instances; (iv) creating an autonomous and dynamic method for **IG** which, based on the classifier's shortcomings, iteratively generates instances that are used to retrain the classifier, minimise the shortcomings, and increase its performance.

In the remainder of this Chapter, we present the framework. This Chapter is divided into five Sections: first an overview of the generalised version of the framework (Section 3.1), afterwards, each relevant component is analysed individually (Sections 3.2, 3.3, 3.4 and 3.5).

3.1 OVERVIEW

EFFECTIVE is composed of three main modules: **EC** engine, **CS** and Supervisor. The framework relies on promoting a competition between the **EC** engine and the **CS**. The **EC** engine is responsible for evolving new training instances. The **CS** classifies the evolved

Figure 3.1: Overview of **EFFECTIVE**.

instances; the results of this classification are used as fitness to guide the evolutionary process. The Supervisor manages the instances that were generated by the **EC** module, deciding which should be added to the dataset. These modules come together to create an iterative process for improvement of classifiers. That is, when the evolutionary runs are over, the classifiers are retrained using an expanded version of the dataset, which includes the evolved instances, and the process is repeated. Figure 3.1 shows an overview of the framework. The approach involves the following steps:

1. Selection of a starting dataset;
2. N framework iterations start; C classifiers are trained based on the available instances, forming the **CS** module;
3. E independent **EC** runs are executed to generate instances; the output of each c in C is used to assign fitness to the evolved instances of the **EC** run (e) assigned to it; the fitness of the instances depends on the results of the classification task;
4. The **EC** runs stop when a termination criterion is met (e. g., a predefined number of generations, or attaining a certain fitness value);
5. The Supervisor selects and filters instances gathered from all **EC** runs, updating the training dataset;
6. The process is repeated from step 2 until the termination criterion is met (e. g., a defined number of framework iterations, attaining a certain number of instances, or reaching a certain performance value).

While these steps provide a high level overview of the process, Algorithm 3.1 provides a more detailed insight of the framework's behaviour.

Algorithm 3.1 The **EFFECTIVE** algorithm.

```

1: procedure EFFECTIVE
2:    $D \leftarrow [ \langle \mathbf{x}_0, \mathbf{y}_0 \rangle, \dots, \langle \mathbf{x}_m, \mathbf{y}_m \rangle ]$  ▷ define a baseline dataset
3:    $C \leftarrow [c_0, \dots, c_i]$  ▷ set of CSs
4:    $E \leftarrow [ec_0, \dots, ec_i]$  ▷ set of ECs
5:    $n_{ec}$  ▷ define the number of EC runs with different random seeds
6:   while criterion not met do
7:      $D' \leftarrow []$  ▷ create an empty temporary dataset
8:     for all  $c_i \in C$  do
9:        $c_i \leftarrow \text{train}(c_i(D))$  ▷ train the classifier with the current dataset
10:      for all  $ec_i \in E$  do
11:        for  $s < n_{ec}$  do ▷ each run has its  $s$  seed
12:           $I \leftarrow \text{evolve}(t, ec_i, s, c_i)$  ▷ using  $ec_i$  with  $s$  seed and classifier  $c_i$ ,
          evolve instances classified by  $c_i$  as belonging to a particular class  $t$ 
13:           $D' \leftarrow [D', I]$  ▷ add them to a temporary dataset
14:        end for
15:      end for
16:    end for
17:     $D \leftarrow \text{supervise}(D', C, E)$  ▷ Supervisor updates the dataset
18:  end while
19:  return  $D$  ▷ return the updated dataset
20: end procedure

```

In each framework iteration, the evolutionary engine must evolve instances of a pre-determined class that are misclassified by the **CS**, otherwise no progress is achieved. After gathering the evolved instances, the Supervisor decides which instances should be used to update the dataset at each framework iteration. Finally, **EFFECTIVE** requires the definition of an appropriate termination criterion (step **6**). This depends, mainly, on the task at hand and on the existing computational resources. In the next Sections, we analyse the details of each module that compose the framework.

3.2 CLASSIFIER SYSTEM MODULE

In theory, **EFFECTIVE** can use any classification algorithm, as long as it is possible to extract information from the classification task that allows the construction of a fitness function able to guide the evolutionary process.

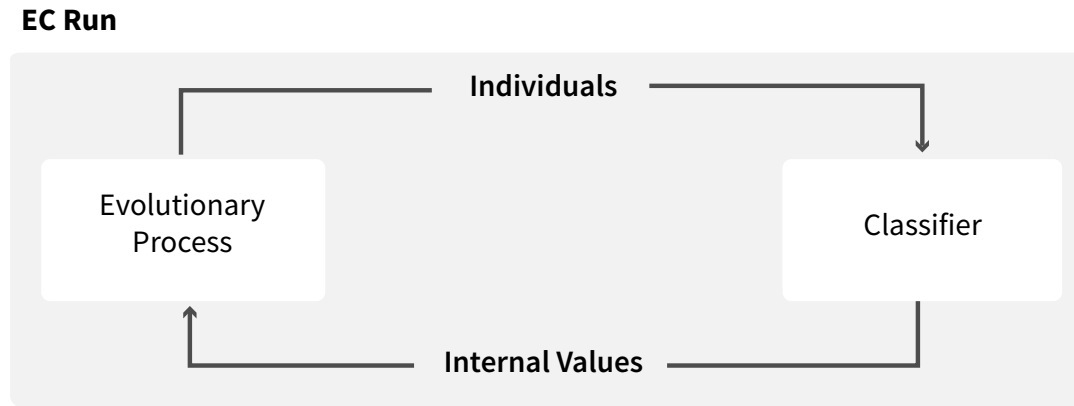
This module is obviously a key part of the framework since we are trying to improve the dataset based on the performance of the **CS**. When looking at Algorithm **3.1**, we can see that it is used in three key points: (i) in the training phase, where the classifiers can be trained under different conditions (step **9**); (ii) in the evolution phase to assign fitness (step **12**); and (iii) in the supervise phase (step **17**), the information on the classification of each classifier (c_i) can be used for the Supervisor algorithm as described in Section **3.4**.

3.3 EVOLUTIONARY COMPUTATION MODULE

The **EC** engine is a key module of this framework, assuming the task of generator of new instances. In **ML**, the generative approach learns how to model the model's distribution of instances of each individual classes. It learns how the instances can be generated by learning to model the joint probability $p(x, y)$, where x is an input and y is the class, generating instances from the model's distribution of instances ($p(x)$) (Y. Ng and I. Jordan, **2002**). To generate suitable instances with an **EC** engine we need to search for instances in the model's distribution of instances ($p(x)$). **EA** algorithms are suitable for search problems as they are well-known for the ability to find and optimise solutions by navigating judiciously in the solution space (Eiben and Smith, **2003**).

The **EC** engine should generate instances of a given class. One way of doing so is to use the **CS** module to assign fitness to the individuals (candidate instances) evolved by the **EC** module. The overview of this particular interaction is shown in Figure **3.2**.

Based on the evaluation that comes from the output and internal values of the classifier, the **EC** algorithm ranks the individuals in the evolutionary process. Individuals with higher fitness values have more chance to survive and reproduce. With a proper **EC** engine we are able to generate instances that come from the distribution ($p(x)$) of the **CS**. However, during the generation process we can have two scenarios: either the instance generated is a true positive or a false positive. When designing the **EC** engine,

Figure 3.2: Overview of an **EC** run.

one should choose a representation that can easily generate instances of the target class but hard to generate for the other class.

Similarly to the concept of adversarial learning from **GANs** (Goodfellow et al., 2014), this module must synthesise instances that the **CS** does not classify correctly or no progress is achieved. There are differences between using the **EC** model as the generator and using a network (a **ML** model), as it is the case in **GANs** (as described in Goodfellow et al. (2014)). With an **EC** algorithm the search is controllable and unrestricted through hyper-parameters, representation and fitness assignment. In contrast, with the search on the latent space (z space) of the **ML** generative approach which is more restrictive and typically controlled with a vector of numerical values in the latent space or a random noise input (Goodfellow, 2016; Kingma and Welling, 2013).

The **EC** engine is a key module that is used in the following parts of the Algorithm 3.1: (i) definition of different **EC** engines with different parameters (in step 4) (ii) the control of how many different evolutionary searches we want to perform, which we refer to as **EC** runs, to promote the exploration of different areas of the search space (step 5); (iii) generation of instances (x) of a pre-determined class (t) through evolution, a key step of the **EFFECTIVE** algorithm (step 12); and (iv) extract information for the Supervisor module (step 17).

3.4 SUPERVISOR MODULE

The **EFFECTIVE** framework relies on the ability of the **EC** engine to find and exploit the weaknesses of the classifiers to increase the quality of the dataset. The ability of the **EA** to find shortcuts that exploit weaknesses of the fitness assignment scheme is well-known (Baluja et al., 1994; Machado et al., 2007a; Spector and Alpern, 1994; Teller and Veloso, 1995). Adding these instances to the training dataset and re-training the clas-

sifier could promote the correction of its exploitable flaws. However, blindly adding new instances could hinder performance. For instance, assuming the **EC** algorithm converges towards a given instance, generating thousands of exact copies or small variations. Adding all these instances to the dataset is likely to be harmful. With this in mind, after completing all the **EC** runs, the resulting instances are submitted to the Supervisor module, which is responsible for choosing the instances that are going to be added to the training dataset. The Supervisor may be a human observer, or an automatic method able to filter and select instances (Machado et al., 2012a,b). Using a human observer is time-consuming and prone to errors and biases (Correia et al., 2012). However, in some scenarios, human supervision is required, due to the absence of a better solution. Using an automatic supervision method, such as another classifier, implies that the approach is limited by the performance of the Supervisor's classifier. Nevertheless, the Supervisor's classifier is not subjected to evolutionary pressure and, therefore, its weaknesses are less likely to be exploited by the **EC** engine.

The Supervisor's task is performed through the use of a filtering and selection modes. For instance, one may randomly select instances to be added, or use sampling techniques and similarity measures to select and filter the instances. We generalised the selection and filter modes in the following way: (i) during the selection step, a subset of the instances synthesised during the **EC** runs is selected according to a certain criteria and (ii) during filtering, some of the instances of this subset are discarded based on the comparison with other instances of the subset.

One of the goals of the selection stage is to discriminate between the true positives and false positives. We consider the following selection modes: *Aggregator*, *Majority* and *External*.

The most straightforward selection mode is the *Aggregator*, where all the instances generated throughout the **EC** runs identified as containing a pre-determined class are selected.

The idea of the *Majority* mode is to explore the fact that we have several classifiers to estimate if an instance classified as belonging to a pre-determined class is a true or false positive. If the classifier guiding evolution, and as such subjected to evolutionary pressure, classifies an instance as belonging to the positive class while the majority classifies it as belonging to the negative one, then the instance is likely to be a false positive, and, as such, should be selected in order to prevent such kind of errors in the future. Conversely, if the majority agrees with the classification, the instance is likely to be a true positive, and no action is required.

The *External* mode relies on the feedback of an external classifier to select the instances. The rationale is that the external classifier is not subjected, directly or indirectly, to evolutionary pressure, in the sense that its flaws are not exploitable by the **EC** engine. Furthermore, the external classifier was trained using a different dataset, and as such is likely to have different weaknesses. For these reasons, the classifications performed on the evolved instances by this external classifier are prone to be more ac-

curate. In fact, even if the external classifier is weaker in general, it is prone to be more accurate in these specific circumstances since its weaknesses are not being exploited by evolution.

After selecting the instances, the Supervisor performs a filtering step. We consider two filtering modes: *Unequal* and *RMSE*. The first discards instances that are duplicated. The *RMSE* mode, it calculates the root mean square error between all pairs of instances of the sub-set, discarding instances that are below a given *RMSE* threshold (i. e., instances that are similar yet not equal). The contribution of this mode is twofold: promote the scalability of the approach by removing similar and redundant instances; eliminate instances that could bias the training process.

3.5 SETUP AND PARAMETRISATION

The **EFFECTIVE** algorithm is modular and was designed to be generic and scalable. The algorithm assumes a simple, straightforward pipeline: train a classifier using a base training dataset; evolve new instances for a pre-determined class using the trained classifier; select and filter instances to update the training dataset; and repeat. Nevertheless, this pipeline can have dependencies and conditions relate to the problem that we are solving, as it will be presented for the several framework instantiations and all the experiments conducted during this thesis.

We define the modules as single units participating in the algorithm. Nevertheless, both **EC** engine and the **CS** can be a committee composed of more than one unit or one type of algorithm. As such, we conduct experiments where two different **EC** engines operate, one for expanding the positive class and another for the expanding the negative class, using different **EAs**.

The algorithm has several control parameters, including: the stopping criterion, classifiers trained per iteration, and the number of evolutionary runs per classifier. Nevertheless, the framework is modular, and these parameters can be dynamically changed when required. To evaluate the framework's performance, we use pre-determined values, which we adopt for all the experiments made during this thesis.

EVOLVING IMAGES OF A PARTICULAR TYPE

In this Chapter we describe a set of experiments using **EFFECTIVE** to generate images. As seen in the previous Chapter, the framework must act as a generator of *useful* instances. Since we are focusing on the image domain, before getting to *useful* we first must address the question: “How can we generate images of a particular type?”. From the literature about Evolutionary Art systems, we know that with the appropriate representation it is possible to generate any image (Machado et al., 2007a; McCormack, 2007). However, in practice, Evolutionary Art systems that use expression-based schemes tend to generate abstract images. Nevertheless, this did not stop the efforts of authors to evolve images of a particular type and figurative ones (see, e. g., Lewis 2007 and World 1996).

This question has been mainly addressed by two main types of approach: (i) develop tailor made Evolutionary Art systems which resort to representations that promote the discovery of figurative images, usually of a particular type; and (ii) use general purpose Evolutionary Art systems and develop fitness assignment schemes that guide them towards images of a particular type. We are particularly interested in the second approach.

Thus, we address the question posed at the start of this Chapter by expanding the idea of Romero et al. (2003): combining a general purpose evolutionary art system with an image classifier trained to recognise faces, or other types of objects, to evolve images of a particular type. With this idea as the starting point, we developed the first experiments with a prototype system and pipeline of what would become **EFFECTIVE**.

We describe the work done with **EFFECTIVE** at an early stage showing its potential by applying it in different image generation tasks: (i) evolution of faces (Section 4.2) (Machado et al., 2012a); (ii) evolution of figurative images (Section 4.3) (Correia et al., 2013a); (iii) evolution of ambiguous images (Section 4.4) (Machado et al., 2015b); and (iv) evolution of photorealistic faces (Section 4.5) (Correia et al., 2016). All share at least one of the following aspects: either following the **EFFECTIVE** pipeline or using the same **CS** module(s) or the same **EC** engine. Each Section describes the instantiation, setup differences, results, and overall analysis.

4.1 INSTANTIATION

In the work that is described in the next Sections, the framework operates in the following manner: a classifier is selected to represent the **CS** module, several **EC** runs are

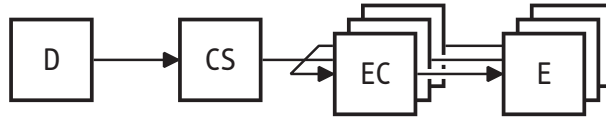


Figure 4.1: Overview of **EFFECTIVE** workflow with one **CS** trained with a dataset D , and multiple **EC** runs that yield different instances E .

deployed and the results are analysed. This process is schematically summed up in Figure 4.1.

As we will describe in the following Sections, the framework instantiation was thought to generate different instances (images) of a particular class. It represents the base version of **EFFECTIVE** algorithm. We need to ascertain if we can generate images of a particular type; therefore, in the first experiments we used an off-the-shelf classifier for the role of **CS**. As such, for the first experiments, we did not have to define a dataset (D) as the schematic shows. As it will be discussed, after showing its potential in the following works, we define our own datasets to train our own classifiers.

In the following Subsections, we present the **CS** and **EC** engines used in the experiments.

4.1.1 Classifier System(s)

The **CS** module is key for the evaluation of the instances. As defined in Section 3.2, it can be any **ML** approach. In this Section, we describe the **CSs** used during this thesis: a cascade classifier and an **ANN**.

In the first experiments, the **CS** module classifier is the cascade classifier trained to detect objects in images. We used this classifier with different feature sets used separately (see Viola and Jones (2001)). It was chosen because of its fast detection algorithm, easy distribution of the algorithm and for being a well-established approach. The code and executables used are included in the OpenCV API¹. The approach uses a set of small features in combination with a variant of the Adaboost (Freund and Schapire, 1997), which can attain efficient classifiers. The classifiers assume the form of a cascade of small and simple classifiers that use single features as the decision rule similar to the description in Section 2.1.1.

The training algorithm for each stage of the cascade classifier can be summarised in these steps:

1. Define the negative and positive datasets, and the desired false alarm and hit rate for the stage;

¹ OpenCV — <http://opencv.willowgarage.com/wiki/>

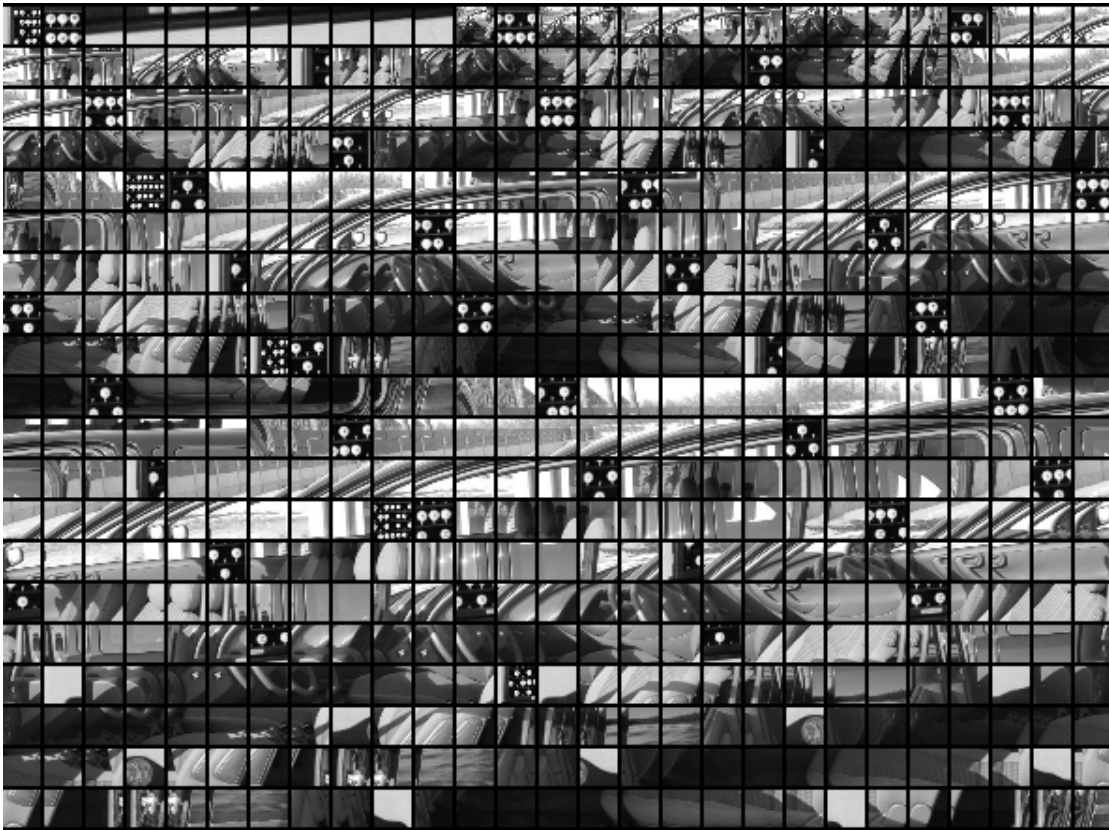


Figure 4.2: Sample of sub-windows of size 20x20 extracted using Sung and Poggio's (1995) sampling algorithm during the training of one stage of the cascade classifier.

2. Initialise the weights for each example, in such a way that the sum of all weights equals 1;
3. Add and train a classifier with only one feature until it achieves the lowest error possible;
4. Update the weights (the wrongly classified instances receive a greater weight);
5. Repeat from step 2 until the desired false alarm, and hit rate are achieved.

The final cascade is the combination of all simple classifiers trained with a desired number of stages. Note that in the OpenCV's implementation the definition of the negative dataset is performed using the subsampling algorithm by Sung and Poggio (1995) on the starting dataset. This means that, in each training stage, an exhaustive search for misclassified sub-windows in the negative image dataset is performed. A sample of these subsamples selected during the training of one stage using this method is presented in Figure 4.2

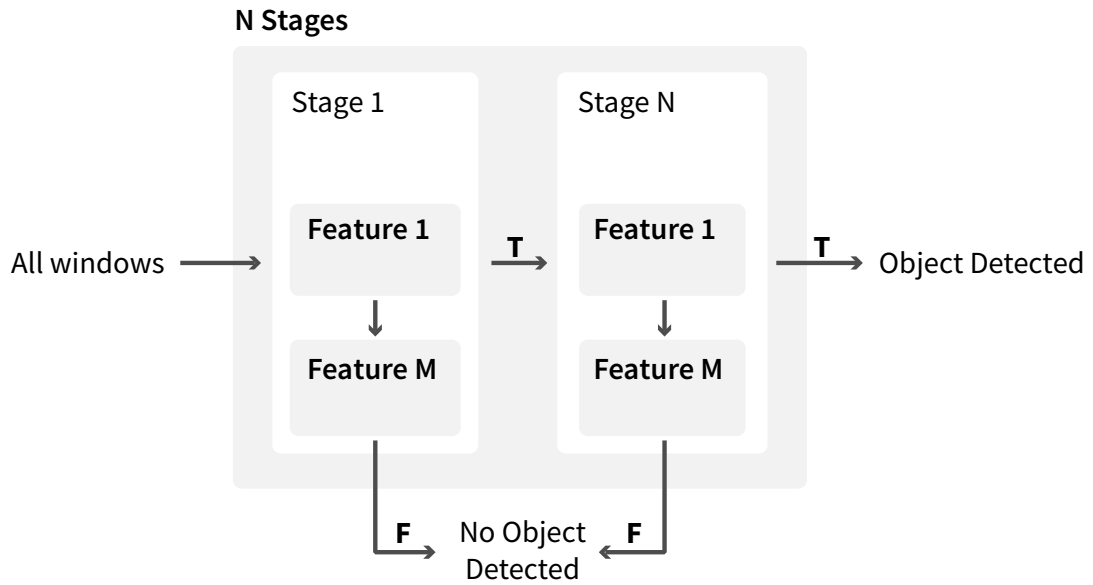


Figure 4.3: Overview of the object detection process with a Cascade Classifier.

The object detection algorithm is a sliding window algorithm and operates in the following manner (for further details see Viola and Jones (2001)):

1. Define w and h as the width and height, respectively, of the input image;
2. Define a window of size $w' \times h'$, e.g. 20×20 ;
3. Define a scale factor s greater than 1. For instance, a scale factor of 1.1 means that the window will be enlarged by 10%;
4. Calculate all windows with size $w' \times h'$ from the position $(0, 0)$ to $(w - w', h - h')$ with 1 pixel increments of the upper left corner;
5. Apply the cascade classifier for each window. The cascade has a group of stage classifiers, as represented in Figure 4.3. Each stage is composed of a group of low-level features that are applied to the window. If the resulting overall value is lower than the stage threshold, the classifier considers that the window does not contain an object and for this reason terminates the search. If it is higher, it continues to the next stage. If all stages are passed, the window is classified as containing the object;
6. Apply s to w' and h' , and go to step 4 until w' exceeds w or h' exceeds h .

Among these low level features we have worked with the Haar (Figure 4.4) and Local Binary Pattern (LBP) (Figure 4.4) which are included in OpenCV. Briefly describing, Haar features are convolutional kernels where each feature is the result of the

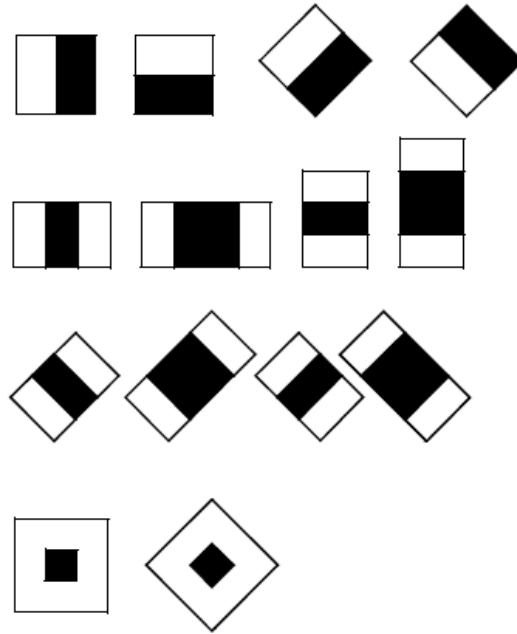


Figure 4.4: Haar features (image from Viola and Jones, 2001).

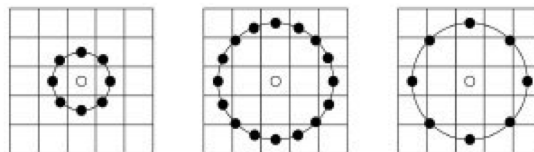


Figure 4.5: Local Binary Pattern features (image from Ahonen et al., 2006).

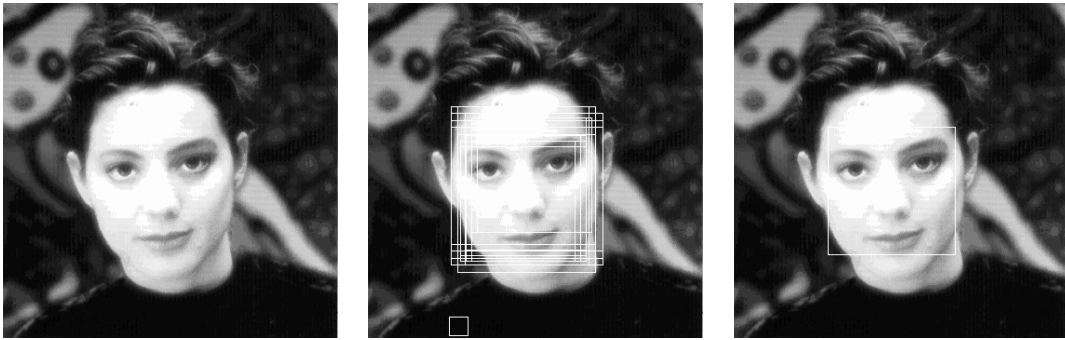


Figure 4.6: Output of the detection's grouping algorithm, with minimum neighbours = 3. On the left we have the input image, in the middle the result after the detection algorithm at different scales and on the right the output image after the grouping algorithm.

computation of subtracting the sum of pixel values under the white rectangle with the sum of pixels belonging to the black rectangle (Viola and Jones, 2001). LBP are texture descriptors that compute local representations by comparing each pixel with its surrounding neighbourhood of pixels (Ojala et al., 1996). As it will be described in the following Sections, both sets of features resulted in different experimental results. These features were considered due to their efficiency and simplicity. There are others, but an exhaustive study on this topic is out of scope for the objectives of this thesis.

Another important part of the detection phase, more precisely in the sliding window process, is the algorithm that groups the different detections at different scales. The built-in algorithm from OpenCV was used for that effect. The different detections windows are combined into single detection by clustering detections with similar size and locations up to a threshold. Small clusters with less than or equal to that threshold are discarded. This is controlled by the parameter "minimum neighbours" presented in the experimental setups of the works that used an object detector as the classifier. For each cluster that remains after the filtering, the average detection rectangle is computed and considered as one detection. An example can be seen in Figure 4.6.

The cascade classifiers were used in all the Chapters that involved object detection, i. e., Chapters 4, 5 and 6. In Chapter 7 we use ANN. In particular, the ANN is a feed-forward network, with one hidden layer and two output neurons. It is trained with standard backpropagation as described in Section 2.1.1. The classifier was built using WEKA's² FastNeuralNetwork. WEKA is a workbench for machine learning with a significant number of algorithms and tools available (Hall et al., 2009).

² <http://www.cs.waikato.ac.nz/ml/weka/> WEKA 3: Data Mining Software in Java

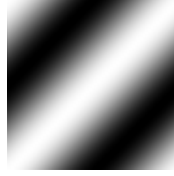


Figure 4.7: Example of an individual with the expression-tree $\cos(x + y)$ rendered at the size of 64 by 64 pixels.

4.1.2 Evolutionary Engine(s)

The **EC** engine is a core part of the framework as described in Section 3.3. Along the several experiments performed with the framework, most of the work was done using a **GP** expression based approach. In the first set of experiments, we used **NEvAr** (Machado and Cardoso, 2002) and in the last experiments we used **NORBERT** (Vinhas, 2015). Both of them are expression-based evolutionary engines that are described in Section 4.1.2.1. We also created a tailor made solution using a **GA** described in Section 4.1.2.2.

4.1.2.1 Expression-Based

The expression-based engines used in this thesis operate similarly and have common features. **NORBERT** was inspired in **NEvAr** which in turn is inspired by the work of Sims (1991). Both are general purpose, expression-based, **GP** engines that allow the evolution of a population of images. A thorough description of each engine is out of this thesis' scope but can be found in Vinhas (2015) and Machado et al. (2007a), respectively. Their core functionality is described in this Section.

As stated, both **NEvAr** and **NORBERT** allow the evolution of population of images. The individual's genotypes are expression-trees composed of functions and terminals. The function set includes mathematical and logical operations; the terminal set is composed of two variables, x and y , and constant values. The phenotypes are images, rendered by evaluating the expression-trees for different values of x and y , which serve both as terminal values and image coordinates. In concrete, to calculate the value of the pixel in the top left corner of an image (the (0,0) coordinates), one assigns zero to x and y and evaluates the expression-tree; a similar procedure is performed for all the other pixel coordinates. Figure 4.7 presents an example of the phenotype for the individual of the following genotype: $\cos(x + y)$.

NORBERT has particular functionalities and configurations available that are going to be described in Chapter 7. Furthermore, due to the multiplicity and particularities of each experiment, the fitness function(s) and parameters are detailed in their corresponding Sections.

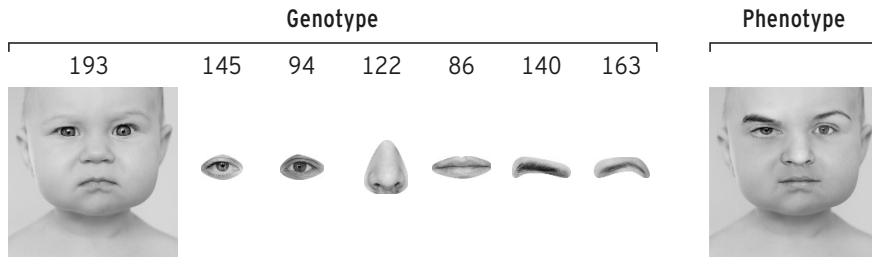


Figure 4.8: Genotype and phenotype of an individual. The genotype consists in the tuple of integers (*face*, *left eye*, *right eye*, *nose*, *mouth*, *left eyebrow*, *right eyebrow*). Each integer encodes an index of an annotated example (image under each integer encoded in the genotype). The phenotype consists in a composite of the face parts encoded in the genotype.

4.1.2.2 *eXploit-Faces Engine*

Unlike the previous [EC](#) engines, this one was created with a particular purpose. Since it is used in three experiments along this thesis, the general description is done in this Section. The idea behind this particular approach is to evolve combinations of items guided by a proper fitness function. It was developed as an experiment for generating photorealistic images of faces from parts of existing images of faces (Section [4.5](#)).

For this particular purpose, we developed a tool capable of recombining parts of pre-annotated face parts. Each image has an identifier, an index, and each annotated part shares that same index. e. g., for an image with an index = 10 with the following annotations: face, left eye, right eye, nose, mouth, left eyebrow and right eyebrow; will have the following set of indexes: {10, 10, 10, 10, 10, 10, 10}. This representation allowed us to explore combinations of different indexes to generate different faces. With this representation we have $\#D^n$ different possible faces to generate, where $\#D$ is the number of instances in the image dataset, n is the number of parts. For this particular application involving faces, we would have $n = 7$ (or $n = 5$ as it will be described later when we performed tests pairing eyes and eyebrows). As the number of $\#D$ grows the number of different faces that we can generate grow at a polynomial rate. We need a way to explore and choose interesting individuals from the different possibilities. To traverse such space, we choose a conventional [GA](#) using a proper fitness function to evolve populations of sets of indexes later to be translated to a composite face, i. e., a face made from other faces' parts.

Thus, this [EC](#) engine is a conventional [GA](#) where the individuals are faces constructed from parts of different faces. Figure [4.8](#) explains the genotype of each individual and the corresponding phenotype. Each genotype is mapped into a phenotype by creating a composite face, i. e., the parts of faces encoded in the genotype are placed over a base face that is also encoded in the genotype. This process is accomplished by using a clone algorithm that allows the seamless placement of an image upon another (Pérez et al., [2003](#)).

4.2 EVOLUTION OF FACES

As the first proof of concept of the approach as a generator, we tested it on a frontal face generation problem. Following Romero et al. (2003) idea, we use a general purpose art system to generate images that contain faces. This Section revolves around the work done in Machado et al. (2012a) and part of Correia et al. (2013a). We choose face generation because face detection is a topic of interest and research, where applications that employ this kind of systems are becoming widespread in an almost “plug and play” fashion. For instance, they can be found in search engines, social networks, incorporated in cameras, or in applications on smartphones.

Our approach was mainly informed by previous research, e. g. Baluja et al. (1994), Machado et al. (2007a), and Saunders and Gero (2001), where classifier systems, namely ANN, are used to guide the evolutionary runs. However, among others, our approach operates these experiments with the following discriminating characteristics:

- Using an off-the-shelf classifier instead of developed one to guide evolution;
- The goal is to evolve specific figurative images, i. e., faces, while the mentioned classifiers try to assess aesthetics, style or novelty;
- A Haar cascade classifier is used instead of ANN.

We have already discussed most of these details in the previous Section. Thus, in the following Sections we describe the experimental setup, we present the fitness assignment scheme, and afterwards we discuss the results.

4.2.1 Experimental Setup

In this work, we performed 30 independent runs of the framework described in the previous chapter. The framework proposes the use E independent evolutionary runs. However, we are primarily interested in assessing the contributions that each EC run may bring. Thus, for the scope of this Section, we set $E = 1$ and perform 30 independent runs.

In order to conduct the experiments, three pre-trained off-the-shelf classifiers were used. These were obtained from Lienhart’s (Lienhart and Maydt, 2002) website³ and will be named C1⁴, C2⁵, C3⁶. Documentation reports that these classifiers were trained with 2000 positive and 3000 negative samples. C1 is a stump cascade classifier trained with a 20×20 window size; C2 is a tree cascade classifier (i. e., similar to a

³ Haar Cascades – <http://alereimondo.no-ip.org/OpenCV/34>

⁴ “haarcascade_frontalface_alt.xml”

⁵ “haarcascade_frontalface_alt2.xml”

⁶ “haarcascade_frontalface_default.xml”

Table 4.1: Parameters of the GP engine. See Machado et al. (2007a) for a detailed description.

Parameter	Setting
Population size	50
Number of generations	100
Crossover probability	0.8 (per individual)
Mutation probability	0.05 (per node)
Mutation operators	sub-tree swap, sub-tree replacement, node insertion, node deletion, node mutation
Initialisation method	ramped half-and-half
Initial maximum depth	5
Mutation max tree depth	3
Function set	$+$, $-$, \times , $/$, min, max, abs, neg, warp, sign, sqrt, pow, mdist, sin, cos, if
Terminal set	x , y , random constants

decision tree that is applied per stage) with a 20×20 window size; and C_3 is a stump cascade classifier but with a 24×24 window size.

For these experiments we use NEvAr as the EC engine. The settings presented in Table 4.1 are similar to those used in previous experimentation in different problems. Since the used classifiers only deal with greyscale information, the GP engine was also limited to the generation of greyscale images.

The goal of the EC engine is to evolve instances that the CS classifies as containing faces. In Figure 4.9 an overview of an EC run in the context of the face detection problem is presented. The fitness of an individual depends on the results of submitting it to the CS i. e., the fitness is a product of the outcome of the classification task.

4.2.1.1 Fitness Assignment

Fitness assignment is crucial for any EA, and therefore it holds large importance for the success of the described system. The goal is to evolve images that the face detector classifies as faces. However, the face detector returns a binary output per input, which is inappropriate to guide evolution. A binary function gives no information of how close an individual is to be a valid solution to the problem and, as such, the fitness landscape would be deceptive.

At this point, we concluded that it is necessary to extract additional information from the face detection process to build a suitable fitness function. This is achieved by accessing internal results of the classification task that give an indication of the degree of certainty in the classification. In several informal experiments, we focused

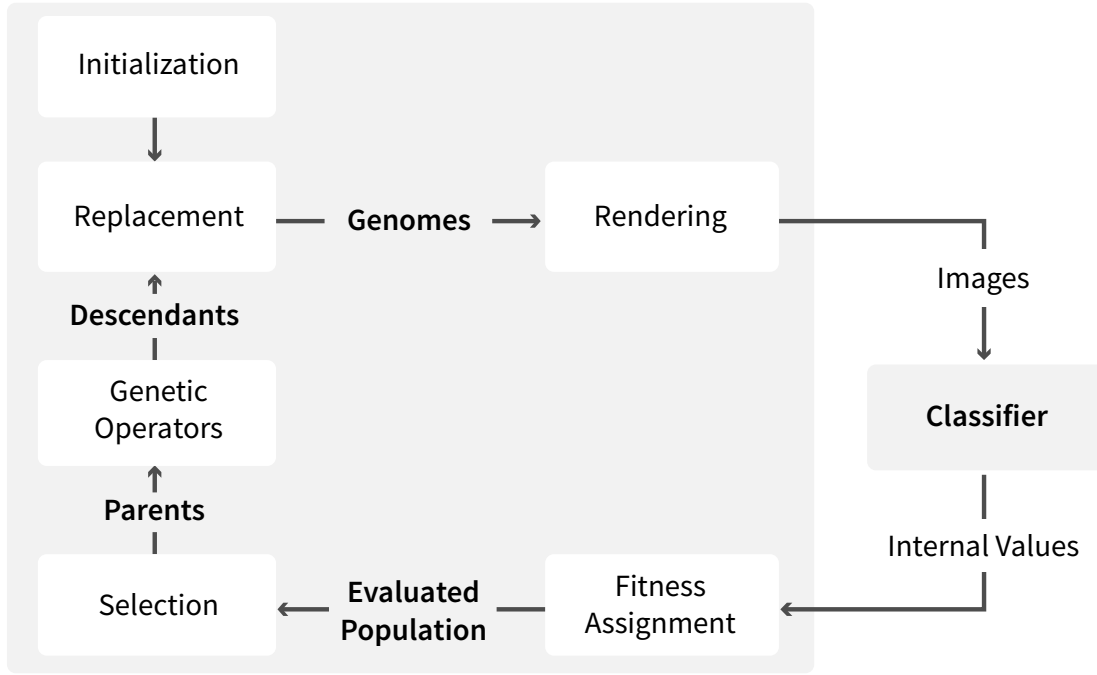


Figure 4.9: Overview of an EC run.

on developing an appropriate fitness function, by analysing the results of several runs, by trial and error, and by incremental improvements and refinements. We eventually settled on the following formula, which takes advantage of the cascade structure of the classifier:

$$\text{fitness}(x) = \sum_i^{\text{nstages}_x} \text{stagedif}_x(i) * i + \text{nstages}_x * 10 \quad (4.1)$$

In a nutshell, images that go through several classification stages, and that may be closer to be classified as a face, have higher fitness than those rejected in early stages. Variables nstages_x and $\text{stagedif}_x(i)$ are extracted from the face detection algorithm. Variable nstages_x , holds the number of stages that the image x has successfully passed in the cascade. The rationale is the following: an image that passes several stages is likely to be closer to being recognised as having a face than one that passes fewer stages. In other words, passing several stages is a pre-condition to being identified as an image that contains a face. Variable $\text{stagedif}_x(i)$ holds the maximum difference between the threshold necessary to overcome stage i and the value attained by the image at the i^{th} stage. Images that are clearly above the thresholds are preferred over

ones that are only slightly above them. Obviously, this fitness function is only one of the several possible ones.

4.2.2 *Experimental Results*

Figure 4.10 summarises the results achieved regarding mean fitness and maximum fitness per run. Since the fitness values attained by different classifiers are not comparable, the values are normalised by dividing the raw fitness by the mean's maximum achieved in each classifier's test. Each chart displays the fitness attained by the classifier used to guide fitness and also the fitness that would be assigned by the two classifiers that had no interference in the run.

An analysis of these charts reveals interesting aspects concerning similarity among classifiers. As it can be observed, the curves of classifiers C1 and C2 vary in similar ways, particularly regarding maximum average fitness, independently of which classifier is guiding the run, which indicates that these classifiers are strongly correlated. In contrast, fitness according to classifier C3 only reaches high values when C3 is used to guide the evolutionary runs. As a whole, these results suggest that classifier C3 is more robust than C1 and C2, in the sense that it is less likely to classify non-face images as faces. Viola and Jones (2001) arrive at a similar conclusion based on experiments done in a non-evolutionary context.

The EC engine was able to find images classified as faces in all of the 90 performed runs. However, and somewhat surprisingly, from a human perspective, most of the runs did not evolve images that look like faces (obviously this statement has a degree of subjectivity). Thus, in most evolutionary runs the GP engine exploited the limitations of the classifier and found "shortcuts" that allowed it to improve fitness and evolve images that are classified as faces, without evolving images that look like faces (see Figure 4.11). The ability of EC to find such shortcuts and exploit weaknesses of the fitness assignment scheme has been reported in previous studies (see, e. g., Baluja et al. (1994), Machado et al. (2007a), Spector and Alpern (1994), and Teller and Veloso (1995)). These results open a series of possibilities, including the use of this approach to assess the robustness of face detection systems, and also the use of evolved images as part of the training set of these classifiers to overcome some of their shortcomings. This line of research was explored and is presented in Chapter 5.

According to our subjective assessment, some of the runs were able to find images that resemble a frontal human face (5 using C1, 4 using C2, and 5 using C3). Figure 4.12 shows the evolution of the best individual, per generation, during the course one of those runs. We rendered the fittest individual in the last generation from the same run and its corresponding expression tree, as can be seen in Figure 4.13. The tree was created using EvoLutlinary Computation vIsualizaTion (ELICIT), a tool that allows the visual exploration of EC algorithms (Cruz et al., 2015).

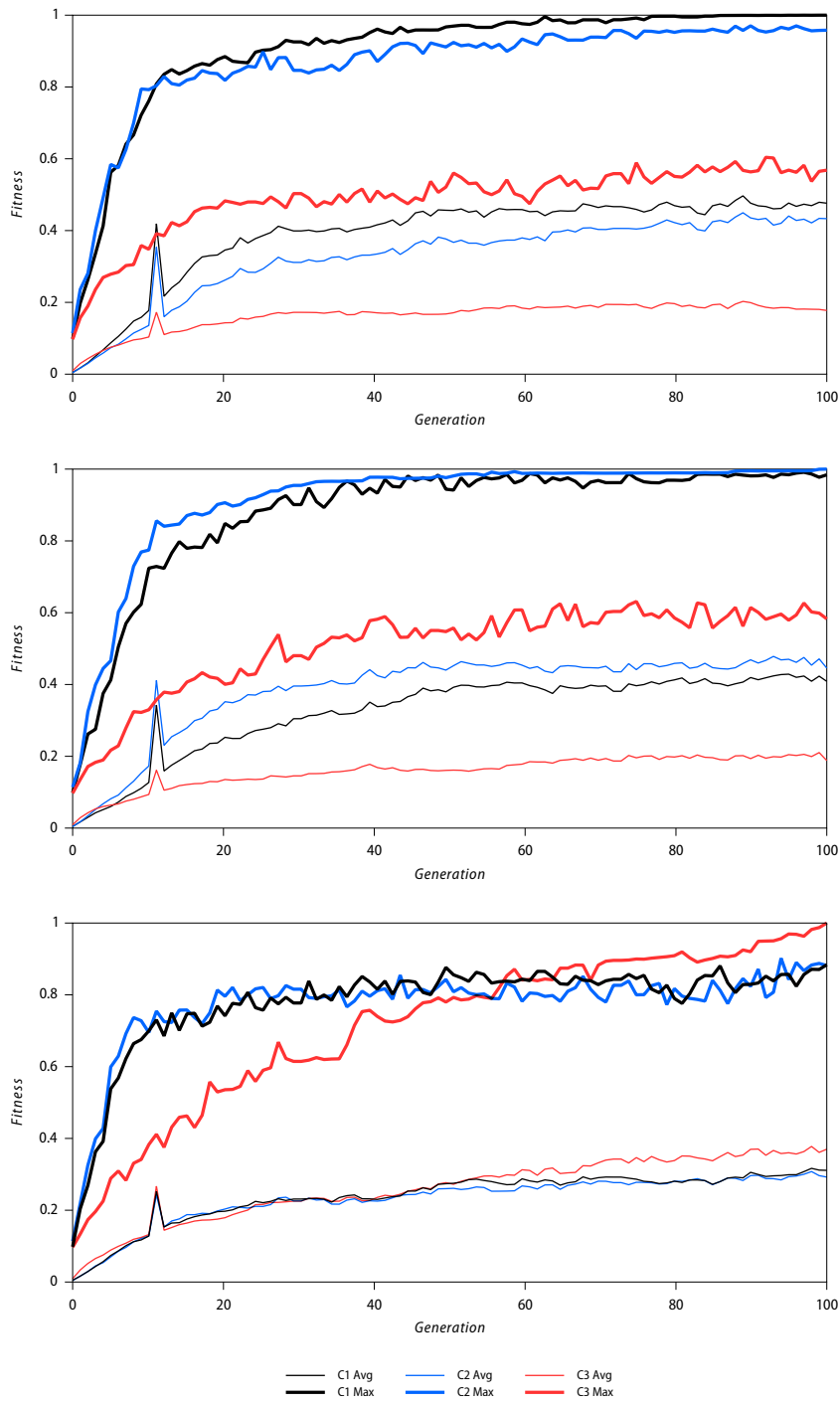


Figure 4.10: Evolution of the average and maximum fitness when using C1 (top), C2 (middle), and C3 (bottom) to assign fitness. Results are averages of 30 independent runs.

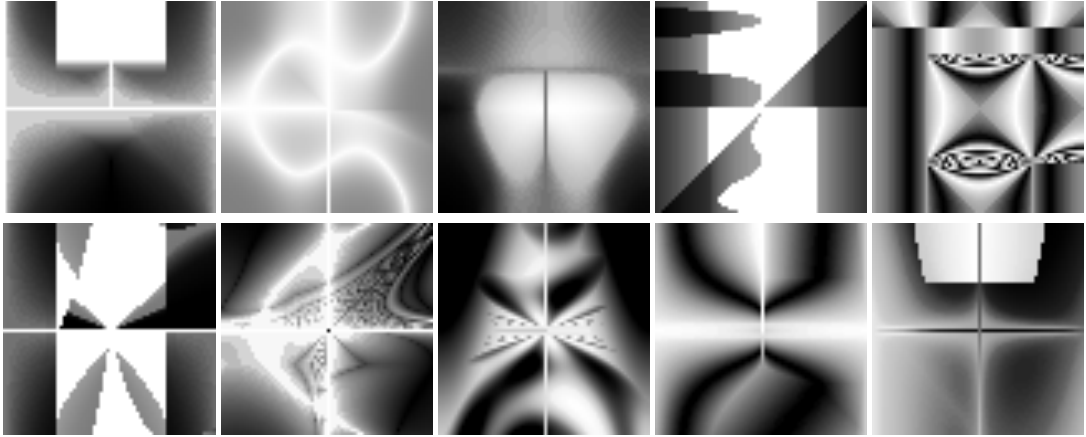


Figure 4.11: Examples of evolved images identified as faces by the classifiers that do not resemble faces from a human perspective.

In Figure 4.14 we show some of the most, from a subjective perspective, interesting evolved images. These results show the ability of the GP engine to create figurative images, which are reminiscent of human faces. Several of these images are evocative of faces of cartoon characters (e. g., the first image of the first row of Figure 4.14, which has been described by several of our co-workers as Wolverine’s face) and African masks (e. g., the last image of the first row of Figure 4.14). This result may reveal a tendency towards the exaggeration of facial features, and hence caricature, which is consistent with the fitness assignment scheme, in the sense that the presence of distinguishable facial features may promote face detection.

We could say that some of these images provoke pareidolia (as shown in Figure 4.15), a psychological phenomenon in which the mind responds to a stimulus perceiving a familiar pattern where none exists. Nevertheless, some of the main features of a typical face (e. g., eyes, nose, face outline) are present in the images.

4.2.2.1 Exploration using Local Binary Pattern features

As mentioned in Section 4.1.1, the cascade classifier can be combined with another type of features, namely the LBP features. Based on the promising results attained while using Haar features, in this Section we explore the use of LBP for the same purpose. We expect to minimise some of the known limitations of the Haar features and improve our results. LBP features are very robust regarding greyscale variations, (which can be caused, e. g., by changes in illumination) intensity since the operator is, by definition, invariant against any monotonic transformation of the greyscale. This aspect is very attractive in situations where nonuniform illumination conditions are a concern, e. g., in face detection.

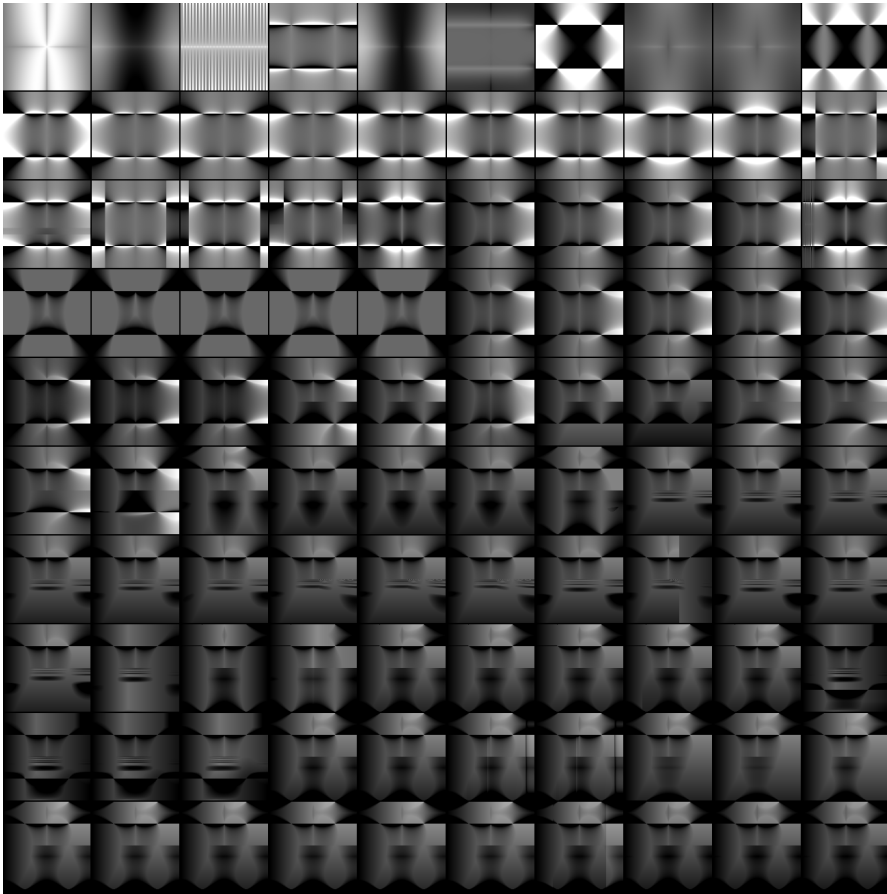


Figure 4.12: Fittest individual per generation. From the first generation (top left) to the last generation (bottom right).

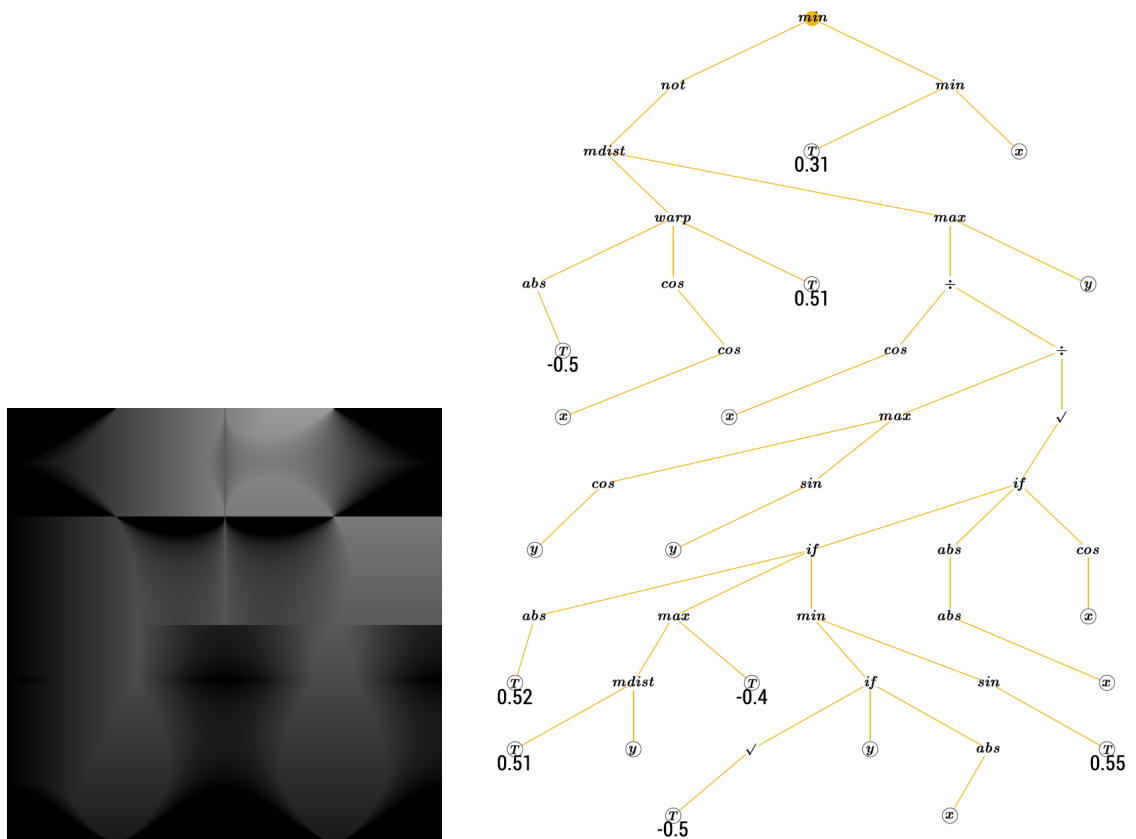


Figure 4.13: On the left, a rendering of the fittest individual of the last generation rendered. On the right, the genotype of the individual.

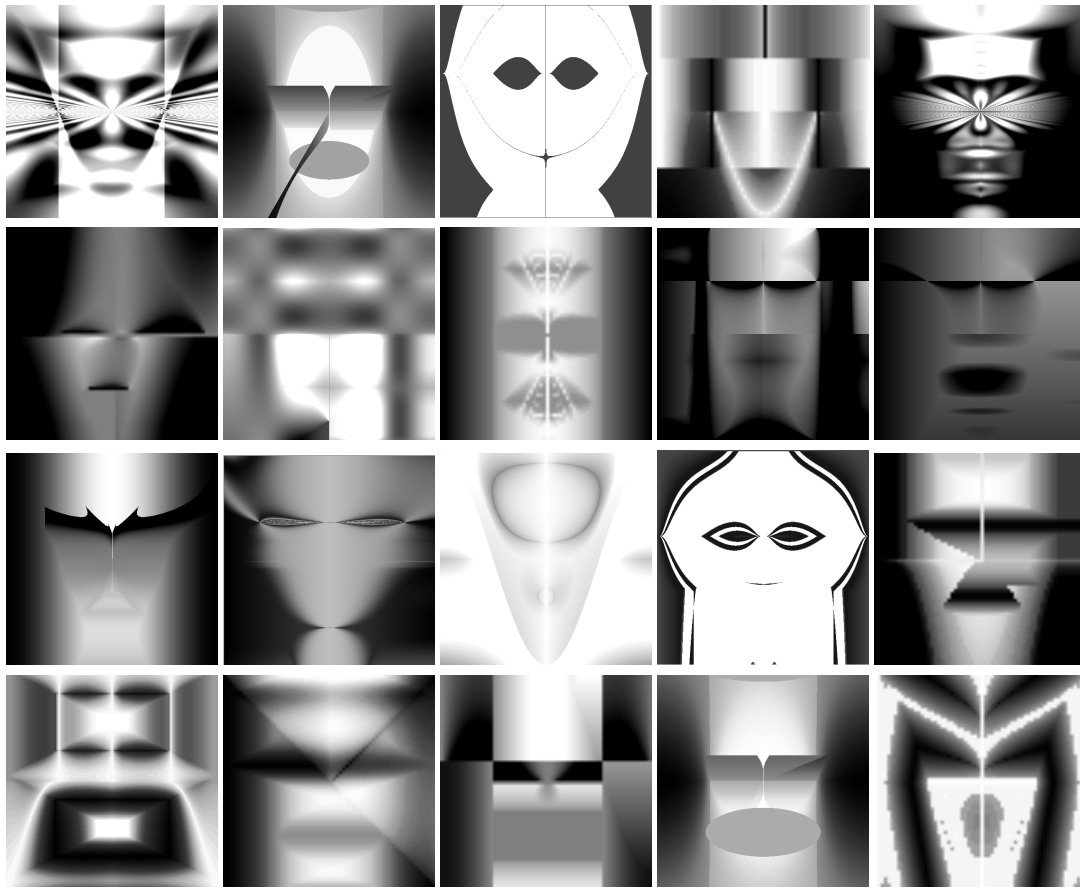


Figure 4.14: Examples of some of the most interesting images that have been evolved, considered to be faces by the classifier.

In order to conduct the tests we resort to the available off-the-shelf classifiers from OpenCV (“LBP1” from file “lbpcascade_frontalface.xml”) and from Vision-ary⁷ (“LBP2”). As reported by the documentation, *LBP1* was trained with 3000 positive samples and 1500 negative samples per cascade stage, whereas *LBP2* was trained with 5000 positive samples and 7000 negative samples. In theory, since it was trained with more samples per stage, *LBP2* should be more robust than *LBP1*.

The **EC** engine used was **NORBERT** with the same parameters used in the previous experiment. In terms of evolutionary process, as we can see in Figure 4.16 and Figure 4.17, using either **LBP** classifier results on a similar evolutionary curve. As previously the results are normalised by the maximum fitness observed by its corresponding classifier and concern 30 independent evolutionary runs.

⁷ Vision-Ary cascades—<http://www.vision-ary.net/2015/03/boost-the-world-face-detection/>



Figure 4.15: Examples of pareidolia phenomenon in everyday objects.

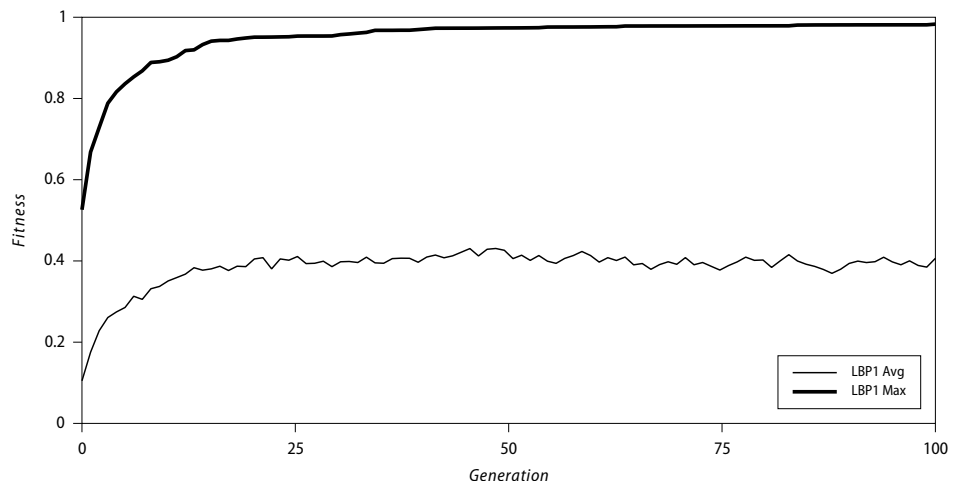


Figure 4.16: Evolution of the average and maximum fitness using *LBP1*.

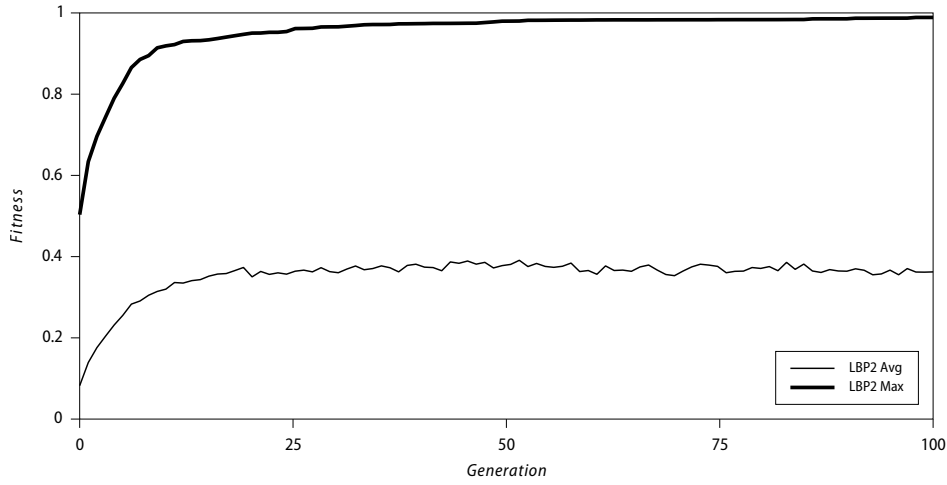


Figure 4.17: Evolution of the average and maximum fitness using *LBP2*.

From a subjective perspective, we can say that some of the runs were also able to generate images that resemble frontal human faces: 6 from *LBP1* and 11 from *LBP2*. Regarding visuals, there are some differences when comparing with the Haar. Figure 4.18 show some of the most interesting individuals generated along the runs. The fact that *LBP2* was trained with more examples and yields more interesting runs points to the fact that it is indeed more robust, pushing the EC engine to generate images that contain more frontal facial features so it can be detected as face. Most of the presented examples have face resembling features, e. g., eyes and mouth; almost like smiles/emojis (third row in Figure 4.18). Similar to the Haar results these features are exaggerated and bigger overall.

On the other hand, we also have some examples that do not resemble faces at all. As Figure 4.19 shows, these appear to be less noisy than Haar. Furthermore, some of the runs that have been detected as faces but do not resemble one, the images tended to be blurry, oval-shaped and vertically symmetrical as shown in Figure 4.20. Some of them resemble blurry frontal face outlines (third and fourth image from the bottom row of Figure 4.20). Similar to the observation in the experiment with Haar features, these images also explore shortcomings of the classifier.

4.2.3 Summary

The goal of the current Section was to generate images by evolutionary means, without resorting to representations specifically tailored to promote the evolution of images of a certain kind. We explored EFFECTIVE idea by using a general-purpose expression-

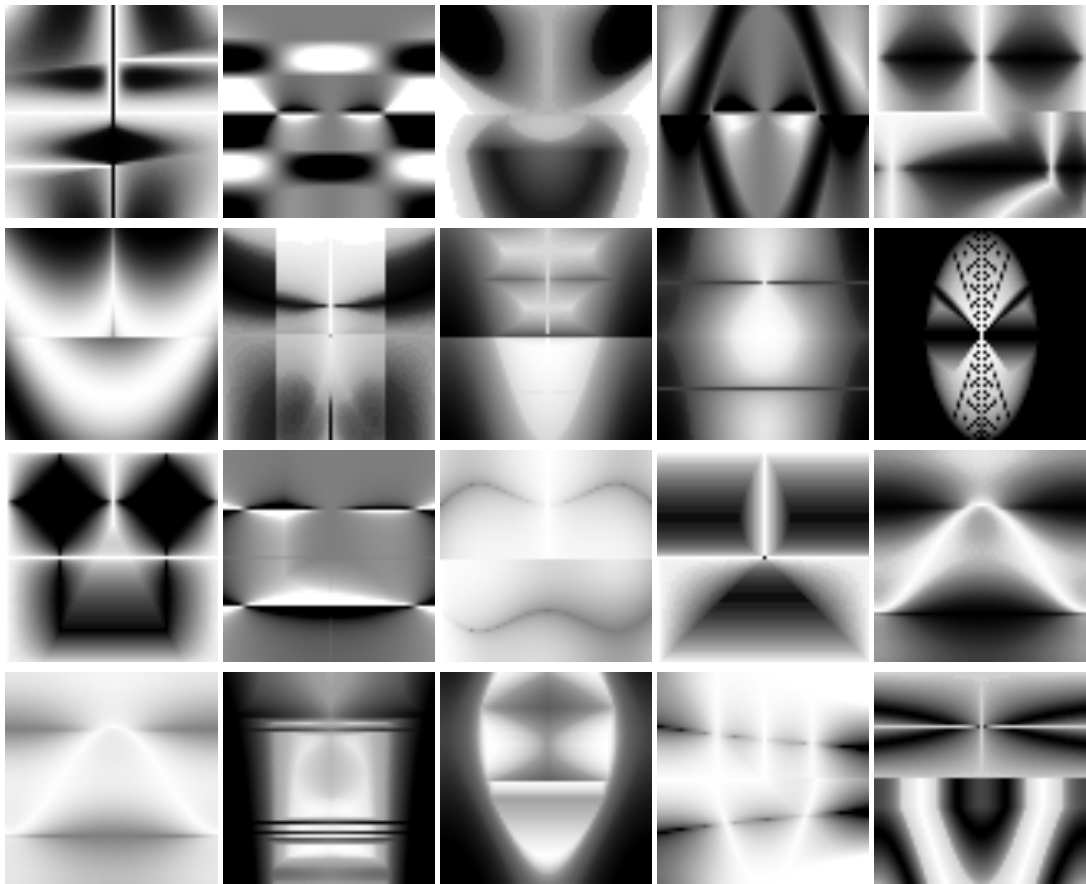


Figure 4.18: Example of images detected as faces by **LBP** classifiers that resemble faces.

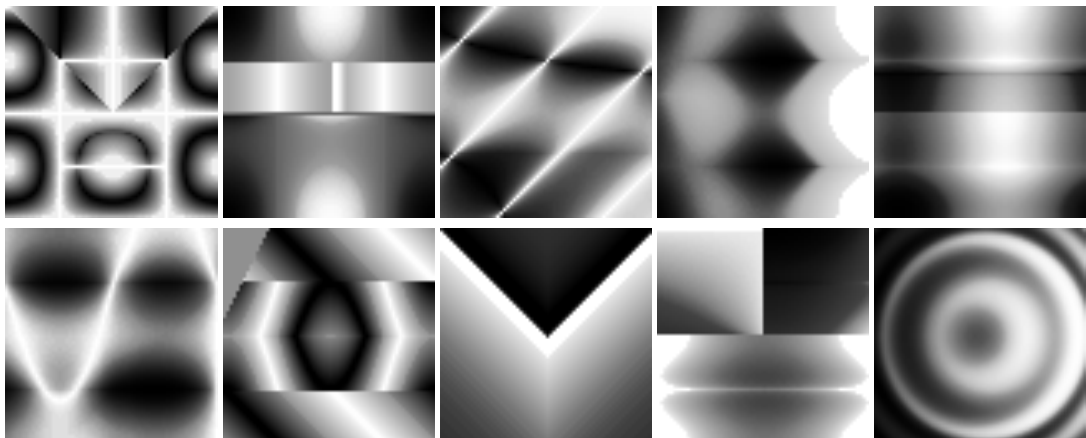


Figure 4.19: Example of images detected as faces by **LBP** classifiers which do not resemble faces.

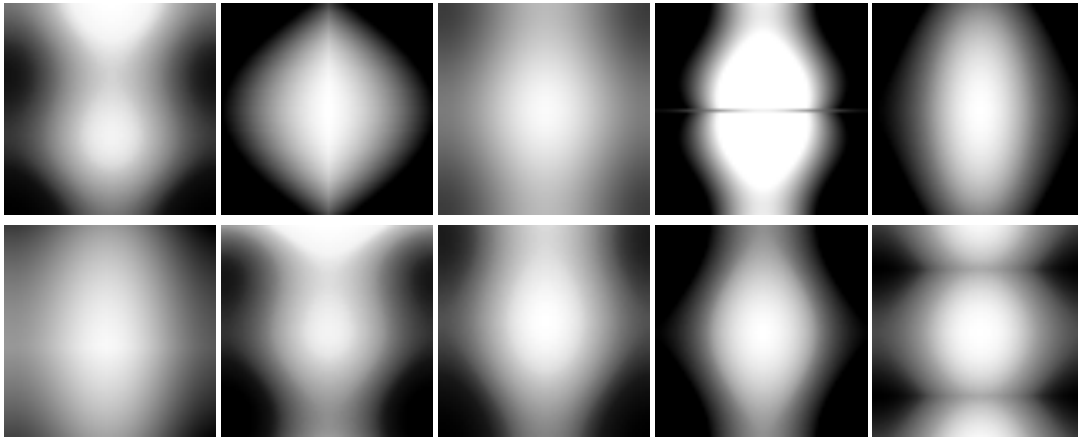


Figure 4.20: Example of images detected as faces by **LBP** classifiers which are mainly blurred images.

based **GP** image generation engine and off-the-shelf face detector system. Internal results of the classification task are employed to build a fitness function.

We did experiments with two types of features: Haar and **LBP**. Thus, the first experimental results attained in 90 independent evolutionary runs show the ability of a **GP** engine to find and exploit shortcomings of the classifier. They also demonstrate the ability of the framework to evolve images that are evocative of human faces and masks. The other set of experiments with **LBP** features revealed similar results. The images that are evocative of human frontal faces presented larger and sharper features. The examples that were not evocative also show the existence of symmetrical objects or blurry images that we can consider as blurry face outlines.

The images evolved in different runs can be combined, refined and explored for artistic purposes by using user-guided evolution or automatic fitness assignment schemes, which take into account aesthetic or stylistic properties. In this regard, the plasticity of the expression-based representation may be a valuable asset.

Finally, based on the results attained, the ability of **EC** to find shortcomings of the classifier led us to think that, the instances could be used to improve classifier performance. This idea will be discussed further in Chapter 5.

4.3 EVOLUTION OF FIGURATIVE IMAGES

In the previous Section, we presented a system that allowed the evolution of images resembling human faces by combining a general-purpose, expression-based, Evolutionary Art system with an off-the-shelf face detector. The internal values of the detection task were used for fitness assignment. The results showed that it was possible to guide evolution and evolve images evocative of human faces. This Section intends to explore

Table 4.2: Haar cascade classifier training parameters.

Parameter	Setting
Number of stages	30
Min True Positive rate per stage	99.9%
Max False Positive rate per stage	50%
Object Width	20 or 40 (breasts, leaf)
Object Height	20 or 40 (leaf)
Haar Features	ALL
Number of splits	1
Adaboost Algorithm	GentleAdaboost

such approach further and is related to the work done in Correia et al. (2013a). The work presented in this Section expands the work in two aspects:

- Using both off-the-shelf classifiers and purpose-built detectors;
- Evolving other kind of objects such as lips, breasts and leaves.

4.3.1 Experimental Setup

The objects that we are interested in evolving are the following: lips, breasts and leaves. For the first one, we used an off-the-shelf classifier that was already trained and used by other researchers in different lines of investigation (Lienhart and Maydt, 2002; Lienhart et al., 2002; Santana et al., 2008). For the last two we created our own classifiers, by choosing suitable datasets and training the respective object detector.

In order to construct an object detector we need to design two datasets: (i) positive – examples of images that contain the object we want to detect and; (ii) negative – images that do not contain the object (Figure 4.21). Furthermore, we must crop the location of the object in the images (Figure 4.22) in order to build the ground truth file that will be used for training. For these experiments, the negative dataset was attained by picking images from a random search using image search engines, and from the Caltech-256 Object Category dataset (Fei-Fei et al., 2004). In what concerns the positive datasets: the breast object detector was built by searching images on the web; and the leaf dataset was obtained from the Caltech-256 Object Category dataset and from web searches.

After choosing datasets, we must also define the training parameters. Table 4.2 presents the parameters used for training of the cascade classifier. The parameters were selected from the work by Lienhart et al. (2002). By defining a high *number of stages* we are creating several stages that the images must overcome to be considered a



Figure 4.21: Negatives instances used for training the cascade classifiers.



Figure 4.22: Instances used to train a cascade classifier for leaf detection. On the top row the original image and on the bottom row the clipped instance used for training. (Image from (Fei-Fei et al., 2004))

Table 4.3: Parameters of the EC engine.

Parameter	Setting
Population size	100
Number of generations	100
Crossover probability	0.8 (per individual)
Mutation probability	0.05 (per node)
Mutation operators	sub-tree swap, sub-tree replacement, node insertion, node deletion, node mutation
Initialization method	ramped half-and-half
Initial maximum depth	5
Mutation max tree depth	3
Function set	$+$, $-$, \times , $/$, min, max, abs, neg, warp, sign, sqrt, pow, mdist, sin, cos, if
Terminal set	x , y , random constants

positive example. The high *true positive rate* ensures that almost every positive example is learned per stage. The max *false positive rate* creates some margin for error, allowing the training to achieve the *minimum true positive rate* per stage and a low positive rate at the end of the cascade.

Once the classifiers are obtained, they are used to assign fitness in the course of the evolutionary runs in an attempt to find images that are recognised as lips, breasts and leaves. We performed 30 independent evolutionary runs for each of these classes. In summary, we have 3 classifiers, with 30 independent EC runs, totalling 90 EC runs.

The settings of the EC engine, presented in table 4.3, are similar to those used in previous experimentation in different problem domains. Since the used classifiers only deal with greyscale information, the EC engine is limited to the generation of greyscale images. The population size used is 100 while in previous experiments we used a population size of 50 (Machado et al., 2012a). This allows us to sample a larger portion of the search space, contributing to the discovery of images that fit the positive class.

In all evolutionary runs, the EC engine was able to evolve images classified as the respective objects. Similarly to the behaviour reported by Machado et al. (2012a), the EC engine was able to exploit weaknesses of the classifier, that is, the evolved images are classified as the object, but, from a human perspective, they often fail to resemble the object. In Figure 4.23 we present examples of such failures. As it can be observed, it is hard to recognise breasts, leaves or lips in the presented images.

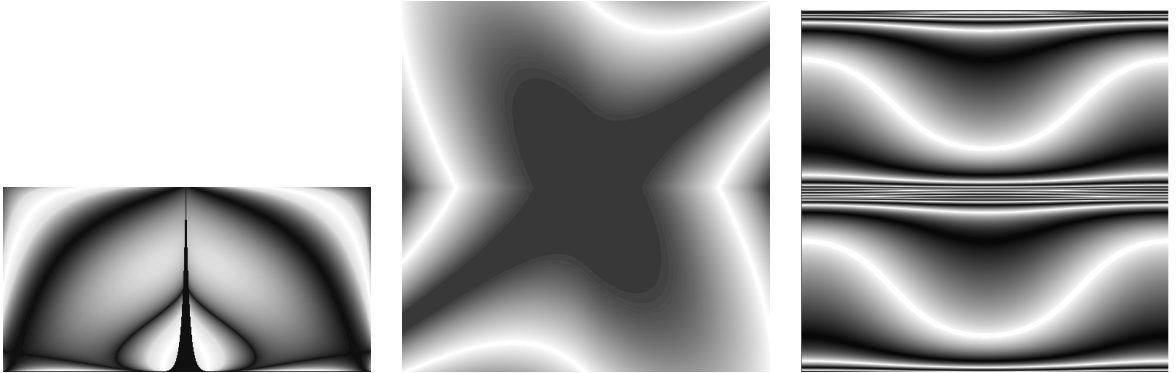


Figure 4.23: Evolved images identified as objects by the classifiers that do not resemble the corresponding objects from a human perspective. From left to right, these images were detected as breasts, leaves and lips.

According to our subjective assessment, some runs were able to find images that resemble the object that we are trying to evolve. These add up to 5 for the lip detector, 4 for the breast detector and 4 for the leaf detector.

In Figures 4.24, 4.25 and 4.26 we show, according to our subjective assessment, some of the most interesting images evolved. These results allow us to state that, at least in some instances, the EC engine was able to create figurative images evocative of the objects that the object detector was designed to recognise as belonging to the positive class.

In what concerns the images resulting from the runs where a lip detector was used to assign fitness, we consider that their resemblance with lips, caricatures of lips, or lip logos, is self-evident. The iconic nature of the images of the last row is particularly appealing to us.

The results obtained with the breast detector reveal images with well-defined or exaggerated features. We found little variety in these runs, with changes occurring mostly at the pixel intensity and contrast level. Most of the runs resulted in unrecognisable images (see Figure 4.23), which is surprising since the nature of the function set would lead us to believe that it should be relatively easy to evolve such images. Nevertheless, the successful runs present images that are clearly evocative of breasts.

Finally, the images from the leaf detector, vary in type and shape. They share however a common feature: they tend to be minimalist, resembling logos. In each of the images of the first row, the detector identified two leaf shapes. On the others, a single leaf shape was detected.

In general, when the runs successfully evolve images that actually resemble the desired object, they tend to generate images that exaggerate the key features of the class. This is entirely consistent with the fitness assignment scheme that values images that are recognised with a high degree of certainty. This constitutes a valuable side

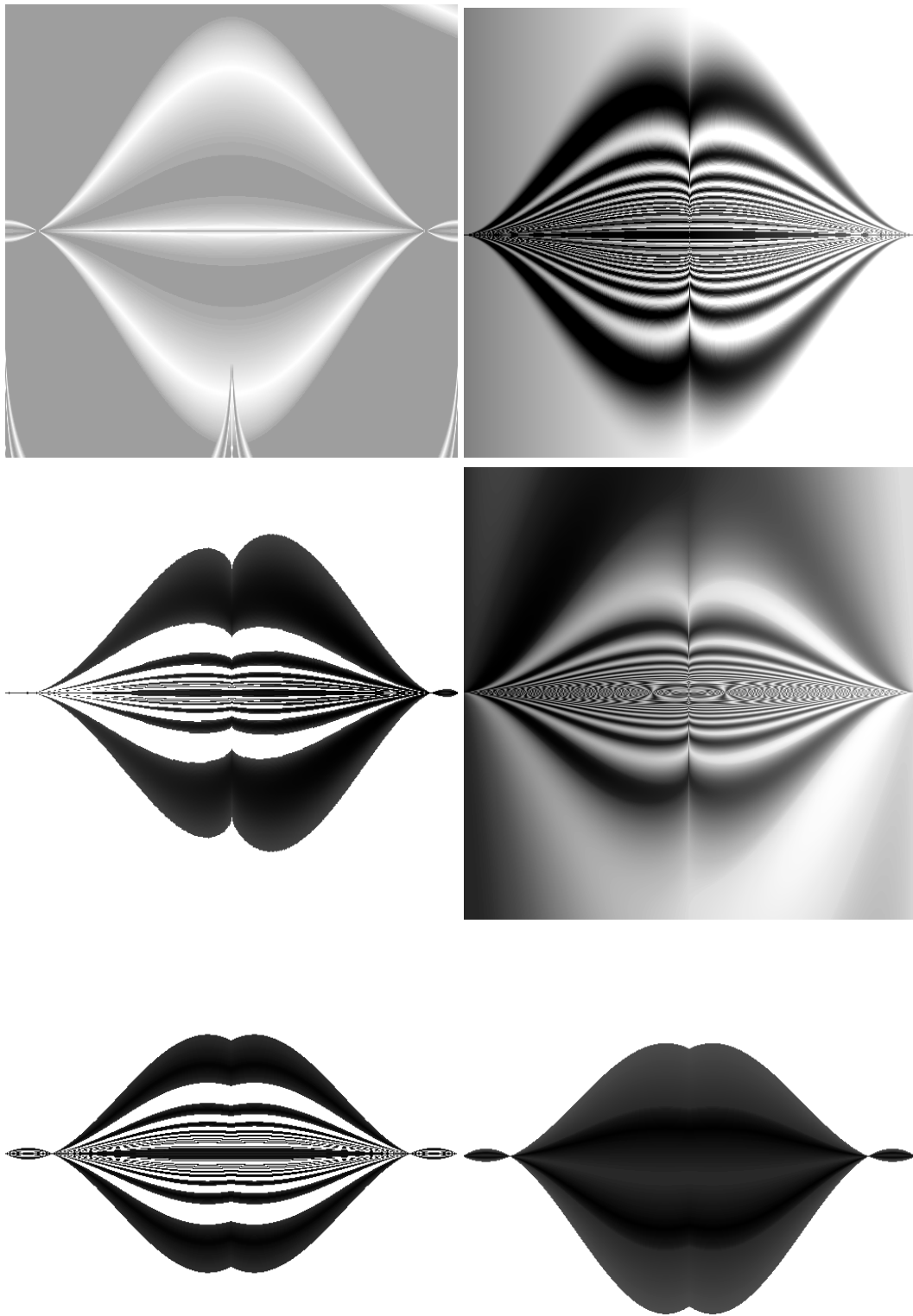


Figure 4.24: Selection of images evolved using a detector of lips to assign fitness.

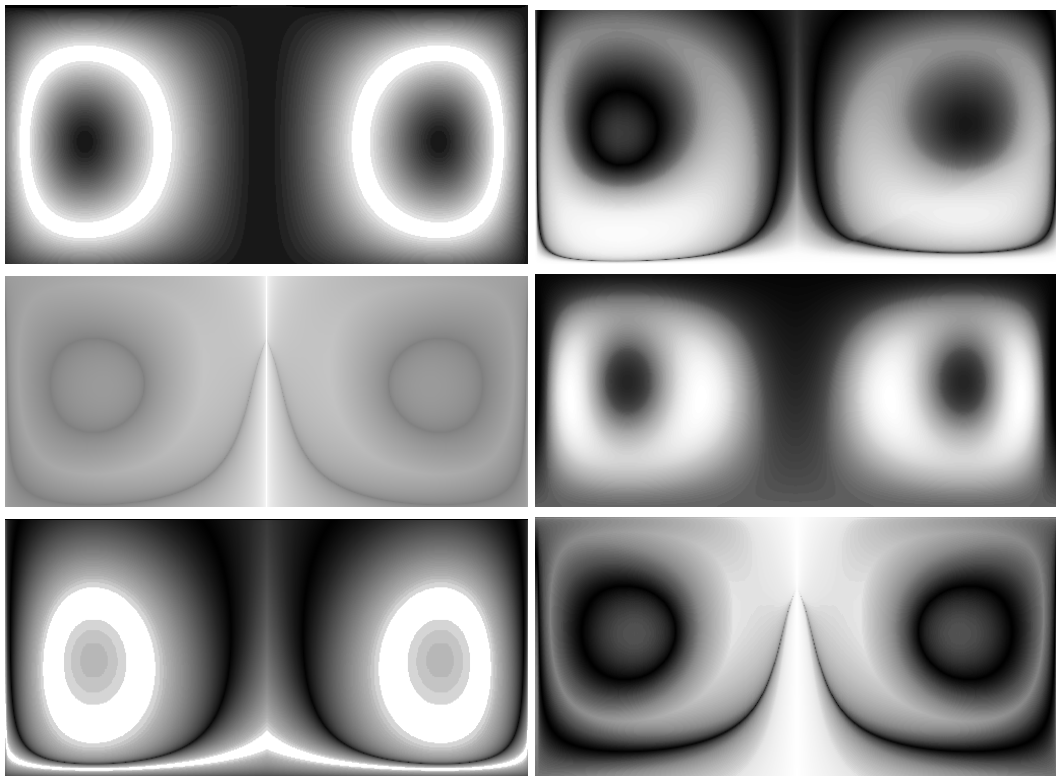


Figure 4.25: Selection of images evolved using a detector of breasts to assign fitness.

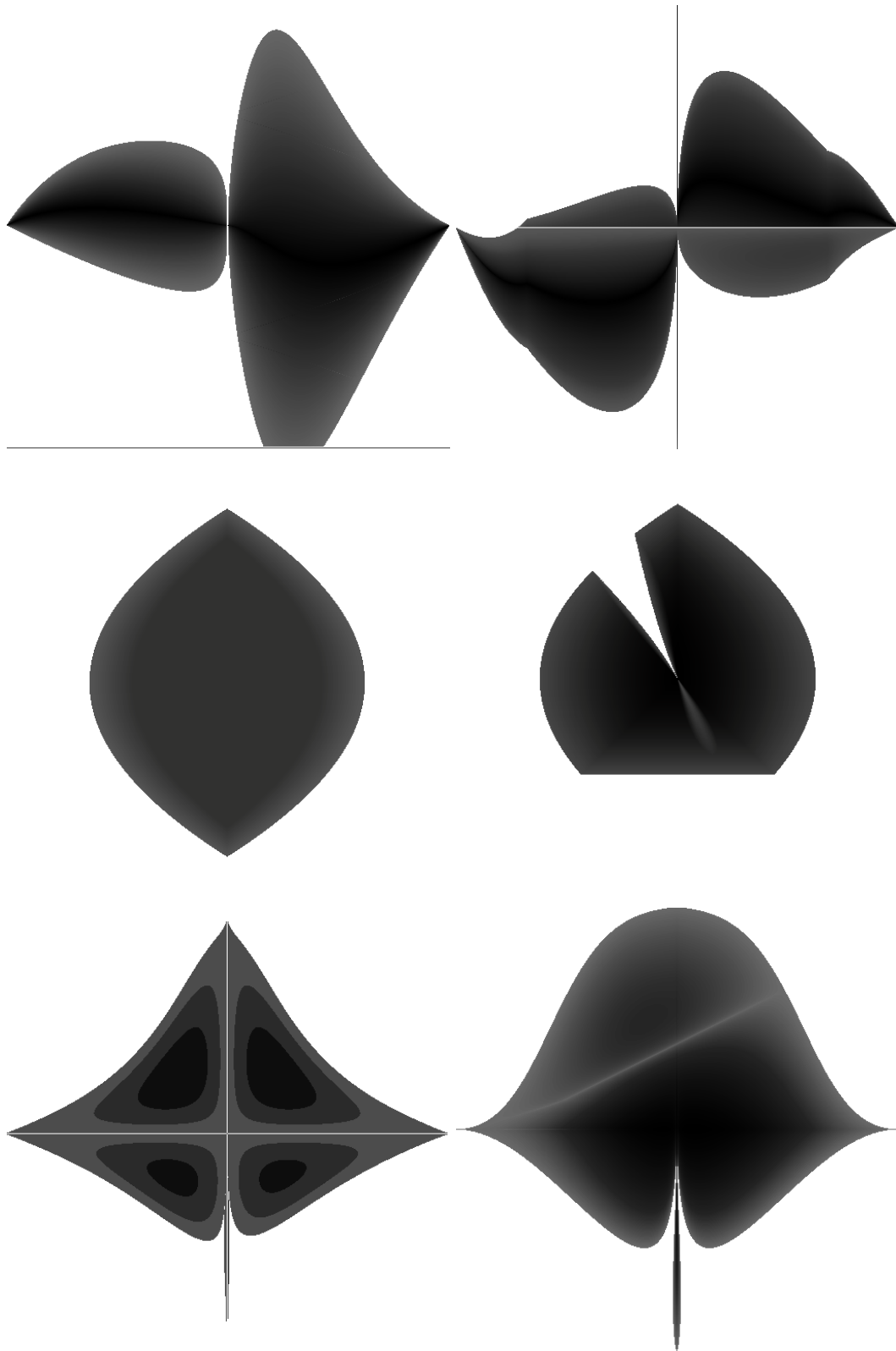


Figure 4.26: Selection of images evolved using a detector of leaves to assign fitness.

effect of the approach, since the evolution of caricatures and logos fits our intention to further explore these images from an artistic and design perspective.

The results obtained using the off-the-shelf classifier tend to be worse than those obtained with the ones we trained. An explanation for this fact follows. When one builds a leaf detector, for instance, one is typically interested in building one that recognises leaves of all types, sizes, colours, in different lighting conditions, against clear and cluttered backgrounds, etc. Although the inclusion of all these examples may lead to a classifier that is able to detect all leaves present in an image, it also means that this classifier will be prone to recognise leaves even when only relatively few features are present. In contrast, when we built our classifiers we selected as positive examples, clear and iconic images. Our classifiers probably fail to identify a large portion of real-world images containing breasts or leaves. However, they are selective and, as such, when the runs succeed they tend to present images that are also iconic and clear.

4.3.2 *Summary*

The goal of this Section was to further explore **EFFECTIVE** to evolve different figurative images by using different object detectors. We further explore and try to address some of the challenges that were stated in this research.

Several object detectors were used to evolve images that resemble: lips, breasts and leaves. The results from 30 independent runs per each classifier have shown that it is possible to evolve images that are detected as the corresponding objects, and that also resemble that object from a human perspective. The images tend to depict an exaggeration of the key features of the associated object, which is explainable by the fitness assignment scheme and constitutes a valuable characteristic, allowing the exploration of these images in design and artistic contexts. This will probably imply the further refinement of the evolved images by means of interactive evolution.

4.4 EVOLUTION OF AMBIGUOUS IMAGES

In the previous two sections, we described the work done evolving faces and other objects. The work described here resulted from extended collaboration with MSc Adriano Vinhas and with Dr. Aniko Ekárt which resulted in two publications (Machado et al., 2015b,c). Parts of this section come from those articles. This work builds upon the findings of Correia et al. (2013a) and Machado et al. (2012a) expanding the approach to evolve ambiguous images. In other words, our goal is to evolve images that induce multistable perception, which occurs when the brain (or the computer in our case) is confronted with a visual stimulus that can be interpreted in multiple ways. Some famous examples of ambiguous images are duck/rabbit; Rubin's vase, which can be perceived as a vase or two opposing faces; "My Wife and My Mother-in-Law", which may be interpreted as a young or an old woman (see Figure 4.27).

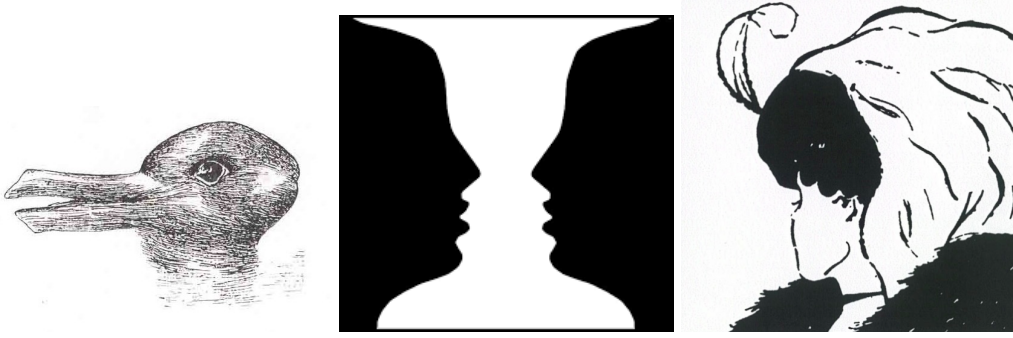


Figure 4.27: Well-known examples of ambiguous images, from left to right: duck/rabbit; Rubin's vase; "My Wife and My Mother-in-Law".

We consider ambiguous images and multistable perception fascinating phenomena, worth studying for both scientific and artistic purposes. Some of the questions that motivate the research reported here are: (i) Can ambiguous images be created by fully automated computational means? (ii) Can this be done from scratch (i. e., without resorting to collages or morphing of pre-existent images)? (iii) How do computational ambiguous images look like? (iv) How do they relate to human ambiguous images? (v) How can the dichotomy between human and computational ambiguity be explored for artistic purposes? (vi) Can one explore computer vs human creativity and perception scientifically via ambiguous images?

To evolve ambiguous images, an incremental approach is followed. First, we evolve images containing a single object. Following in the work presented in the previous sections (Correia et al., 2013a; Machado et al., 2012a) we use an object detector to guide evolution, assigning fitness based on the internal values of the object detection process. Then, using object detectors trained to identify different types of objects, we evolve images containing two distinct objects. Finally, we focus on the evolution of ambiguous images, which is achieved by evolving images containing two distinct objects in the same window of the image.

4.4.1 Experimental Setup

Table 4.4 describes the parameters used in the GP engine while Table 4.5 contains the parameters of the object detection algorithm.

Regarding the general object detection parameters, there are some differences from the approach of Correia et al. (2013a). The size of the individuals rendering, in pixels, for evaluation is substantially increased from 64×64 to 128×128 . This forces the evolved individuals to contain larger objects, and be more robust and less noisy. The minimum object size, was also increased to promote the appearance of objects at the centre of the

Table 4.4: Parameters of the GP engine.

Parameter	Setting
Population size	100
Number of generations	1000
Crossover probability	0.8 (per individual)
Mutation probability	0.05 (per node)
Mutation operators	sub-tree swap, sub-tree replacement, node insertion, node deletion and mutation
Initialisation method	ramped half-and-half
Initial maximum depth	5
Mutation max tree depth	3
Function set	+, −, ×, /, min, max, abs, neg, warp, sign, sqrt, pow, mdist, sin, cos, if
Terminal set	x, y, random constants

Table 4.5: Detection parameters.

Parameter	Setting
Min. window width	90
Min. window height	90
Image Width	128
Image Height	128
Scale Factor	1.1
Image pre-processing	Otsu’s Binarisation

image. Another relevant difference is that the images are all pre-processed, for training or detection, with a binarisation algorithm. We used Otsu’s binarisation algorithm (Otsu, 1979) to transform the images.

We conducted tests with and without binarisation of the output of the GP system⁸. Overall, we found that binarised images tended to be clearer to humans. This is likely to be related to the image equalisation and “normalisation” operations performed by

⁸ We are binarising the output of a floating point GP system. Using a binary GP approach would avoid this intermediate step.



Figure 4.28: Sample of the binarised flowers dataset.

the object detectors before classification, which may highlight features that are hard to see in the original images.

Three different experimental environments were prepared for this work. In the first one, we used the two classifiers – faces and flowers (samples in Figure 4.28) – individually to guide evolution. We then focused on the evolution of images that could simultaneously evoke faces and flowers. In the second experimental setting, we assigned fitness based on the results of the face and flower detectors, and we reduced the minimum window size parameter to (40×40) . This promoted the evolution of images containing both objects, but it did not require the faces and the flowers to overlap. Finally, in the third experimental setting, we used both classifiers to assign fitness and a minimum window size of (90×90) , which forced an overlap between the windows detecting both objects. For each combination of parameters, we performed 30 independent evolutionary runs using different random seeds. To promote readability, we normalised all fitness values by dividing the raw value by the maximum value found in the course of the experiments. The values used for the plots are averages of 30 runs.

4.4.2 Experimental Results

The results obtained when evolving images containing single objects confirm previous work in this field (Correia et al., 2013a; Machado et al., 2012a). In all runs and for all classifiers, evolution was able to produce images where the object was detected. In most situations, this was accomplished in fewer than 50 generations.

Figure 4.29 depicts the evolution of the fitness of the best individual when evolving flowers, as well as the percentage of those individuals where a flower was detected. As can be observed, by the 70th generation all runs had already produced individuals containing flowers. Also confirming previous results in the area, although all runs evolved images where the object in question was detected, the visibility of these objects to a human observer is questionable in some of the cases. Figure 4.30 presents some examples of the evolved images. As can be observed, while the flowers are easy to identify, seeing the faces is not obvious in all the cases. This can be explained by the

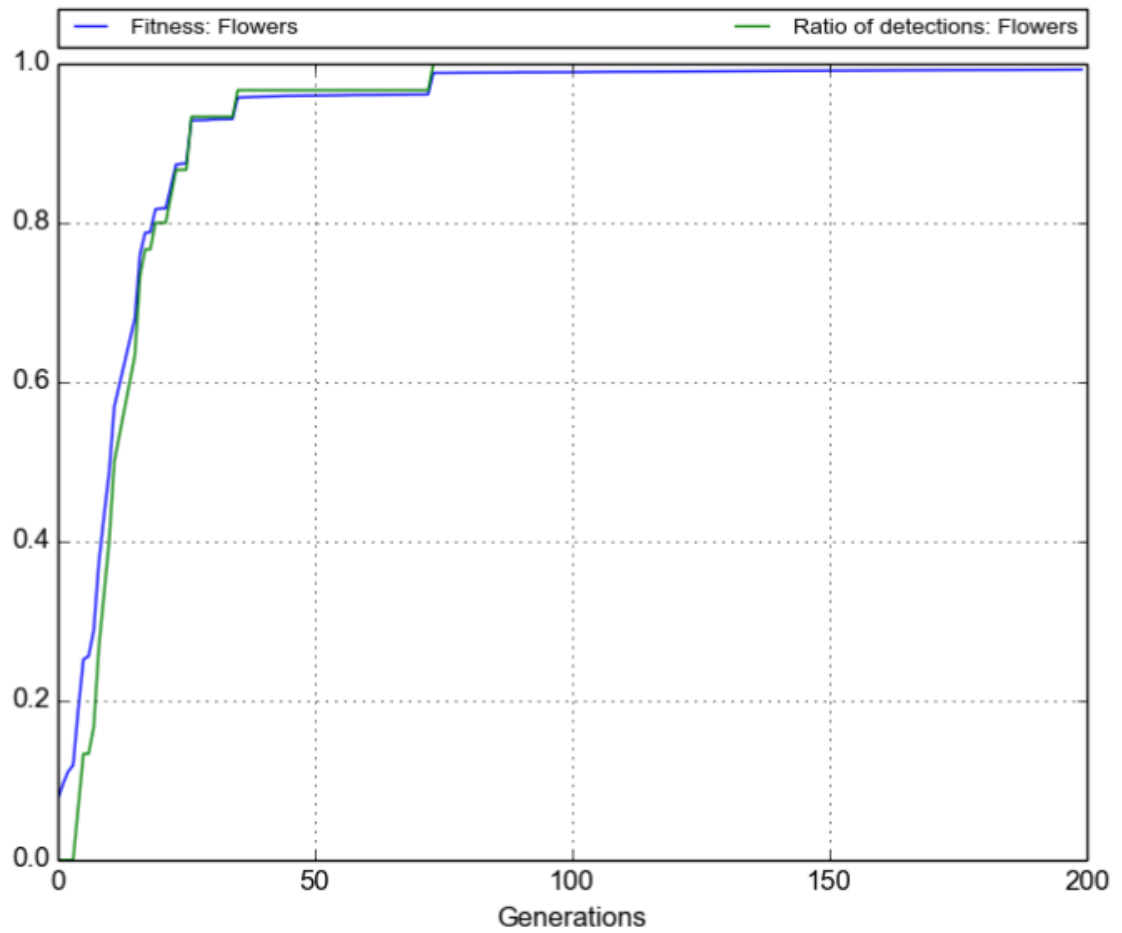


Figure 4.29: Evolution of the fitness of the best individual across generations and of the percentage of best individuals in which a flower was detected. The results are averages of 30 runs.

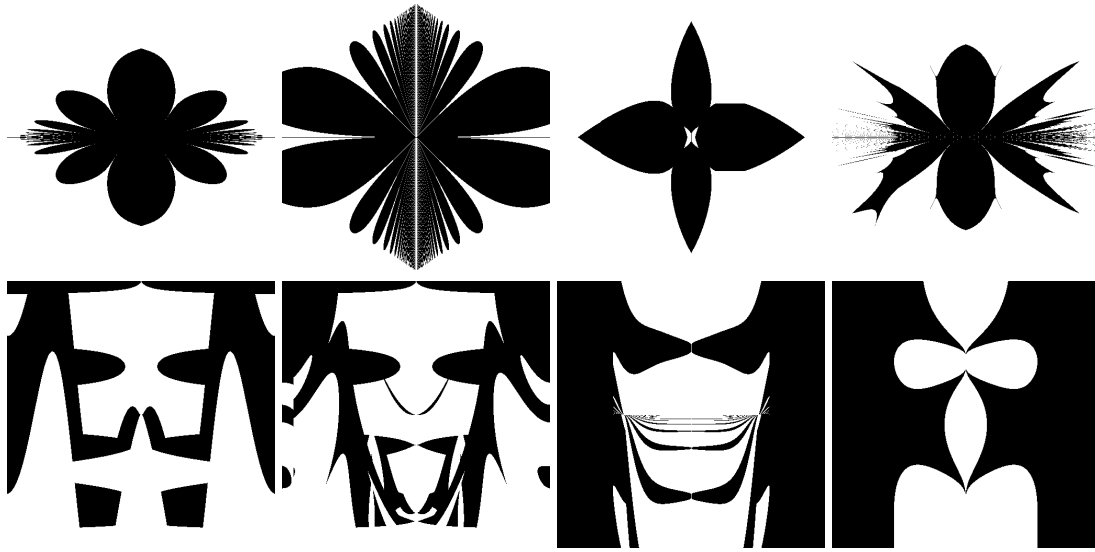


Figure 4.30: Examples of evolved images containing flowers (top row) and faces (bottom row).

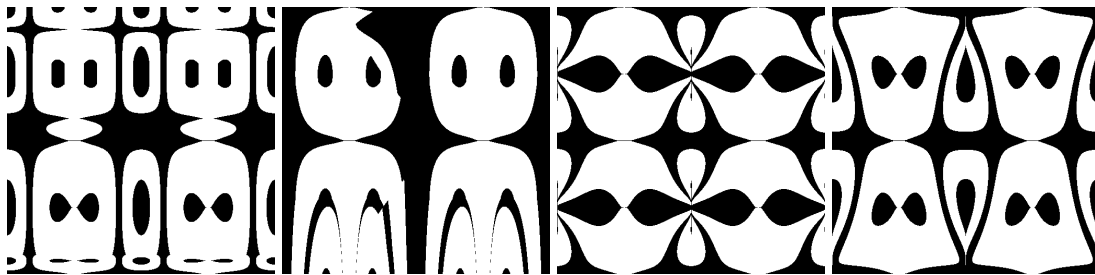


Figure 4.31: Examples of images containing non overlapping faces and flowers.

fact that the detection of flowers relies heavily on the contour of the shape, while the detection of faces relies on the presence of a combination of features that can be identified as eyes, eyebrows, lips, nose, chin, face contour,⁹ which may be obfuscated by other image artifacts.

We then focused on the evolution of images containing faces and flowers simultaneously, without enforcing the overlap between the regions where these objects were identified. Figure 4.31 depicts examples of the results obtained in this setting. In all of the examples presented, the system was able to evolve images where the object detectors found faces and flowers. Interestingly, some of the evolved images (e. g., the ones presented in the bottom row of Figure 4.31) depict the same type of optical illusion as Rubin's vase (see Figure 4.27). In this case, although we do not promote

⁹ Simultaneous presence of all of these features is not necessary.

the overlap between the detection windows and although these could be completely non-overlapping, the solutions found by the EC engine often take advantage of the similarities between visual features of the objects. This is particularly evident in the bottom leftmost image of Figure 4.31 where the eyes of the faces serve as petals for the flowers, and vice-versa. As such, we can state that in some of the evolutionary runs the algorithm evolved images that are ambiguous both from a computational and human perspective, in the sense that both computer and human can recognise a face and a flower in the same region simultaneously. As a side-note, it is also interesting to note that some of these images constitute tiling patterns, which is an unexpected outcome.

In our third experimental setting the overlap between the regions where faces and flowers are detected becomes a requirement. Figure 4.32 shows the evolution of the fitness of the best individual. In addition to the combined fitness value, we also present the fitness scores according to each classifier. As previously, these results have been normalised by dividing by the highest corresponding value found in the course of all the experiments. The percentage of the best individuals where faces and flowers were simultaneously detected in overlapping regions is also depicted, as well as the percentage of the best individuals where faces and flowers were detected. All results are averages of 30 runs.

4.4.2.1 *Visualising the Evolved Instances*

As can be observed, although there is an abrupt increase of fitness during the first generations, improving fitness beyond that point is extremely difficult. Moreover, maximising the response of the face detector is harder than maximising the response of the flower detector. This outcome was expected since the same behaviour was observed when evolving images containing a single object. In 76.6% of the runs, the algorithm was able to evolve images where overlapping faces and flowers were detected. However, when we compare the fitness values obtained by each of the two object detectors with those obtained when evolving single objects, we arrive at the conclusion that the components of the combined fitness are far from their maximum values. This can be observed by contrasting the value reached by the fitness component regarding flowers of Figure 4.32, with the value attained when evolving flowers only, which is depicted in Figure 4.29. Therefore, although overlapping faces and flowers were detected in 76.6% of the runs, the difficulties found in maximising the individual fitness components indicate that these detections are probably not robust.

An analysis of the resulting images reveals that although the majority of the runs evolved images where both objects were detected in the same window, which can, as such, be considered computationally ambiguous, most of the images found are not evocative of both objects (see Figure 4.33). Thus, in most cases they do not induce a multistable interpretation. Nevertheless, in some cases, images that are also ambiguous from a human perspective were evolved. Figure 4.34 depicts some of these exceptions to the norm. For instance, looking at the leftmost image of Figure 4.34 we can

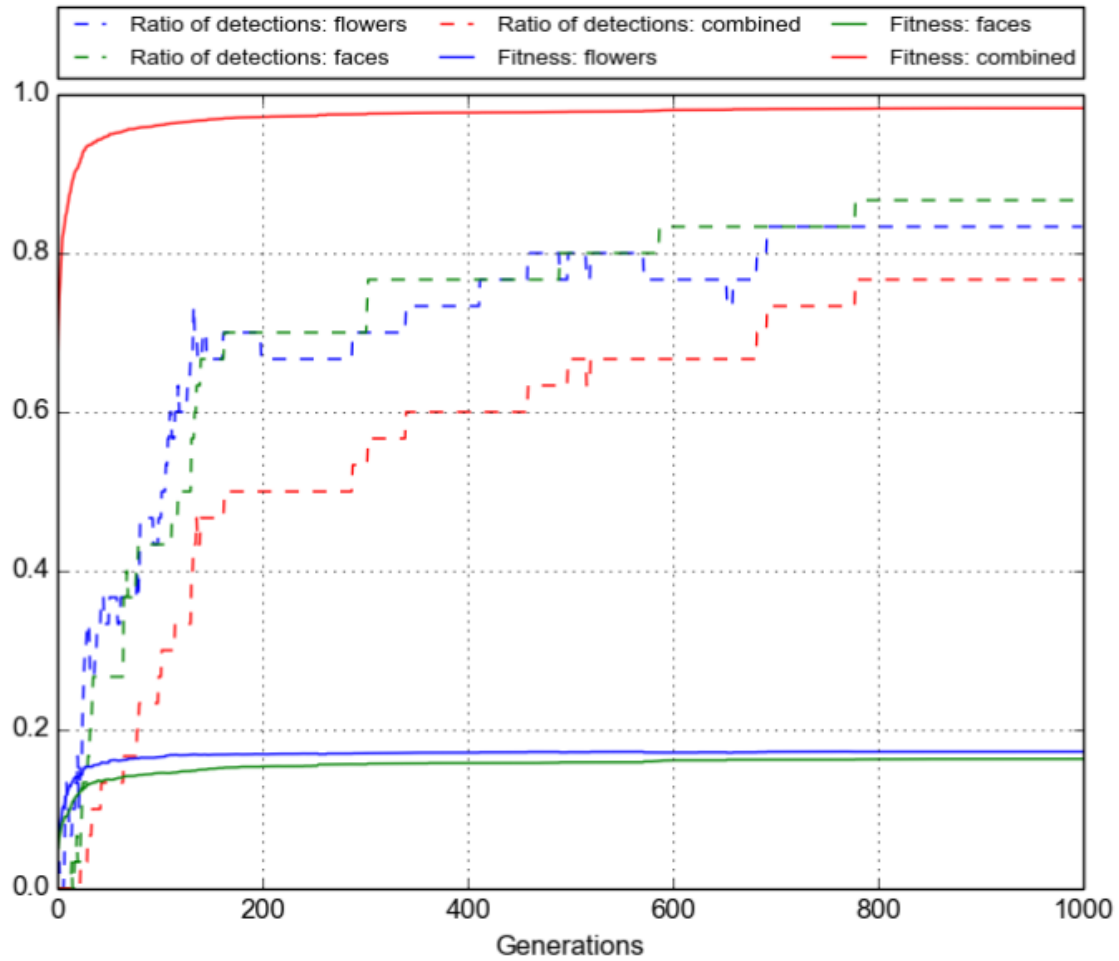


Figure 4.32: Fitness of the best individual and percentage of the best individuals containing an overlap between a face and a flower. In addition to the overall fitness and detection ratios, the partial fitness and the ratios of each of the detectors is also presented. The results are averages of 30 runs.

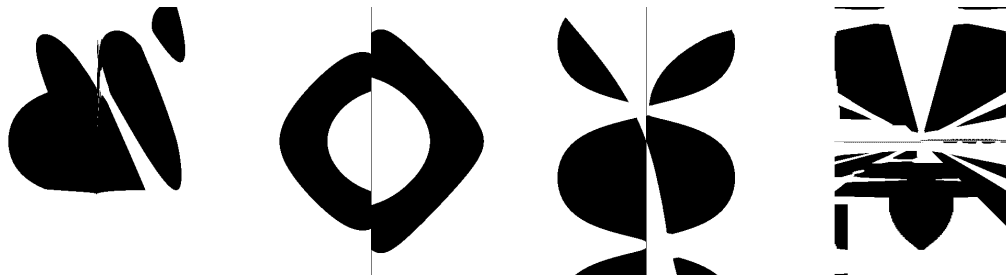


Figure 4.33: Evolved images that are computationally ambiguous, but fail to induce, in our opinion, multistable interpretation in humans.

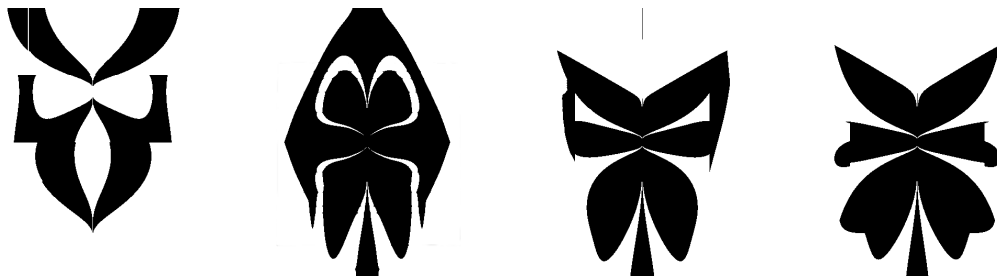


Figure 4.34: Evolved images considered ambiguous by both humans and computers.

identify eyebrows, the white oval shapes create the illusion of eyes, while the remaining symmetrical black shapes create the illusion of a face contour. Simultaneously, the white regions can be interpreted as petals of flowers. Looking at the rightmost image, one can recognise a flowery pattern, but one can also interpret the top two “petals” as eyebrows and the middle shapes as eyes, which immediately evoke a face, and then one will probably interpret the bottom petals as a beard or moustache.

Humans have evolved to quickly recognise faces, which is simultaneously advantageous and problematic in this context. On the one hand, our ability to recognise faces even when only a subset of the features is present makes the task more feasible. On the other hand, the same ability makes the analysis of the experimental results more subjective. The shared left-right symmetry of faces and flowers also plays an important role in the evolution of ambiguous images. We are currently conducting experiments using other objects, some of which are not symmetric (e. g., profile faces). Although we can evolve computational ambiguous images, it is hard for humans to see both objects, particularly the non-symmetric one. Our tentative explanation is that since detection of symmetry plays an important role in human image perception, humans tend to be drawn towards the symmetric object overlooking the non-symmetric one.

4.4.3 Summary

In this section, we explored the generation of ambiguous images by evolutionary means. We used `NORBERT` and several object detectors. The fitness is assigned by utilising values from the detection phase. The experimental results highlight the differences between human and computational ambiguous images.

At first, several object detectors were used to assign fitness and evolve images that resemble faces and flowers. The results from 30 runs per classifier showed that it is possible to evolve images that are detected and resemble, from a human perspective, the object. Next, we focused on the combination of flower and face detectors and evolved images that contained both objects. The results showed the ability of the system to evolve images where both object detectors found their respective object. Some of the evolved images depicted optical illusions, with shared visual features and tiling patterns. In the final experiment, the object detectors were parameterised to detect larger objects, forcing the overlapping of the objects in the evolved images. In several runs, the system was able to evolve images where the two objects were detected by the respective object detectors.

Although the evolution of computational ambiguous images was frequent, only a portion of these images is evocative of both objects to humans. These evolved images can be considered ambiguous to humans, capable of inducing multistable perception. Furthermore, although the results obtained so far are not of the same level as human-designed ambiguous images, we consider them inspiring. They also demonstrate the feasibility of the approach and open new avenues for research.

As previously mentioned, the experimental results contain many false positives. In Chapter 5 we are going to explore the ability of the `EC` engine to find false positives to improve the quality of the training sets, and hence the robustness of the object detectors.

4.5 EVOLUTION OF PHOTOREALISTIC FACES

Sections 4.3 and 4.4 described instantiations of `EFFECTIVE` to generate particular images that contained a certain object by defining and annotating the positive examples of the dataset. From Chapter 2 we know that it can be hard to gather and annotate many instances. In this chapter, we are going to describe an instantiation that helps to deal with this issue. Parts of this section are based on the published article Correia et al. (2016).

As the name of the Section indicates, in this Section we describe the work done to generate photorealistic faces. In the first application of `EFFECTIVE` (Machado et al., 2012b) analysed in Section 4.2, we were able to generate faces using a general purpose evolutionary art tool as the `EC` engine. With some degree of subjectivity, the approach showed promising results, delivering results that were evocative of human frontal

faces, e. g., caricatures, masks and abstract smiles as faces. Nevertheless, most of the images generated were images that from a subjective standpoint were not evocative but were classified as faces by the classifier. Therefore we created a system able to generate false positives. In this work, we have two objectives: (i) design an **EC** engine capable of generating photorealistic frontal faces consistently; and (ii) analyse if we can evolve images that from a subjective perspective are classified as faces but the classifier does not recognise as faces.

Thus, we propose **DA** approach to autonomously generate new frontal faces out of existing ones. The elementary parts of the faces are recombined using **EC** and Computer Vision (**CV**) algorithms. Most parts of the text that is described here are from the resulting publication (Correia et al., 2016).

4.5.1 Annotation Tool

We have developed an image annotation tool (see Figure 4.35). It allows the user to annotate objects present in images. One can annotate an object by positioning a sequence of points along its contour and by choosing the corresponding category. New categories can be added at any moment. The annotations created by the user are automatically saved in output files, particularly in one eXtensible Markup Language (**XML**) file for each image and in one text file for each object category. The tool also exports the mask of each annotated object. When one opens a folder with images, the tool loads the corresponding annotations saved in files if they exist. The interaction and features provided by the tool are depicted in the demo at <https://cdv.dei.uc.pt/2016/annotation-tool.mov>.

We have used this tool to annotate the elementary parts of faces on a set of images. In this work, each face is annotated by indicating the bounds of its eyes, eyebrows, nose, mouth, as well as the bounds of the face itself.

4.5.2 Evolutionary Engine and Fitness Assignment

The **EC** engine was described in Section 4.1.2.2, a conventional **GA** that is used to evolve sets of indexes of annotated images and image parts, which form a composite face. In Figure 4.36 we have examples of these composite faces. In some we can see what parts we swapped, but imagine if they had been mixed up or if we did not give any indication on which images were the originals or the composites, it would be hard to track or notice it. This is one trait that we want to enforce with this method: we want the generated individuals to look like real faces and different from the originals. Of course in some cases (third row of Figure 4.36), the composite face does not look realistic, but nevertheless, it still holds the features that make it a recognisable face. As stated in Section 3.3, designing a **EC** engine that is able to easily generate the target

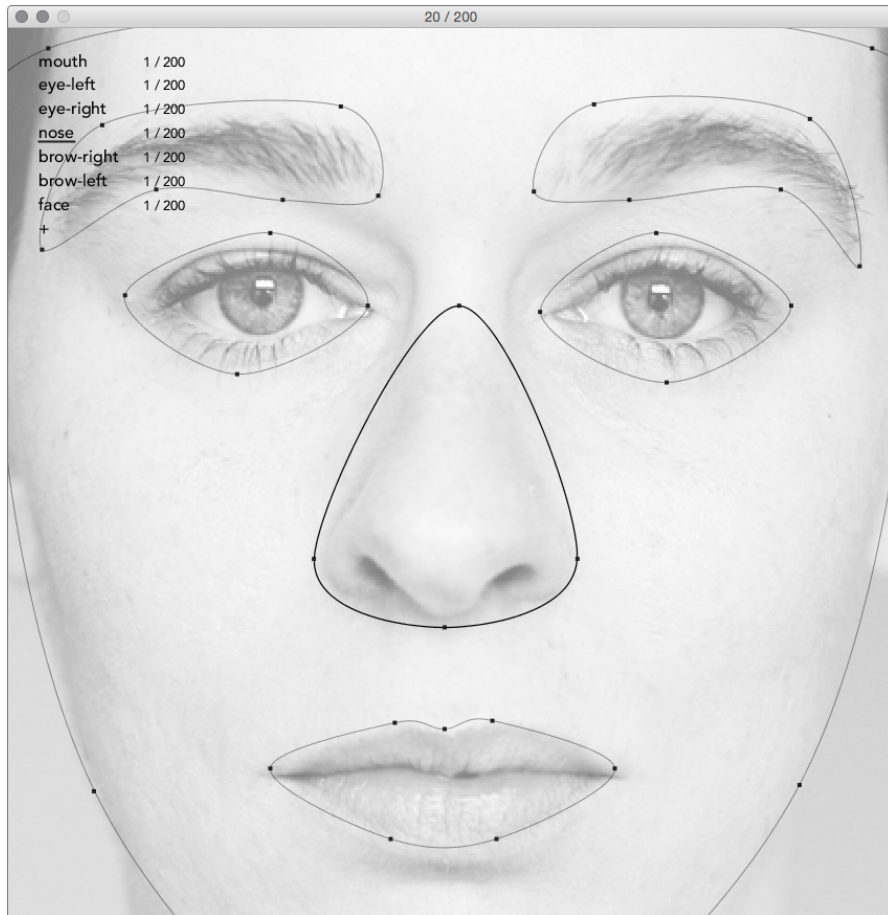


Figure 4.35: Screenshot of the annotation tool.

class is important for the success of the framework. This representation ensures that we are always generating recognisable photorealistic faces.

Similarly to the previous instantiations of [EFFECTIVE](#) to create a fitness function it is necessary to convert the binary output of the face classifier(s) to an output that can provide a suitable fitness landscape. As in previous works we do this by accessing internal values of the classification task that give an indication of the degree of certainty in the classification. In this case, we are interested in a fitness function that penalises individuals that are classified as faces. Thus, the fitness function is defined as:

$$f(x) = (tstg - pstg(x)) + (tstg * ndet(x)) + \frac{1}{1 + stgdif(x)}, \quad (4.2)$$

where *tstg*, is the total number of stages of the classifier, *pstg* is the number of stages that the input image has passed, *ndet* is a boolean variable that tracks if the image is considered a face, *stgdif* is the difference between the value attained in the last stage that the image passed and the threshold of the stage. Of course, this is one of many possibilities.

4.5.3 Experimental Setup

Since we are interested in evolving faces from existing ones, as a [DA](#) approach, our objective is to evolve instances that are misclassified as negative examples, i. e., not classified as faces. We begin by defining two datasets of positive examples, one with 200 examples and the other with 500 examples. We then use these datasets to train two classifiers: *face200* and *face500*. The *face200* dataset includes all 200 annotated examples that are used in the GA for recombination. The *face500* dataset contains all the *face200* examples plus 300 more examples. With these two datasets, we intend to explore the impact of our approach in a scenario where all available faces have their parts annotated and in a scenario that only a fraction of the available instances are annotated. Furthermore, we are interested in analysing the resulting individuals in both scenarios.

The positive examples that compose both datasets were extracted from the FACITY project, a worldwide project that gathers the pictures of photographers capturing the multiplicity of human faces from different cities and countries¹⁰. The examples used for training the *face200* and *face500* classifiers are shown in Figures [4.37](#) and [4.38](#), respectively. We sought to obtain a heterogeneous group of faces.

We maintained the same negative examples used from previous experiments (e. g., Section [4.2](#)), composed of 1905 images from the “Urtho – Negative face Dataset”, which contains images of landscapes, objects, drawings, computer generated images, among

¹⁰ FACITY project – <http://www.facity.com/>



Figure 4.36: Examples of composite faces, one per line. On the left, we have the two original images, and on the right, we have the two composites.



Figure 4.37: The positive instances used to train the classifier *face200*.

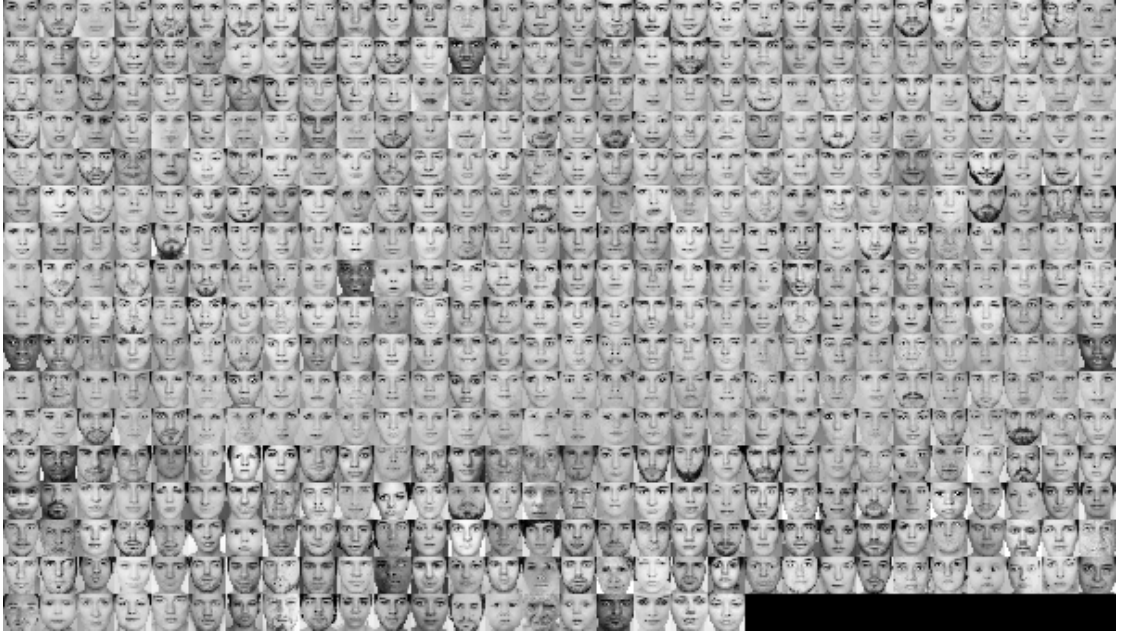


Figure 4.38: The positive instances used to train the classifier *face500*.

others. In the scope of this work, we intend to study the results of our approach while using the classifiers *face200* and *face500* to assign fitness.

The main classifier parameters can be consulted in table 5.1 and were chosen based on the works of Viola and Jones (2001) and Lienhart et al. (2002). As for the face detection settings we use the default parameters of OpenCV, which are presented in table 5.2.

We test two experimental setups: *setup200* and *setup500*. In the first one, the *faces200* is used to guide evolution and the *faces500* to curate individuals; In the second, the *faces500* is used to guide evolution and the *faces200* to curate individuals. The role of the curator is to evaluate the individuals generated by the guiding classifier and to select faces that it does not classify as faces. We study the behaviour of each curator and its selection of individuals.

Table 4.6: Training parameters.

Parameter	Setting
Example width	64
Example height	64
Number of stages	20
Min. hit rate per stage	0.999
Max. false alarm per stage	0.5
Adaboost algorithm	GentleAdaboost

Table 4.7: Detection parameters.

Parameter	Setting
Scale factor	1.2
Min. face width	$0.7 \times \text{example width}$
Min. face height	$0.7 \times \text{example height}$

The evolutionary engine settings are presented in Table 7.4. Concerning face parts recombination, we maintain the pairs of eyes and the pairs of eyebrows, reducing the genotype length from seven to five. The rationale for this decision is related to the fact that most faces have a certain horizontal symmetry that the classifier tends to learn from the positive examples. On early experiments, we have observed that the classifiers struggled on with images where the pair of eyes and eyebrows belonged to different faces, leading to an early convergence of the system. Besides the technical aspects, the images evolved were unnatural and easily noticeable as blends.

4.5.4 Experimental Results

In this section, we present and analyse the experimental results. We begin by analysing the evolution of fitness in the two experimental setups. Afterwards, we discuss the impact of the fitness on the progression of detections over the generations. We present and discuss the individuals selected by the curators and the best individuals generated by the guiding classifier. Finally, we analyse the visuals of the evolved individuals.

Figure 4.39 shows the evolution of fitness of the best individuals along the generations in the *setup200* and *setup500* experimental setups. We plot both fitness curves to examine how one affects the other.

We can observe that the EA can optimise the fitness function. In both setups, when one fitness value increases the other tends to have a similar behaviour. The values reveal that it is easier to satisfy the *face200* in both setups, i.e., when it is evaluating and when it is curating. The observed behaviour in *setup200* suggests that the evolved individuals that affect the *face500* performance also affect the *face200*. This can be a

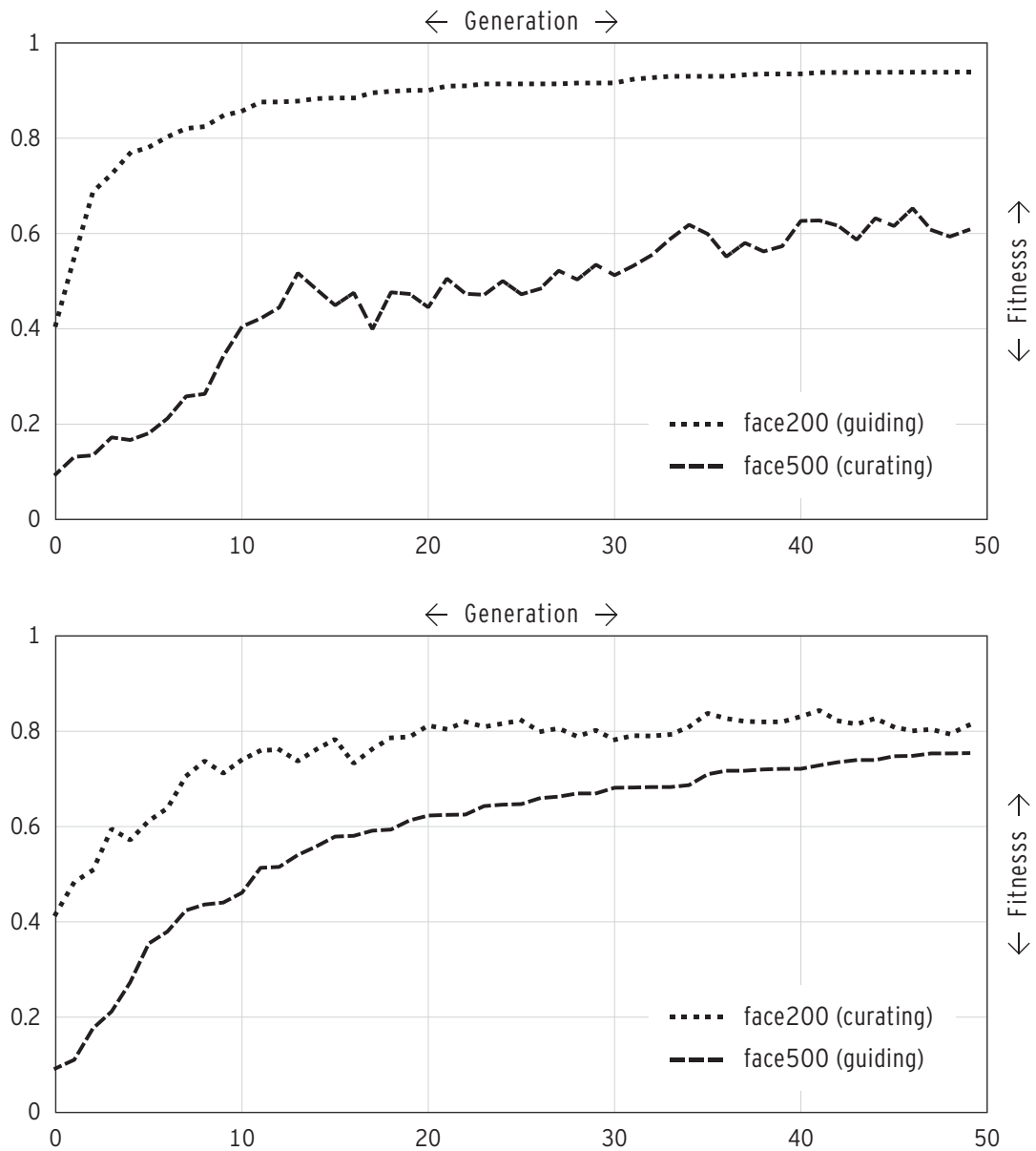


Figure 4.39: Evolution of the fitness of the best individual across generation when using *face200* to guide evolution and *face500* to curate individuals (top) and the other way around (bottom). The visualised results are averages of 30 runs.

Table 4.8: EC engine parameters.

Parameter	Setting
Number of generations	50
Population size	50
Elite size	1
Tournament size	2
Crossover operator	uniform crossover
Crossover rate	0.8
Mutation operator	gene replacement
Mutation rate per gene	0.15

consequence of the *face200* training instances being included in the *face500* training, which makes the evolutionary process evolve individuals that are different from the ones available in both datasets.

In Figure 4.40 we observe the average of individuals that are classified as faces throughout the generations. The number of detections decreases in both setups, showing the ability of the approach to evolve individuals that are not classified as faces, i. e., classified as false negatives. A contrast between the two charts is observable. When *face200* is guiding, *face500* maintains a high rate of detections. When *face500* is guiding and *face200* is curating, both curves behave similarly. Based on the percentage of faces detected in *setup200*, the evolution promotes solutions that are classified as faces by *face500*. This is somewhat consistent with the fitness curve behaviour of Figure 4.39.

Figure 4.41 depicts in which generation the best individual, on average, seizes to be classified as a face. One can conclude that when *face200* is guiding, in less of 10 generations the best individual is not classified as a face. The behaviour of the curator (*face500*) suggests that the best individuals are still detected in the last generations. In contrast, when *face500* is guiding the results indicate that there are evolutionary runs where the best individual is still classified as a face. Furthermore, these individuals are also classified as faces by *face200*.

Figures 4.42 and 4.44 depict a selection of fittest individuals registered in the experiments. As for figures 4.43 and 4.45 one can observe some of the curated individuals. The results suggest that although there are individuals in common, the two curators tend to select different individuals. Some of the selected faces share characteristics that we consider as exploits of the classifiers, particularly at the level of the skin tone, contrasts, and size of some facial features. One can conclude that there are overlaps between the fittest and the curated individuals in *setup200* (see figures 4.42 and 4.43). When *face500* is curating or guiding, the evolved individuals share more characteristics (see figures 4.44 and 4.45). This is consistent with the idea that the exploits of *face500* tend to be also exploits of *face200*.

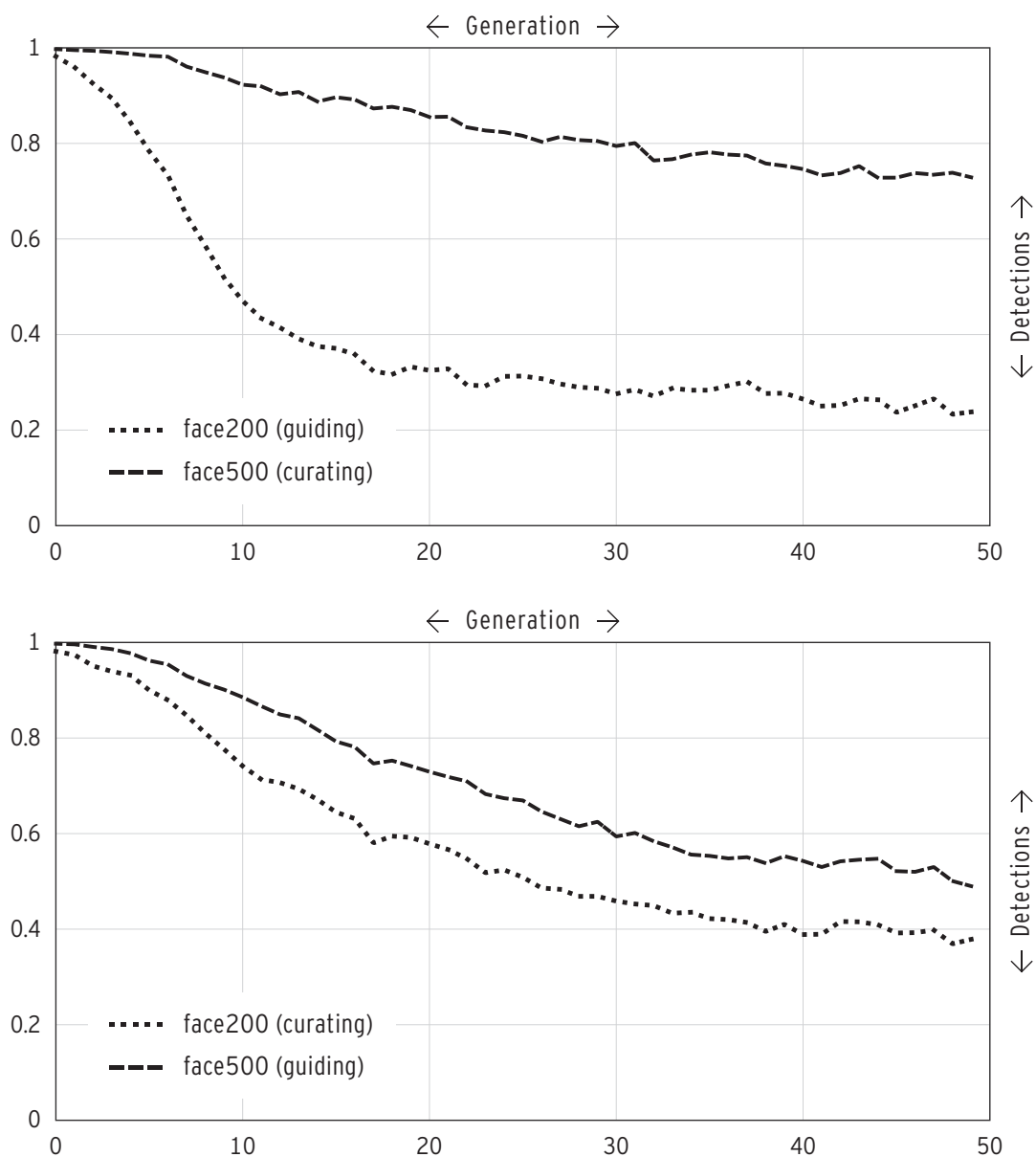


Figure 4.40: Progression of the average of detections when using *face200* to guide evolution and *face500* to curate individuals (top) and the other way around (bottom). The visualised results are averages of 30 runs.

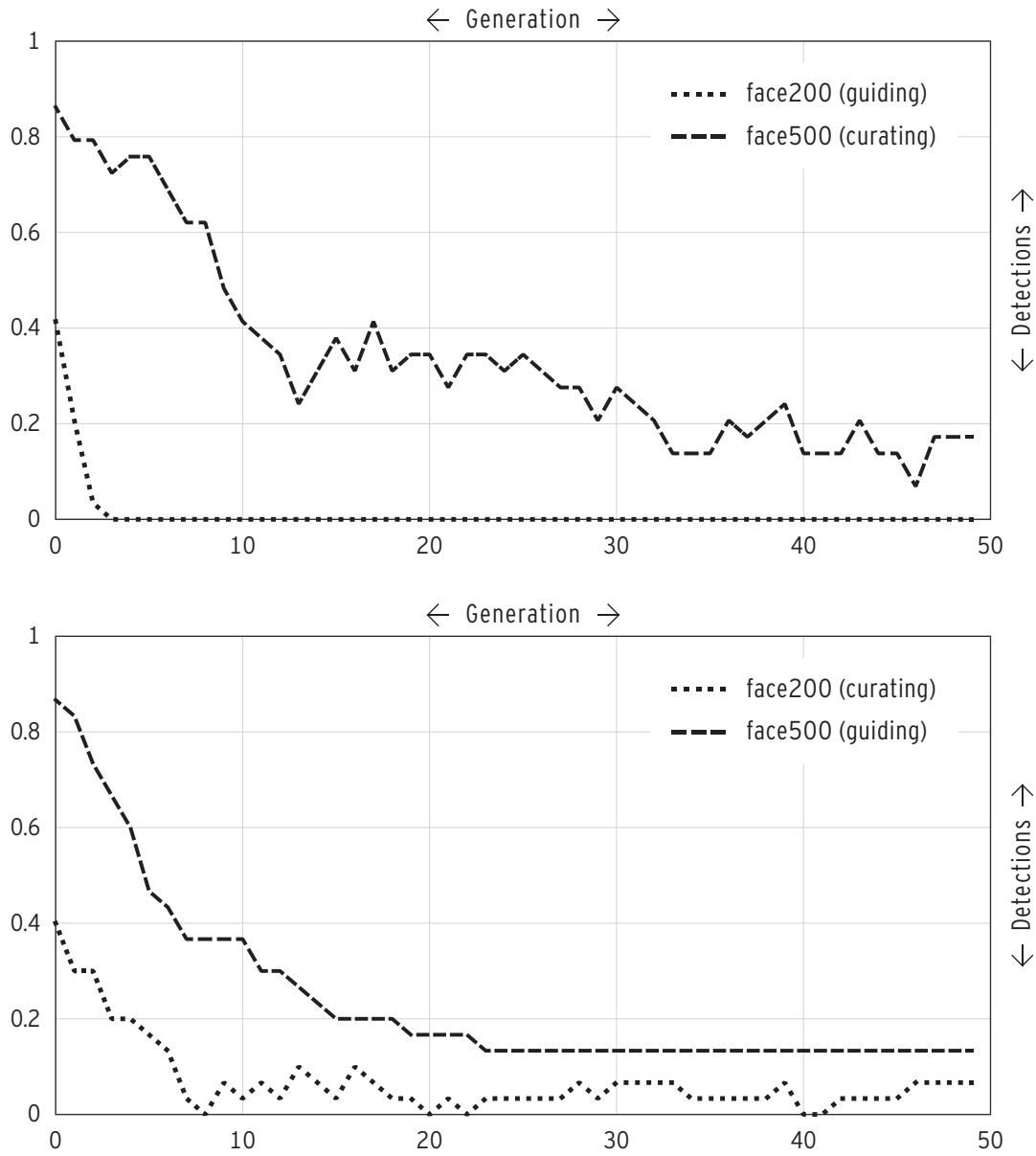


Figure 4.41: Progression of the detection of the best individual across generations when using *face200* to guide evolution and *face500* to curate individuals (top) and the other way around (bottom). The visualised results are averages of 30 runs.

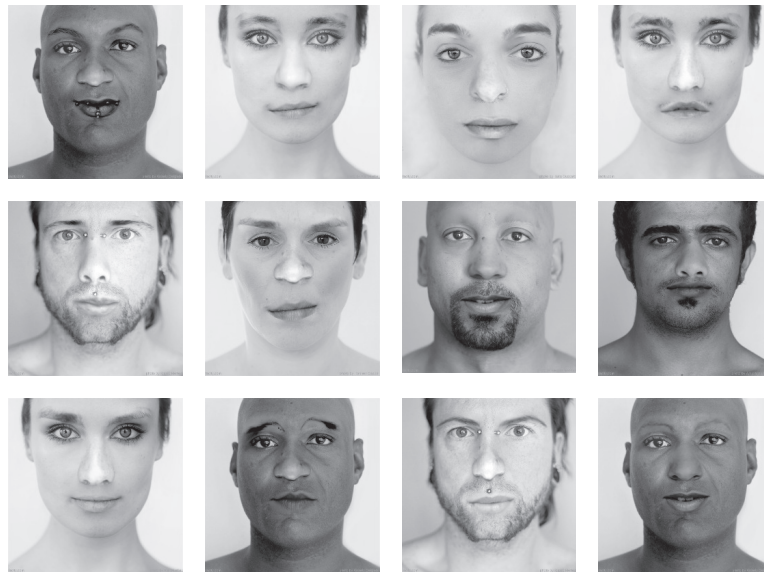


Figure 4.42: Fittest individual in the last generation for 12 different runs when using *face200* to guide evolution.



Figure 4.43: Examples of faces curated by *face500* in different runs when using *face200* to guide evolution.



Figure 4.44: Fittest individual in the last generation for 12 different runs when using *face500* to guide evolution.



Figure 4.45: Examples of faces curated by *face200* in different runs when using *face500* to guide evolution.

Figure 4.46 depicts a selection of individuals evolved in different runs that we found to be interesting and peculiar. This selection shows the ability of the approach to explore the search space and exploit the vulnerabilities of the classifiers in an automatic and tractable way. As such, one could expect simple recombinations of faces that the classifier has not “seen” before or exploits of lighting and contrast conditions. Nevertheless, the system produces atypical faces with unexpected features. For instance, one can see convincing images of babies with piercings, cases of gender ambiguity, and mixtures of interracial attributes that are at least visually uncommon and peculiar. Some of the generated faces are realistic and disturbing at the same time that one could relate with the uncanny valley problem MacDorman et al. (2009), i. e., the phenomenon where computer generated figures or virtual humanoids that approach photorealistic perfection make real humans uncomfortable.

A final comment goes for the potential use of this approach for DA. Similar to bootstrapping, this approach can generate variations or completely new examples from a pre-defined sub-set.

4.5.5 Summary

We have described and tested EFFECTIVE for the automatic generation of photorealistic faces that are not classified as faces by the classifier, i. e., false negatives. The experimental results demonstrate the ability of the proposed approach to generate a wide variety of faces that test the ability of the classifiers to detect them. As such, we consider the approach proposed herein a viable solution for DA in the field of FD. The results also show the impact of using different classifiers. Besides fulfilling the main objective (generate false negatives), from our perspective, the faces created have interesting and unexpected features.

4.6 SUMMARY

In this Chapter, we analysed the viability of the EFFECTIVE framework to generate instances, i. e., act as a generator. We explored the approach in several image generation scenarios. First, we described the framework instantiation and dwelled into the CS module and EC engine details.

The first work was a frontal face generation approach which was motivated by the idea of Romero et al. (2003). By combining a general expression-based evolutionary art tool with an off-the-shelf classifier, we showed that it is possible to generate images that are evocative of human frontal faces. At the same time, we were surprised with some of the results that exposed shortcomings of the classifier. Based on these results we thought to explore two lines of research: (i) expand to different objects, not only faces; and (ii) explore the instances that we consider to be shortcomings of the classifier and attempt to overcome them.



Figure 4.46: Examples of faces evolved in different runs. More faces can be visualised at <https://db.tt/iSPaqPLU>.

Following the first line of research, we expanded the experimentation and tested with three more objects: lips, leaves and breasts. The mouth classifier was off-the-shelf, while the other two were purposely trained for the task. The results attained showed that our approach worked for all the objects, evolving images that were evocative of the target object. Once again, the images present exaggeration of key features of the objects which, due to the fitness function employed, is an expected and consistent behaviour.

Next, we explored the generation of ambiguous images by evolutionary means. After experimenting with the evolution of images that resemble a target object, we tried combining several detectors in the same image. Next, we explored the same approach but tried to overlap the detections. We were able to evolve images that depicted optical illusions, with shared visual features and tiling patterns. We observed that the computational evolution of ambiguous images was frequent, but only a portion of these images are evocative of both objects to humans. These evolved images can be considered ambiguous to humans, i. e., induce multistable perception. We conclude that the results obtained so far are not of the same level as human-designed ambiguous images. However, they are inspiring.

Afterwards, we described the exploration of the **EFFECTIVE** framework as a generator of positive instances, namely as a **IDA** approach. We describe an approach that allows the generation of photorealistic faces by combining parts of the different faces present in the dataset. We annotated several instances of a dataset and performed experiments using two classifiers trained with a different number of instances, one containing only the annotated instances and the other containing the annotated plus unannotated ones. The fitness assignment was formulated to reward images that are not classified as faces. We performed experiments alternating in using one classifier to guide fitness and the other to curate instances. The idea of the curator is to find which evolved instances are also not detected by the curator. The results show the impact of different classifiers on the evolved faces. Besides fulfilling its purpose from our perspective, the faces created have interesting and unexpected features.

TOWARDS THE IMPROVEMENT OF CLASSIFIERS' PERFORMANCE

After showing the potential of **EFFECTIVE** as a generator of images of a particular type, we now focus on another line of research. Based on the results attained during the experiments of frontal face generation we explore **FD** as a classification problem. After answering the question “how can we generate images of a particular type?” with our approach, we focus on the question “how to generate useful instances?”. By useful we mean instances that make an impact on a classifier’s training dataset, and consequently on the classifier’s performance.

We test the framework in a **FD** problem. **FD** is a real-world application and challenging problem related to areas of **CV**, **ML** and Pattern Recognition (**PR**). At the time that these experiments took place, the state of the art approaches relied mainly on feature templates and sliding window algorithms. For this problem, we continued to employ cascade classifiers based on Adaboost with features.

To create a **FD**, one needs to define a dataset with both positive and negative instances. In this particular case, the positive instances are images containing at least one face and its localisation (image window delimiting the face); and the negative instances are images that do not contain faces. For the positive dataset one can gather images from all sort of sources containing faces. However, since any image without a face is a viable candidate for the negative dataset – which implies that the space we are trying to sample is vast – collecting a representative, effective and manageable negative training dataset becomes particularly hard, complex and time-consuming, since little guidelines exist.

Based on the results attained in previous works, we test **EFFECTIVE** in this context. We applied a different pipeline for these experiments which is explained next in Section **5.1**. Before describing the set of experiments around this problem, we need to define a baseline classifier and dataset. Afterwards, we present the evaluation methodology, define the test datasets and instantiate the framework. In this Chapter, we explore the proof of concept towards the improvement of classifiers. In Section **5.3.1**, we explore the contribution of instances generated from a single **EC** run. Then, in Section **5.3.2**, we present the experiments that combined instances generated from multiple **EC** runs, the first application of the Supervisor module.

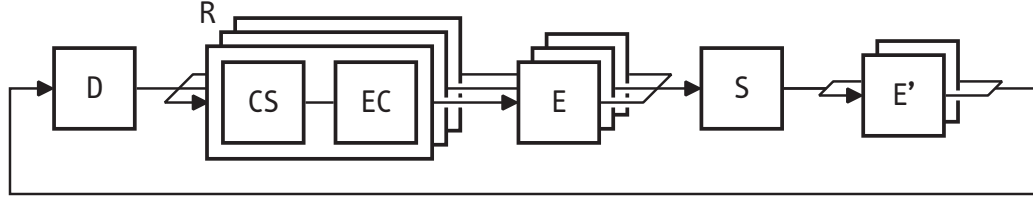


Figure 5.1: Overview of the framework workflow with multiple **CS**, each with multiple **EC** runs that yield different instances E . The instances E are processed by the Supervisor S generating a subset E' that is added to the dataset D

5.1 INSTANTIATION

The framework process employed in the next sections of this chapter is schematically summed up in Figure 5.1. A classifier is selected to represent the **CS** module. Several **CS** modules are trained for each **CS**, and several **EC** runs are deployed and the results are analysed. Since we are going to retrain the classifier, the Supervisor (S) selects and filters the results from the **EC** runs (E). This process can be repeated for several iterations. With this workflow we aim to expand the dataset (D) in every framework iteration. After every iteration, the **CS**s have their dataset augmented by the instances selected and filtered by the Supervisor.

5.2 FACE DETECTOR TRAINING AND EVALUATION

Most of the face detection process and training of a cascade classifier was already explained in Section 4.1.1. In the next sections, we define and explain the parameters, the baseline dataset and the evaluation of a face detector.

5.2.0.1 Classifier Training and Dataset

To train the classifier, we use the “opencv_traincascade” tool of OpenCV. It is beyond the scope of the thesis to fine tune the parameters of the **CS**. Therefore, we opted to use the default parameters of OpenCV for the **FD** phase and the ones that were tested in the work of Lienhart et al. (2003) for the training phase. The relevant classifier parameters can be consulted in Table 5.1 and were chosen based on the works of Lienhart et al. (2003), Lienhart et al. (2002), and Viola and Jones (2001). The parameters represent a compromise between performance and training time.

In addition to the training parameters, in Table 5.2 we define the relevant parameters for the detection phase, which were used in the **EC** runs. The scale factor and pre-processing are based on the work of Lienhart et al. (2002). Regarding the minimum face width and height, we defined that the face should occupy at least 75% of the image.

Table 5.1: Haar training parameters.

Parameter	Setting
Features	ALL
Input width	20
Input height	20
Number of stages	14
Min Hit rate per stage	0.999
Max False Alarm per stage	0.5
Adaboost Algorithm	GentleAdaboost

Table 5.2: EC run detection parameters.

Parameter	Setting
Scale factor	1.1
Min. face width	$0.75 \times \text{inputwidth}$
Min. face height	$0.75 \times \text{inputheight}$
Pre-processing	Histogram equalisation

This aspect promotes the evolution of images that have a clear pattern identified as a face (Machado et al., 2012a).

In the context of this thesis, we consider the positive dataset to be a set of instances where each instance is composed of an image that contain at least one face. On the other hand, the negative dataset is a set of instances where each instance is composed of an image that do not contain a face. For this experiment images from two well-known datasets were used: “The Yale Face Database B” (Georghiades et al., 2001) and “BioID Face Database” (Jesorsky et al., 2001). “The Yale Face Database B” is a dataset with a total 5850 grayscale images with the subjects in diverse positions and light variations. The Bio-ID Face Database dataset has 1521 frontal grayscale images. Each image shows the frontal view of a face of one out of 23 different test persons with various expressions.

We wanted to test if the proposed framework contributes to improvements in the classifier’s performance. Adding different poses has no interest in this context and would make development and analysis harder. As such, we decided to focus exclusively on frontal faces. Although it is easier to develop a good initial classifier, it is likely to make improvements harder, since there is less room for improvement.

Considering this constraint, the total number of available positive instances is 2172. To build the ground truth file, the images have to be manually selected and cropped. These cropped images (see Figure 5.2), are the objects that the Haar classifier attempts to discriminate from negative samples. After manually filtering out images that were too dark, or where only part of the face was illuminated, a total of 1905 positive

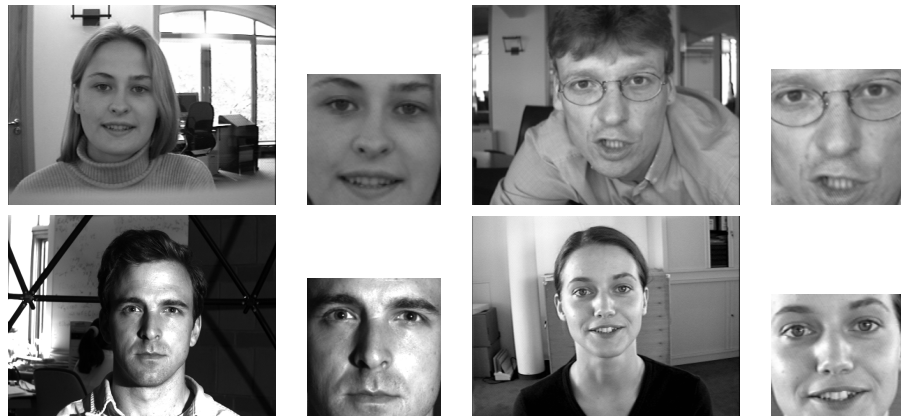


Figure 5.2: Sample of images of the positive training dataset with their corresponding cropped image corresponding to the object to detect.

instances and corresponding cropped versions were obtained. All these positive instances pre-processed and at scale defined for training can be seen in Figure 5.3.

The negative dataset influences both the training time and test performance. Generally speaking hard and large negative datasets imply longer training times, but also better performance. We employed the “Urtho - Negative face Dataset”¹, which consists of a total of 3019 images of landscapes, objects, drawings, etc. To keep the cardinality of the negative and positive datasets balanced we randomly selected 1905 of the Urtho images. A sample is presented in Figure 5.4.

5.2.0.2 Classifiers' Assessment

The performance of the face detection classifiers is measured regarding percentage of hits (%H), number of false alarms (FA) and percentage of correct instances (%C). The results are compared with the ones in the ground truth file, and if the result matches or lays within the tolerance area defined by the performance tool parameters, it is hit. However, if it lays outside the tolerance area, it is counted as a false alarm. If no face is detected and a face exists, it is counted as a miss. Finally, an instance is considered correctly classified if, and only if: (i) all the ground truth faces are detected (note that an image may contain several faces) and; (ii) the regions where the faces were detected match the expected region. A negative instance is classified as correct if the classifier detects no faces in the input image. To illustrate these concepts, three instances are displayed in Figure 5.5 with example cases of correct and incorrect instances. The parameters for the performance assessment task are defined in Table 5.3 and are based on the default parameters of OpenCV's performance tool and the work of Lienhart et al. (2002). In our performance evaluation, we focus on the following metrics: percentage

¹ Tutorial haartraining — <http://tutorial-haartraining.googlecode.com/svn/trunk/data/negatives/>

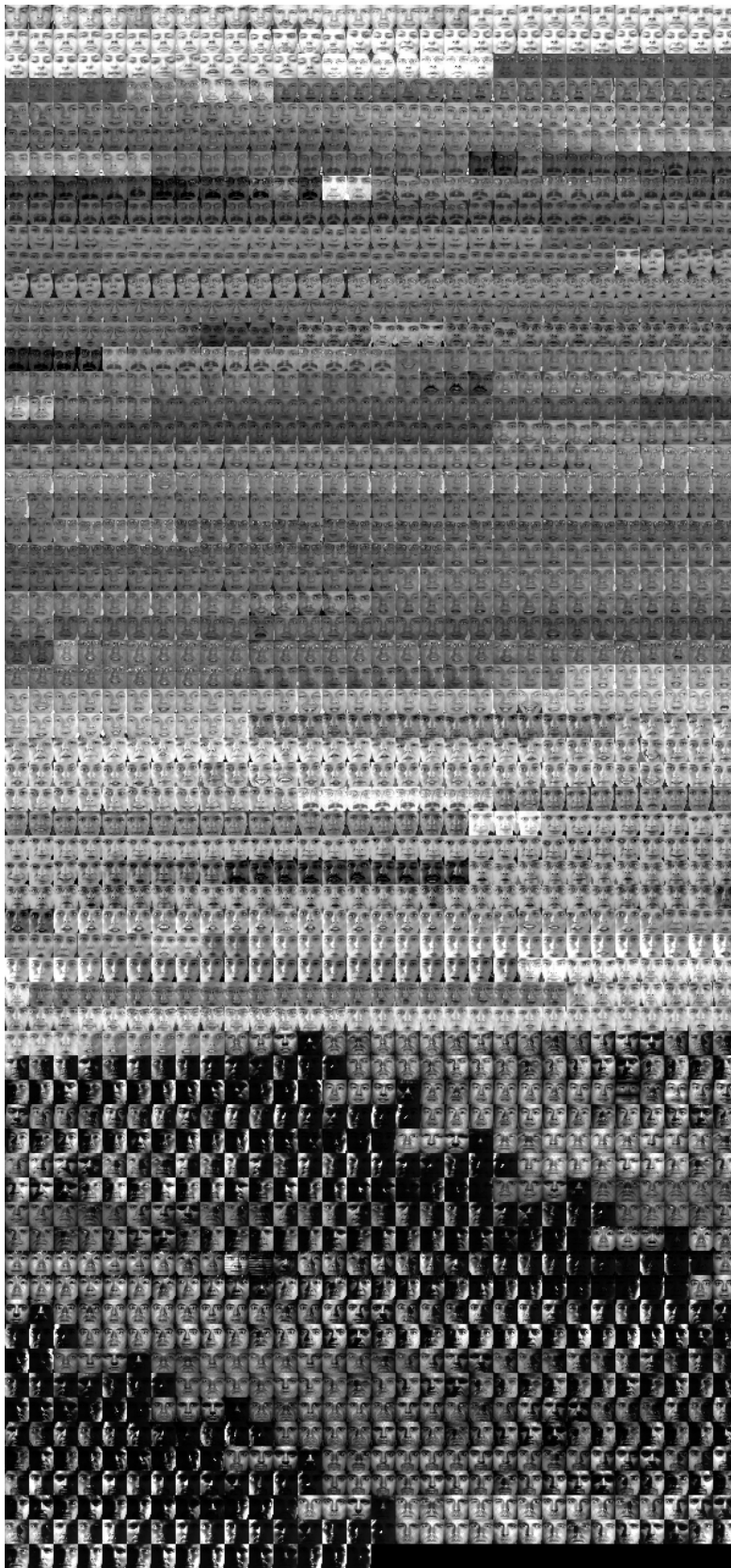


Figure 5.3: All positive instances of the training dataset.

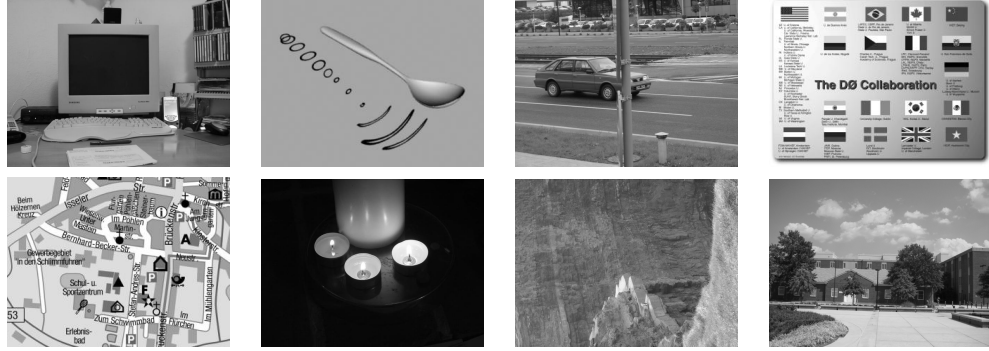


Figure 5.4: Sample of images of the training dataset considered as negative instances.

Table 5.3: Performance tool parameters.

Parameter	Setting
Minimum window width	20
Minimum window height	20
Scale factor	1.1
Maximum size difference factor	1.5
Maximum position difference factor	0.3

of hits (%H), the total number of false alarms (FA), the percentage of correct instances (%C).

Considering the goals of these experiments, the primary interest is the comparison of the performance of the classifiers, from the initial iteration to the last, using different setups. We assess the performance in three test datasets:

1. Flickr – 2166 negative images, from the Flickr search engine²;
2. Feret – 902 positive images with 902 faces, one per image, from Facial Recognition Technology Database³;
3. CMU-MIT – 130 positive and negative images containing a total of 511 faces (Rowley et al., 1998b)⁴

The Flickr image dataset consists of images retrieved from a search in Flickr using the keyword “image”. We excluded images that contain frontal human faces. This results in a negative dataset composed of landscapes, buildings, animals, computer screenshots and various objects. The Feret test dataset is a positive dataset composed

² Flickr – <https://www.flickr.com/search/>

³ The Feret Database – <http://face.nist.gov/colorferet/colorferet.html>

⁴ CMU Database – http://vasc.ri.cmu.edu/idb/html/face/frontal_images/

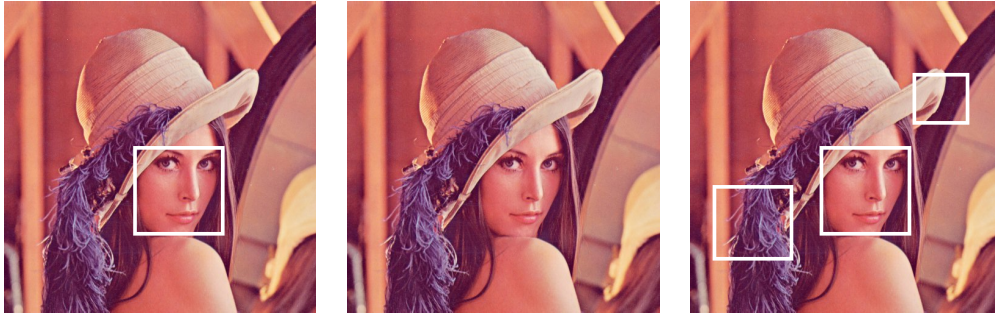


Figure 5.5: Three different detection outcomes. On the left we have a correct instance, where only one detection was made in the right region, hence considered a hit. In the middle we have an instance where no detections were made, meaning that no FAs were encountered but the face was not detected, counting it as a miss and this instance as incorrect. On the right we have a case where the face was correctly detected, hence granting a hit, but there are two FAs and for that reason it is also considered an incorrect classification.

of grayscale frontal faces, one face per image with a simple background. The dataset was manually filtered to exclude instances that did not correspond to frontal faces (e. g., partial profiles). The purpose of using this test dataset is to test the ability of the classifiers to detect a clear frontal face. Finally, the CMU-MIT image dataset gathers multiple types of images, such as cd-covers, wallpapers, photographs, buildings. This dataset is commonly used to compare the performance of face detectors (Kienzle et al., 2005; Lienhart et al., 2002; Viola and Jones, 2001). Samples from the described dataset can be seen in Figure 5.6. All the positive instances of Feret and CMU-MIT can be seen in Figure 5.7 and Figure 5.8, respectively.

5.3 PROOF OF CONCEPT

In the following Subsections we describe and analyse the results of the first steps towards the improvement of classifiers. Based on the conclusions from the experiments in Chapter 4, with EFFECTIVE we were able to generate instances that are wrongly classified by the classifier. We aim to use the misclassified instances to expand the classifier's dataset in order to eliminate its shortcomings. More specifically, in the next Sections, we aim to generate false positives, retrain the classifiers with them, and analyse their impact on the performance classifiers.



Figure 5.6: Instances from the test datasets. On the top row, images of the Flickr dataset; on the middle row, images of the Feret dataset; and on the bottom row, images of the CMU MIT dataset.

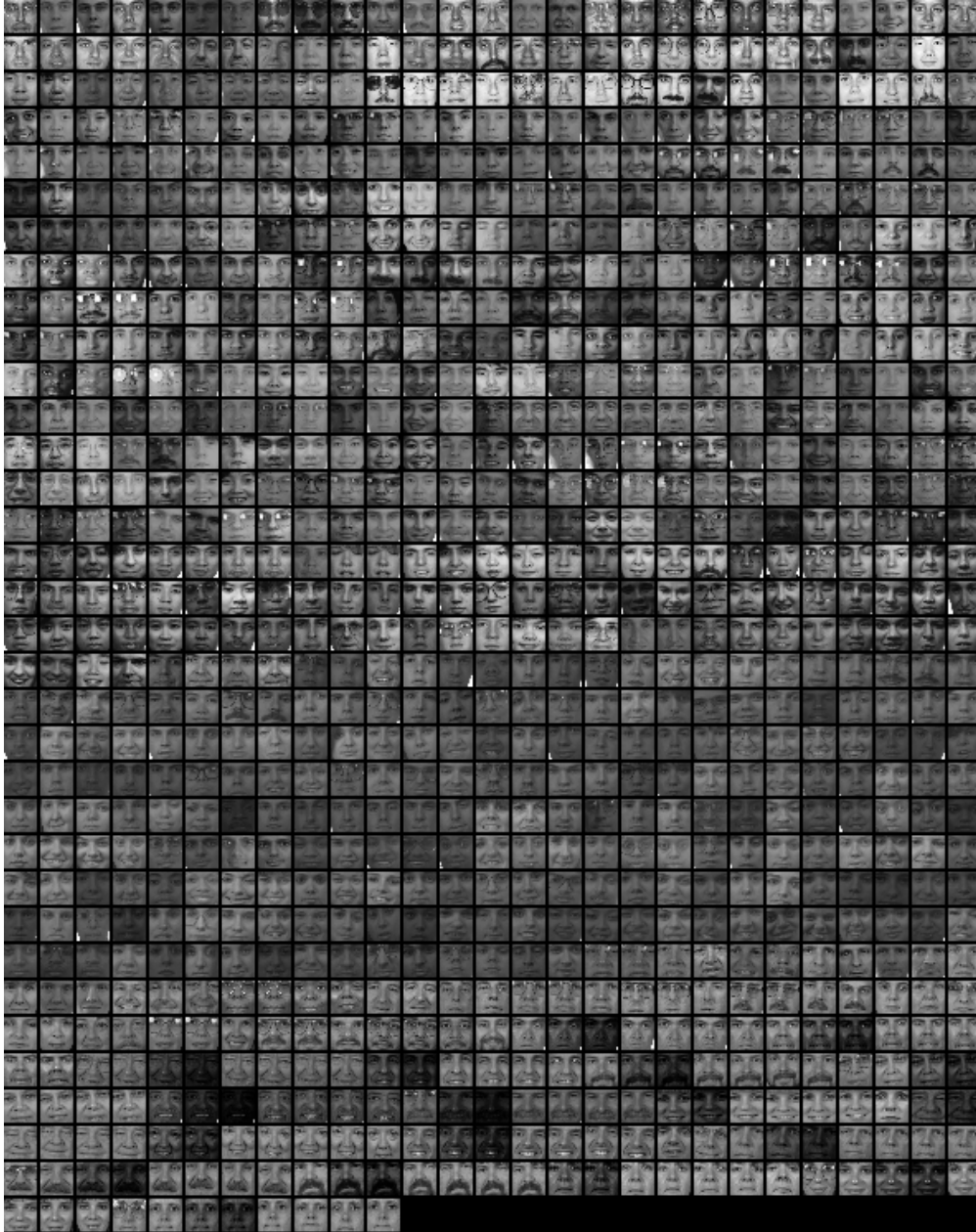


Figure 5.7: All positive instances of the Feret dataset.

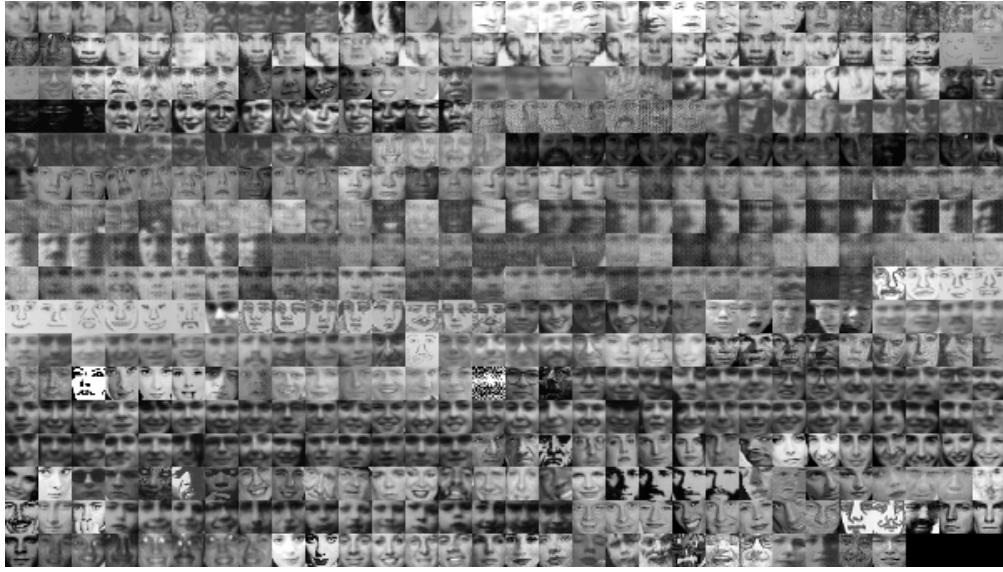


Figure 5.8: All positive instances of the CMU-MIT dataset.

Table 5.4: Parameters of the **GP** engine.

Parameter	Setting
Population size	50
Number of generations	100
Crossover probability	0.8 (per individual)
Mutation probability	0.05 (per node)
Mutation operators	sub-tree swap, sub-tree replacement, node insertion, node deletion, node mutation
Initialisation method	ramped half-and-half
Initial maximum depth	5
Mutation max tree depth	3
Function set	$+$, $-$, \times , $/$, min, max, abs, neg, warp, sign, sqrt, pow, mdist, sin, cos, if
Terminal set	x , y , random constants

5.3.1 Single Run

As previously mentioned, although the framework proposes the use of several parallel classifiers, each with its evolutionary run, we only used one. So, in this experiment, we train one **CS**, perform 30 independent **EC** runs, and perform one framework iteration.

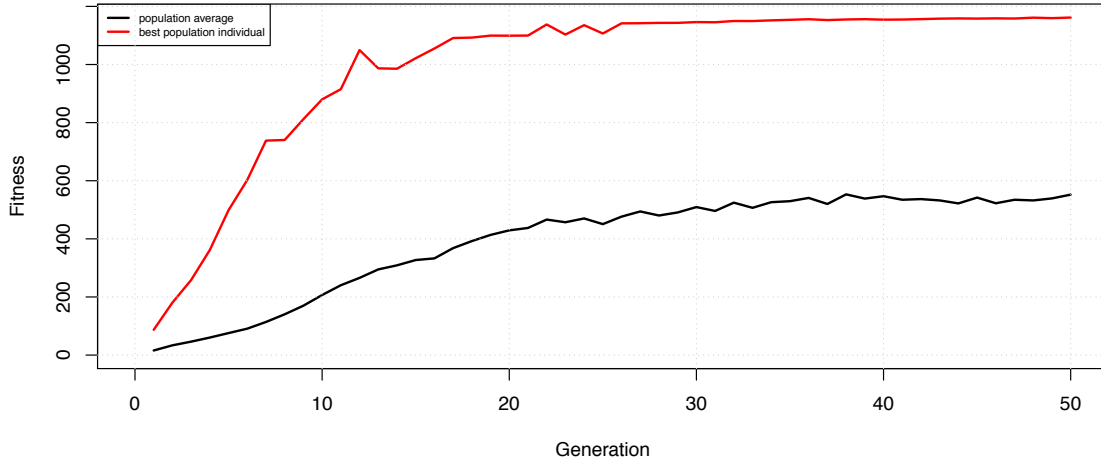


Figure 5.9: Evolution of fitness across generations. Results are averages of 30 independent runs.

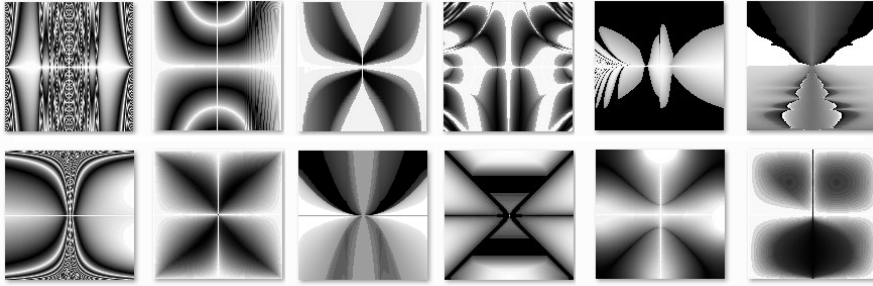


Figure 5.10: Evolved images classified as faces by the classifier.

To analyse the impact of the evolved instances of each [EC](#) run, we train a classifier, one per [EC](#) run, with the instances of the corresponding [EC](#) run, resulting in 30 different classifiers. The training and detection parameters are described in Section [5.2](#). The [EC](#) engine parameters are the same from Section [4.2](#) and are presented in Table [5.4](#).

Figure [7.6](#) displays the evolution of the population average fitness and of the best population individual across generations. In essence this chart shows that in successful runs, the [EC](#) engine finds images that are classified as faces in few generations. Note that runs where the framework was unable to find images classified as faces were discarded. These runs are useless for improving the classifier's performance since no images would be added to the dataset.

Figure [5.10](#) presents examples of images evolved in different evolutionary runs. All of these images have been considered faces by the classifiers. This highlights the shortcomings of the classification system, based on a state of the art classification approach, and further indicates the ability of the [EC](#) engine to exploit these shortcomings finding images that are false positives.

Table 5.5: Results attained by the initial classifier and the average of 30 classifiers trained with images of single **EC** runs.

	Flickr		Feret			CMU			
Classifier	FA	%C	H	FA	%C	H	FA	%C	AVG(%C)
Initial	861	73.5	852	97	85.6	193	60	48.5	69.2
Average	750.9	76.4	845.3	86.6	85.4	194.8	47.0	51.8	71.2

Table 5.5 presents a synthesis of the results achieved, presenting the performance of the initial classifier and the average performance of the 30 classifiers (Average) created from a single iteration of **EFFECTIVE**.

Focusing on the comparison of the initial classifier with the average performance of the framework classifiers, the most striking difference in performance is the significant decrease in the number of FAs which occurs for all test datasets. In average, for the three datasets, there is a decrease of 15% in terms of FAs. Adding false positives to the negative training dataset results in classifiers that are more “demanding” than the initial one when it comes to consider the presence of a face in an image. As a consequence, it becomes more robust and precise in the detection, which leads to a decrease in the number of false positives.

The disadvantage is that some face images may go unnoticed. In fact, a decrease in the %H occurs in the Feret test dataset (94.4 vs. 93.6, which represents a decrease of less than 1%). More importantly, in the CMU test dataset, the percentage of correctly identified images increases for all test datasets. As expected, the improvements of performance are more noticeable in the Flickr dataset, which is composed exclusively of negative images.

Although in this Section we focus on examining the behaviour of **EFFECTIVE**, it is important to notice that from a practical perspective we do not need 30 classifiers, we just need one. In the next Section we analyse the experiments performed using multiple runs.

5.3.2 Aggregation of Multiple Runs

In order to manage the results of multiple runs we resort to the Supervisor module of the framework. As stated in Chapter 3, this module manages the generated instances from different **EC** runs. Thus, after finalising the N different independent **EC** runs, started with different random seeds, the resulting individuals are submitted to a Supervisor module responsible for choosing the set of images to be added to the training set. This module comprises two parts: select and filter. Select creates a sub-set of negative examples from the **EC** runs. In the experiments performed in this Section we consider two selection modes: *Aggregator* and *External*.

Table 5.6: Results attained by the initial, the external classifiers and by all the framework classifiers in three independent test datasets. Performance improvements over the initial model and Average models are presented in bold typeface. Improvements over the initial model in italic. Decreases of performance over the initial model are underlined.

	Flickr		Feret			CMU			
Classifier	FA	%C	%H	FA	%C	%H	FA	%C	AVG(%C)
Initial	861	73.5	94.5	97	85.6	62.1	60	48.5	69.2
Fdlib	499	82.5	77.4	32	75.7	39.5	34	32.3	63.5
Average	750.9	76.4	93.7	86.6	85.4	62.6	47.0	51.8	71.2
<i>Manual</i>	456	84.6	<u>93.2</u>	52	88.7	65	35	<u>50.8</u>	74.7
<i>Aggregator Unequal</i>	522	82.2	<u>94</u>	53	89.2	68.8	40	57.7	76.4
<i>Aggregator RMSE</i>	769	75.7	<u>93</u>	<u>125</u>	82.2	63.7	36	55.4	<i>71.1</i>
<i>External Unequal</i>	412	84.9	<u>92.6</u>	38	89.0	<u>60.1</u>	38	53.1	75.7
<i>External RMSE</i>	319	88.6	<u>91.6</u>	51	86.7	<u>61.7</u>	39	51.5	75.6

In *Aggregator* mode all the images generated throughout the **EC** runs identified as containing a frontal face are selected. In *External* mode an external classifier is used and images will only be selected when the internal and external classifier disagree (e. g., the classifier used in the **EC** runs classifies the image as having a face, yet the external classifier does not recognise a face in the image). The rationale for using an external classifier is the following, some of the evolved images may actually look like frontal faces, as such, adding them to the training set may hinder performance. The chosen external classifier was Face Detection Library (FDLib) developed by Kienzle et al. (2005).

Once the selection process is over, the resulting sub-set of images is submitted to a filter operation to remove similar images. We considered the filter methods defined in Section 3.4 *Unequal* and *RMSE*. Since we are in a image classification scenario, these filter modes operate at the image's pixel level (e. g., perform comparison at the pixel level).

We use 30 parallel EC runs and the images generated throughout these runs are gathered and submitted to the Supervisor. The resulting sub-set of images is then added to the training set and the classifier is retrained. As previously stated, we consider two select modes – *Aggregator* and *External*– and two filter modes – *Unequal* and *RMSE*– which results in a total of four combinations. Additionally, we consider the option of using a human Supervisor in charge of manually handling the selection and filtering process (*Manual*).

Table 5.6 presents a synthesis of the results attained in the three considered test datasets by the: initial classifier; the Average classifier; the classifier resulting from

manual selection of examples; and the four classifiers resulting from the combination of select and filter modes.

When comparing the performance of the classifiers resulting from parallel **EC** runs with the initial model we find an overall performance improvement. The number of FAs decreases for all considered test datasets, the only exception is the number of FAs attained by the classifier created with the *Aggregator RMSE* strategy in the Feret test set. Furthermore, adding examples from multiple EC runs reduces significantly the FAs in comparison to the Average runs (34.04% on Flickr, 26.30% Feret, 20% CMU) and to the initial classifier (42.44% Flickr, 34.23% Feret, 37.33% CMU). Combining these results with the ones attained by the Average experiments, it becomes clear that the addition of evolved false positives to the training dataset significantly reduces the FA rates. The results also indicate that parallel EC runs models tend to perform better than the Average model in terms of FA.

While comparing the %H of the parallel **EC** models to those of the initial classifier, we observe a tendency to the decrease in the %H in the Feret dataset. This is an expected result, since as we only expand the set of negative examples, the classifiers tend to become more "cautious" in identifying faces. Interestingly, the same behaviour does not occur in the CMU dataset.

In terms of percentage of correctly classified instances, the parallel **EC** runs models achieve an average increase of performance of 5.52% over the initial model and a 3.46% performance increase when compared with the Average.

Focusing on the comparison of the results attained by the two selection modes – *Aggregator* and *External* – the *External* models tend to perform better in terms of percentage of correct instances (5.0% average improvement). Concerning the filter methods – *Unequal* and *RMSE* – equal achieves a better average performance in terms of percentage of correct instances (2.69% improvement). In these experiments, the only case where the advantage of using parallel EC runs is not clear is when the *Aggregator RMSE* strategy is employed. Nevertheless, even in this case, there is an improvement in terms of percentage of correctly classified instances over the initial classifier. Overall, the best classifier resulted from using a *External RMSE* strategy, which matches our expectations.

It is important to notice that the external classifier, FDLib, has shortcomings. In fact, although it achieves low FA rates in all test datasets, in terms of percentage of correctly classified instances it performs worse than the initial classifier in the Feret and CMU test datasets. One may consider that the performance of the classifiers created through the proposed framework would be limited by the performance of the external classifier. Although this is true to some extent, the experimental results point in a different direction.

The manual supervision experiment was created to assess what differences of performance would be observable when using a human as Supervisor. With the exception of the classifier resulting from the use of the *External RMSE* strategy, all other parallel

EC runs models achieved performances comparable to those attained by the classifier created through human supervision. In fact, *External Unequal* and *External RMSE*, surpass the performance of the *Manual* classifier. These results suggest that is possible, without the intervention of a user, to boost classifiers in an automatic way, achieving reasonable performance increases. Although the external classifier has significant weaknesses, these weaknesses are not under evolutionary pressure and, therefore, cannot be exploited by the **EC** runs to create instances that are misclassified by the external classifier. As such, in spite of its weaknesses, the external classifier remains a valid alternative for supervision purposes.

When we compare the performance of the *External* models with the performance of the FDLib classifier, we observe that FDLib is clearly outperformed in terms of %C classified instances in all test datasets, as is the initial classifier. The number of FAs obtained by the *External* models is lower than the one attained by FDLib on the Flickr dataset, higher on the Feret dataset and CMU datasets. It is always lower than the initial model. Thus, overall, the *External* models significantly outperform both the initial and external classifier.

5.4 SUMMARY

In this Chapter we performed a proof of concept of using **EFFECTIVE** to improve the performance of classifiers. First we performed experiments using one **CS** model and performed 30 **EC** runs. The instances of each **EC** run were used to train a single classifier. We analysed the performance of the resulting classifiers in the test datasets. Based on the average results, a reduction of the FAs and an increase in terms of %C was attained.

In the other set of experiments we tested the use of the Supervisor to select and filter instances from multiple **EC** runs in one iteration of the framework. The experimental results attained indicate that by choosing instances from multiple **EC** runs and using them to retrain the classifier, resulted in a better overall performance when compared to the initial, the Supervisor's classifier and individual **EC** run classifiers. These results are comparable to the result obtained by a classifier trained with manually selected instances from all the **EC** runs. Furthermore, the results suggest that although the performance of the Supervisor's classifier is worse than the initial classifier, in the role of Supervisor it contributes to the selection of instances that impact the performance of the classifiers.

ASSESSING AND IMPROVING CLASSIFIERS' PERFORMANCE

In the previous Chapter we presented a series of experiments that show the feasibility of using **EFFECTIVE** for assessing and improving classifier's performance. In this Chapter, we explore the approach further by performing multiple framework iterations. We present and analyse the following experiments: (i) expand the negative dataset (Section 6.1); (ii) expand the positive dataset (Section 6.2); and (iii) expand the negative and positive datasets (Section 6.3).

6.1 EXPANDING THE NEGATIVE DATASET

After the proof of concept with parallel evolutionary runs and aggregation of different instances from different runs we explore the framework further. We start by exploring the contribution of multiple framework iterations. We also analyse the training phase and how the size of the dataset progresses. We address the evolutionary process and how these evolved instances impact the framework by doing a control experiment (Section 6.1.5). Lastly, we compare the different approaches in terms of performance metrics, described before in Section 5.2.0.2. In the next Section we describe the changes and updates to the framework that allowed us to further test and validate the framework.

6.1.1 Framework Changes

The **CS** module, **EC** engine and fitness assignment were maintained from the previous experiments. **EFFECTIVE** relies on the ability of the evolutionary engine to find and exploit the weaknesses of the classifiers to increase the quality of the dataset. The disposition of the **EC** to find shortcuts that exploit weaknesses of the fitness assignment scheme is well-known (Baluja et al., 1994; Machado et al., 2007a; Spector and Alpern, 1994; Teller and Veloso, 1995). In this case, the objective is to evolve false positives: images that are classified as faces, but that should not have been classified as faces. Adding these false positives to the training dataset and re-training the classifier promotes the correction of its exploitable flaws. With this in mind, after finalising all the **EC** runs, the resulting instances are submitted to the Supervisor module, which is responsible for choosing the set of instances that are going to be added to the training dataset. In this experiment, we consider a candidate instance an instance that is classified as belonging to the positive class, i. e., classified as a face.

In the selection part we collect a subset of images from the **EC** runs. We consider the two selection methods: *Majority* and *External*. As we previously show in Chapters 4 and 5, although most of the evolved images are false positives, some of them actually look like faces and, as such, adding them to the training dataset may hinder the performance of the classifier. Thus, the goal of the selection stage is to discriminate between the true positives and false positives.

In the *Majority* selection method, all the instances generated throughout the evolutionary runs and identified as containing a frontal face are gathered. All the classifiers of the **CS** evaluate each instance and if it is identified as not containing a face by the majority of the classifiers, the instance is selected for the subset. The idea is to consider instances that the majority of the **CS** consider as an error (false positives), that should be corrected.

The *External* selection method relies on the feedback of an external classifier. The rationale is that the external classifier is not subjected to evolutionary pressure, in the sense that its flaws are not directly exploitable by the **EC**, as such, the classification it produces for evolved images are prone to be more accurate.

In this work, an *off-the-shelf* **LBP** cascade classifier for frontal face detection from the OpenCV distribution is used as the external classifier (Ahonen et al., 2006), i. e., the supervisor's classifier. It was selected due to its state of the art relevance and similarity with the type of classifiers used in our approach. The difference between the **LBP** classifier and the ones used in this work is the type of features used in the detection phase (as we have seen in Section 4.2.2.1). Instead of using Haar features, it uses **LBP** features. The evolved images are only selected if they are classified as containing faces by the **CS** and as **not** containing faces by the **LBP** detector.

When successful, the **EC** runs tend to converge to a certain kind of image and to produce various similar images. The addition of all these images would lead to an explosion of the size of the training set, without significantly improving performance. To avoid the addition of visually similar images we introduce a filtering stage. For this purpose we considered several image similarity metrics (see e. g. (Goshtasby, 2012)). Taking into consideration the task at hand – which involves filtering images that tend to result from converging **EC** processes – and the temporal complexity of the metrics, we developed two filtering schemes: *Unequal* and *RMSE*. The former performs a pixel based comparison of the images' subset, discarding images that are equal. The latter *RMSE* mode, also performs a pixel based comparison, calculating the root mean square error (*RMSE*) between all pairs of images of the subset, and discards images that are below a given *RMSE* threshold. The threshold was attained empirically and kept fixed from previous experiments (Section 5.3.2).

Table 6.1: Framework parameters.

Parameter	Setting
Framework iterations (N)	10
Classifiers per iteration (C)	30
EC runs per classifier (E)	1
Minimum candidates per EC run	300
RMSE threshold	0.10

6.1.2 Experimental Setup

The experiments are designed to test the different possible combinations of filter and select methods, using multiple classifiers and multiple framework iterations.

Each framework iteration involves training several [CS](#), conducting an [EC](#) run for each [CS](#), selecting and filtering the results to prepare the next framework iteration. We define the different setups as the combination of selection and filter methods, as follows: *Majority Unequal*; *Majority RMSE*; *External Unequal* and; *External RMSE*.

The overall framework parameters are presented in Table [7.1](#), which yields 1200 distinct Haar cascade classifiers.

The settings for the [GP](#) engine for each [EC](#) run are presented in Table [7.4](#). An example of the images generated in the initial population is depicted in Figure [6.1](#). Results from Section [5.3.1](#) show that these parameters allow the generation of images classified as containing faces in less than 50 populations of 100 individuals. Preliminary tests indicated that some [EC](#) runs were unable to find a significant number of images detected as faces, biasing the analysis of the results. To overcome this problem, when an [EC](#) run fails to find at least 300 true positive instances, the run is discarded, and a new evolutionary run with a different seed is performed. This process is repeated until this minimum is reached. Only instances of successful [EC](#) runs are given to the supervisor for selection and filtering. It is important to state that we are only discarding these instances to ease the analysis. Thus, in a practical scenario it would be more advisable to use these images.

6.1.3 Experimental Results

We start by describing the results of the evolutionary process and its behaviour for the different defined setups. We show that it is possible to guide evolution with the current fitness scheme, and that the results progress from one framework iteration to the next. The results of each framework iteration and their differences are analysed in Section [6.1.3.1](#). We discuss the assessment of the classifiers, by performing a statistical analysis of the results of each setup (Section [6.1.4](#)). Then, we perform *Control* experiments, comparing the results obtained using evolved instances, i. e. adversarial

Table 6.2: GP engine parameters.

Parameter	Setting
Population size	100
Number of generations	50
Crossover probability	0.8
Mutation operators	sub-tree swap, sub-tree replacement, node insertion, node deletion, node mutation
Initialisation method	ramped half-and-half
Initial maximum depth	5
Mutation max tree depth	3
Function set	+, -, *, /, min, max, abs, sin, cos, if, pow, mdist, warp, sqrt, sign, neg
Terminal set	x, y, scalar random constants
Phenotype width (pixels)	64
Phenotype height (pixels)	64

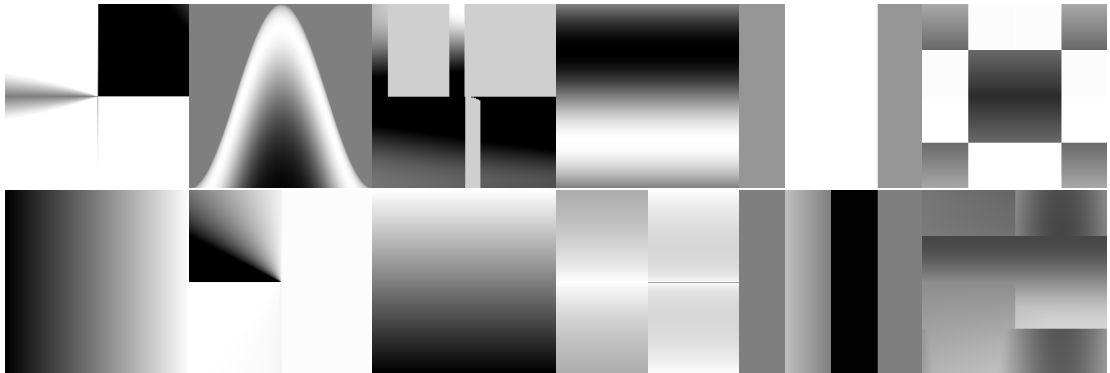


Figure 6.1: Sample of images generated in the starting population.

instances, to those obtained using datasets of equal size, without evolved instances, generated via a standard non-adversarial method for [FD](#) (Subsection [6.1.5](#)).

We rely on statistical tests to make inferences about the experimental results. First we apply the Lilliefors test (Lilliefors, [1967](#)) to test whether the samples follow a normal distribution or not. The test showed that the data does not follow the normal distribution. Thus, we have to use the Kruskal-Wallis between the groups to assess if statistically significant differences exist. When differences exist, we rely on post-hoc tests, using Wilcoxon-Mann-Whitney test (Wilcoxon, [1945](#)) to perform pairwise comparisons. All the tests are performed with a confidence level of 99%, i.e., the p-value for rejecting the null hypothesis has to be less than 0.01. We use the Bonferroni correction ($\alpha/\text{number of comparisons}$) in the cases where multiple comparisons were performed (Bonferroni, [1935](#)).

6.1.3.1 Training Phase and Evolutionary Process

In each iteration of the framework, we train 30 distinct classifiers. As such, since we have 4 different framework setups and 10 iterations, the total number of trained classifiers is equal to 1200. For each of the trained classifiers we perform a single [EC](#) run. These [EC](#) runs are guided by its corresponding classifier to evolve images that are detected as belonging to the positive class (i. e., images classified as containing a face). 30 distinct [EC](#) runs start with the same initial population, but each with a different classifier to assign fitness.

At the end of each [EC](#) run we count the number of images classified as belonging to the positive class that were generated. If the number is above the pre-defined threshold (in our case 300, see Table [7.1](#)) the run is considered valid. If this threshold is not achieved, a new [EC](#) run is initiated and the process is repeated until the threshold is surpassed. After all [EC](#) runs for all classifiers have ended, all images from the valid [EC](#) runs are gathered, and we advance to the supervisor module. After the supervisor performs its selection and filtering methods, the resulting instances are added to the training dataset and a new framework iteration begins.

6.1.3.2 Fitness Evolution

In the first iteration of the framework (Iter. 0), the positive and negative datasets are the same for all supervisor setups. Moreover, the initial classifiers are the same for all the setups. Hence the results of the EC runs in iteration 0 are the same across all setups.

Figure [6.2](#) shows the evolution of the fitness of the best individual of each generation (Max) and of the average fitness (Avg), during the initial framework iteration. We present the results obtained using the 30 valid EC runs (solid line), which have reached the pre-established threshold of candidates, and the ones obtained with all runs (valid and invalid), which are depicted by dashed lines.

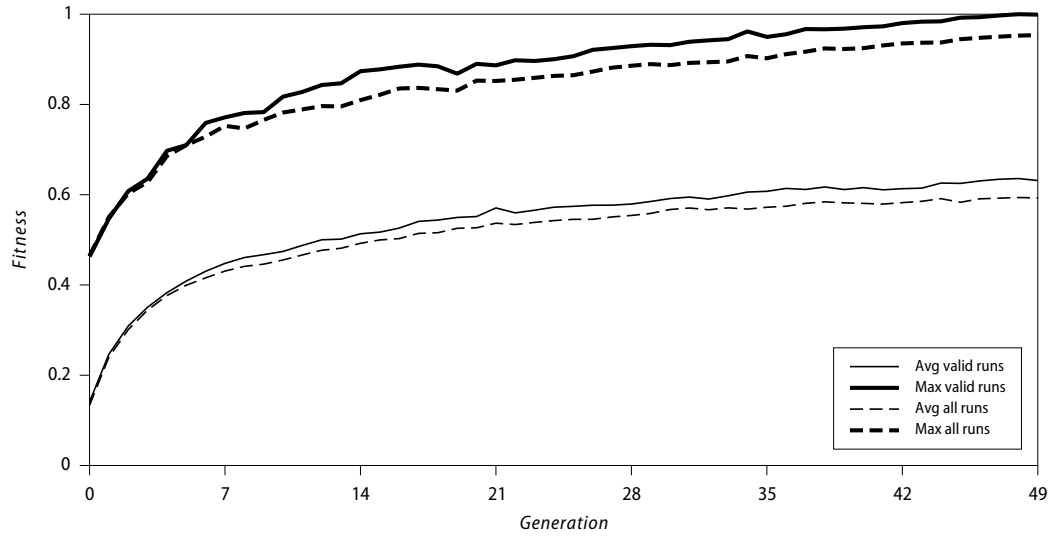


Figure 6.2: Evolution of the average and maximum fitness across generations for the initial framework iteration, considering the valid and all the EC runs. The results are averages of 30 independent runs.

To improve the readability, the fitness values were normalised to the $[0, 1]$ interval by dividing the raw fitness, calculated according to Equation 4.2 by the average maximum fitness found in the course of the framework iteration¹.

As can be observed, the fitness increases rapidly in the first 5 generations, and steadily, although at a lower pace, from there on. This is an indication that the fitness function is capable of guiding the evolutionary process, leading it to the discovery of images classified as members of the positive class. Additionally, the curves concerning the valid runs and all runs depict similar behaviours. This can be explained by two factors: the percentage of valid runs is relatively high (see Table 6.3); all the invalid runs were able to generate images classified as faces, although not in sufficient number, therefore reaching high fitness values.

After the initial iteration, the dataset size for the four setups becomes different, due to the different supervision setups. Therefore, from there on, the classifiers are trained using different datasets and, as such, will have different performances.

Figure 6.3 shows the evolution of the fitness of the best individual of each generation (Max) and of the average fitness for the final framework iteration. As previously, we present results for the 30 valid EC runs and for all runs.

As depicted, the curves exhibit the same overall behaviour. However, the differences among valid runs and all runs are more visible. This can be explained by the fact that, in spite of a significant drop in the percentage of valid EC runs (from 47% to, 5%), the

¹ It is important to notice that the raw fitness values resulting from two different classifiers are not directly comparable, since the number of stages and inner structure tends to be different.

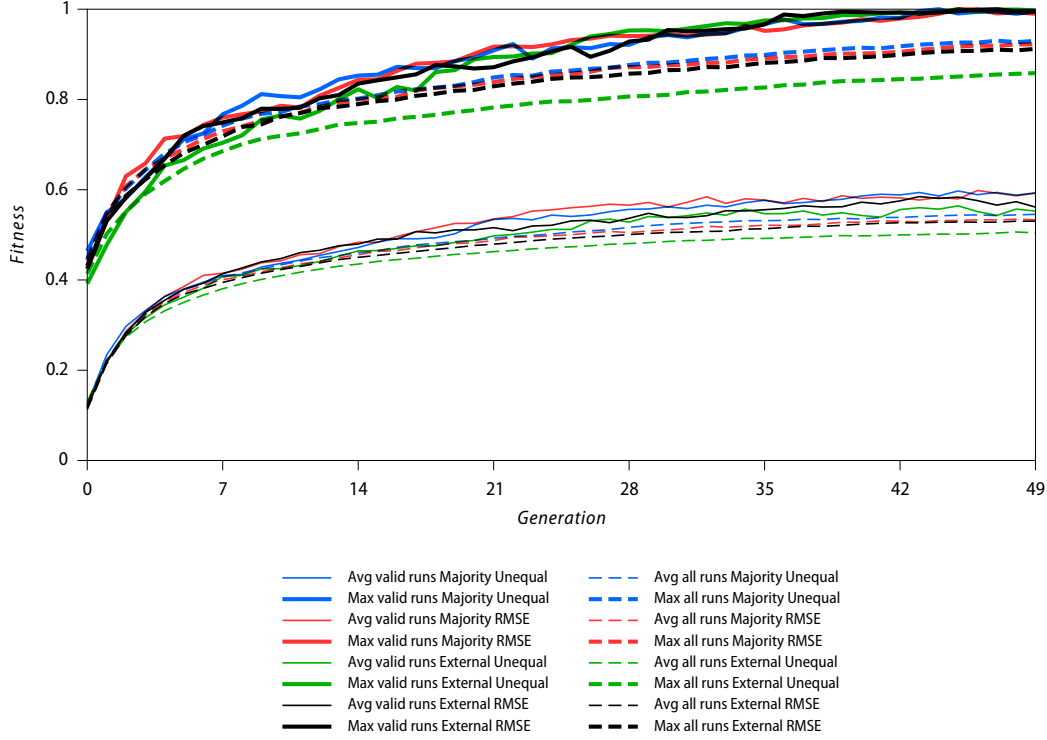


Figure 6.3: Evolution of the average and maximum fitness across generations for the last framework iteration for all Supervisor setups, considering the valid and all the EC runs. The results are averages of 30 independent EC runs.

percentage of invalid runs able to evolve faces is still relatively high (74%). The *Majority* based setups yield the highest percentage of invalid runs that were able to evolve faces (88% for *Majority Unequal* and 86% for *Majority RMSE*). These results indicate that the *External* setups are likely to be the ones that result in a higher improvement of performance of the classifiers, since it is harder to “fool” them.

Overall, these charts indicate that although the task of the [EC](#) engine becomes tougher as the number of iterations increases, the [EC](#) is still finding new instances of the positive class in the final iteration of the framework.

To better understand how the addition of evolved instances to the dataset increases the difficulty of the task of the evolutionary engine, we present in Table [6.3](#), for each setup and for each iteration, the total number of [EC](#) runs performed in order to reach 30 valid runs, the average number of candidates per [EC](#) run, and the average number of candidates per valid [EC](#) run.

To support the analysis of the evolution of the total number of [EC](#) runs per framework iteration, we performed a non-parametric chi-squared method for statistical pro-

Table 6.3: Number of EC runs necessary to find 30 valid runs, average and median number of candidates per EC run, and average and median number of candidates per valid EC run.

Setup	Iter.	EC runs	Avg candidates all runs	Median candidates all runs	Avg candidates valid runs	Median candidates valid runs
Majority Unequal	0	<u>64</u>	<u>631.09</u>	<u>242.00</u>	1228.23	1069.50
	1	<u>176</u>	<u>173.11</u>	<u>28.50</u>	801.87	722.50
	2	218	142.59	20.50	757.67	722.50
	3	255	111.76	19.00	648.87	538.50
	4	300	<u>78.63</u>	<u>10.00</u>	534.93	470.50
	5	345	101.28	15.00	795.87	665.50
	6	433	<u>63.09</u>	<u>1.00</u>	629.40	554.00
	7	377	<u>76.09</u>	<u>8.00</u>	611.00	586.50
Majority RMSE	8	375	96.53	11.00	789.07	581.00
	0	<u>64</u>	<u>631.09</u>	<u>242.00</u>	1228.23	1069.50
	1	<u>181</u>	<u>168.09</u>	<u>34.00</u>	772.97	646.00
	2	255	<u>109.05</u>	<u>18.00</u>	657.07	578.00
	3	340	91.10	17.00	662.17	556.50
	4	343	97.74	23.00	659.00	479.00
	5	272	121.19	11.50	796.77	640.00
	6	474	<u>60.67</u>	<u>6.00</u>	603.43	469.50
External Unequal	7	530	61.18	7.00	641.43	525.50
	8	356	85.78	10.50	683.87	540.50
	0	<u>64</u>	<u>631.09</u>	<u>242.00</u>	1228.23	1069.50
	1	<u>473</u>	66.43	1.00	732.73	622.50
	2	<u>303</u>	<u>105.14</u>	<u>16.00</u>	686.60	540.50
	3	894	41.61	1.00	690.73	682.00
	4	645	<u>62.59</u>	<u>6.00</u>	760.03	652.00
	5	680	43.75	2.00	615.47	519.00
External RMSE	6	1008	36.33	3.00	611.67	452.00
	7	923	<u>43.01</u>	<u>5.00</u>	701.83	547.00
	8	1135	32.86	2.00	726.27	690.50
	0	<u>64</u>	<u>631.09</u>	<u>242.00</u>	1228.23	1069.50
	1	<u>201</u>	<u>144.72</u>	<u>30.00</u>	699.27	544.50
	2	<u>234</u>	<u>117.65</u>	<u>24.00</u>	614.50	407.00
	3	381	<u>84.70</u>	<u>9.00</u>	748.40	544.50
	4	386	<u>74.41</u>	<u>8.00</u>	660.10	642.00
	5	486	52.02	5.00	577.10	510.50
	6	510	59.34	4.00	648.33	533.50
	7	545	51.10	5.00	605.07	508.50
	8	698	42.67	4.00	568.47	492.50

portion analysis. This analysis reveals that there are statistically significant differences² between the proportion of valid **EC** runs observed during the initial iteration, and all remaining ones. Statistically significant differences between the proportion of valid **EC** runs of a given iteration and the previous one are indicated in *italic*. Statistically significant differences between one iteration and the last one are depicted in underline. As it can be observed, the differences between consecutive iterations are rarely statistically significant. In other words, although the difficulty of the task tends to increase, the differences from one iteration to the other tend to be below the threshold that would allow to classify the differences as statistically significant. The *External RMSE* setup is, arguably, the perfect example of this behaviour. As it can be observed, the total number of **EC** runs steadily increases from one iteration to the other, but the differences are always too subtle to be statistically significant. The analysis of the differences in the last iteration reinforces this interpretation of the results.

For the analysis of the number of candidates we perform the Wilcoxon-Mann-Whitney test with a confidence level of 99% with Bonferroni correction for multiple tests, which implies setting significance level to 0.01/8 for the tests involving comparisons with iteration 8, and 0.01/2 for all the remaining ones.

The results concerning the number of candidates for all **EC** runs confirm the previous analysis. There are statistically significant differences among the results of the initial iteration and all the remaining ones. From one iteration to the other the differences are rarely statistically significant, but several statistically significant differences can be observed from the first iterations and the final one.

When we focus on the number of candidates per valid **EC** run we observe that there are almost no statistical significant differences. Although it is increasingly difficult to find a valid **EC** run, when a valid run is found, the behaviour of the **EC** algorithm within the run tends to be similar for all iterations except for the initial one.

Comparing the results obtained by the different setups according to these metrics is likely meaningless. Nevertheless, it is interesting to notice that the *External RMSE* setup yields a consistent behaviour, with a steady increase of the number of **EC** runs and decrease of the average number of candidates per run, as the number of iterations increases.

6.1.3.3 Visualising the Evolved Instances

As stated, since the supervisor selects and filters the instances after all populations are gathered, the first iteration will be equal across all setups. To help visualise the type of images generated, a sample is presented in Figure 6.4. We can observe in Figure 6.5 that the patterns of the last iteration images differ from setup to setup. It is important to notice that all the sample instances are images that the classifiers identified as containing at least one face.

² with a confidence level of 99%

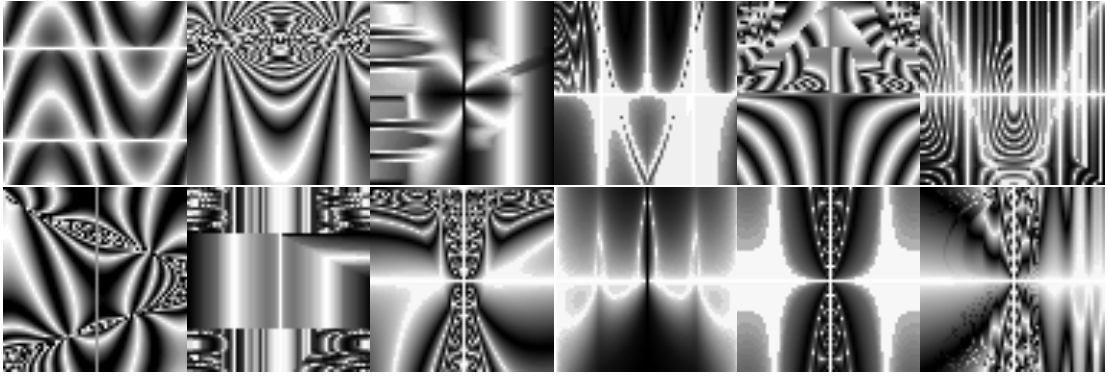


Figure 6.4: Sample of images generated by the framework that were added to the training dataset on the first iteration.

The initial images contain noise and fuzzy patterns. The results of each setup in the last iteration seem cleaner and, in some cases, well-defined patterns emerge. Some of them have parts that resemble a human face. In *Majority RMSE* one can see some rectangular patterns that may appear to be mouths and eyes (first two images on bottom left). *External RMSE* has some patterns that resemble face contours (second image from the top-right position).

One can observe that different images were evolved with different characteristics in all setups. Taking into account the metrics of the evolutionary phase that were previously analysed, the visual results suggest that the different setups explored different areas of the search space.

Overall, the results for the evolutionary metrics and the ones shown in Figure 6.2 and Figure 6.3, suggest that it is possible to properly guide evolution with the chosen fitness scheme (Equation 4.2).

6.1.4 Supervision and Classifier Performance

This subsection focuses on the analysis of the impact of the configuration of the supervisor module in the growth of the dataset size (6.1.4.1) and performance of the classifiers in test datasets (6.1.4.2).

6.1.4.1 Dataset's Size

In Table 6.4 the dataset's size and the number of instances added throughout the iterations are presented. Regarding the selection methods, we observe that *External* adds more negative instances than the *Majority* method. This suggests that the *External*, which uses a LBP cascade classifier, identifies more false positives than the majority of the classifiers that were trained throughout the iterations. This behaviour can be observed by looking at the dataset size of the first iteration (Iteration 1), where all

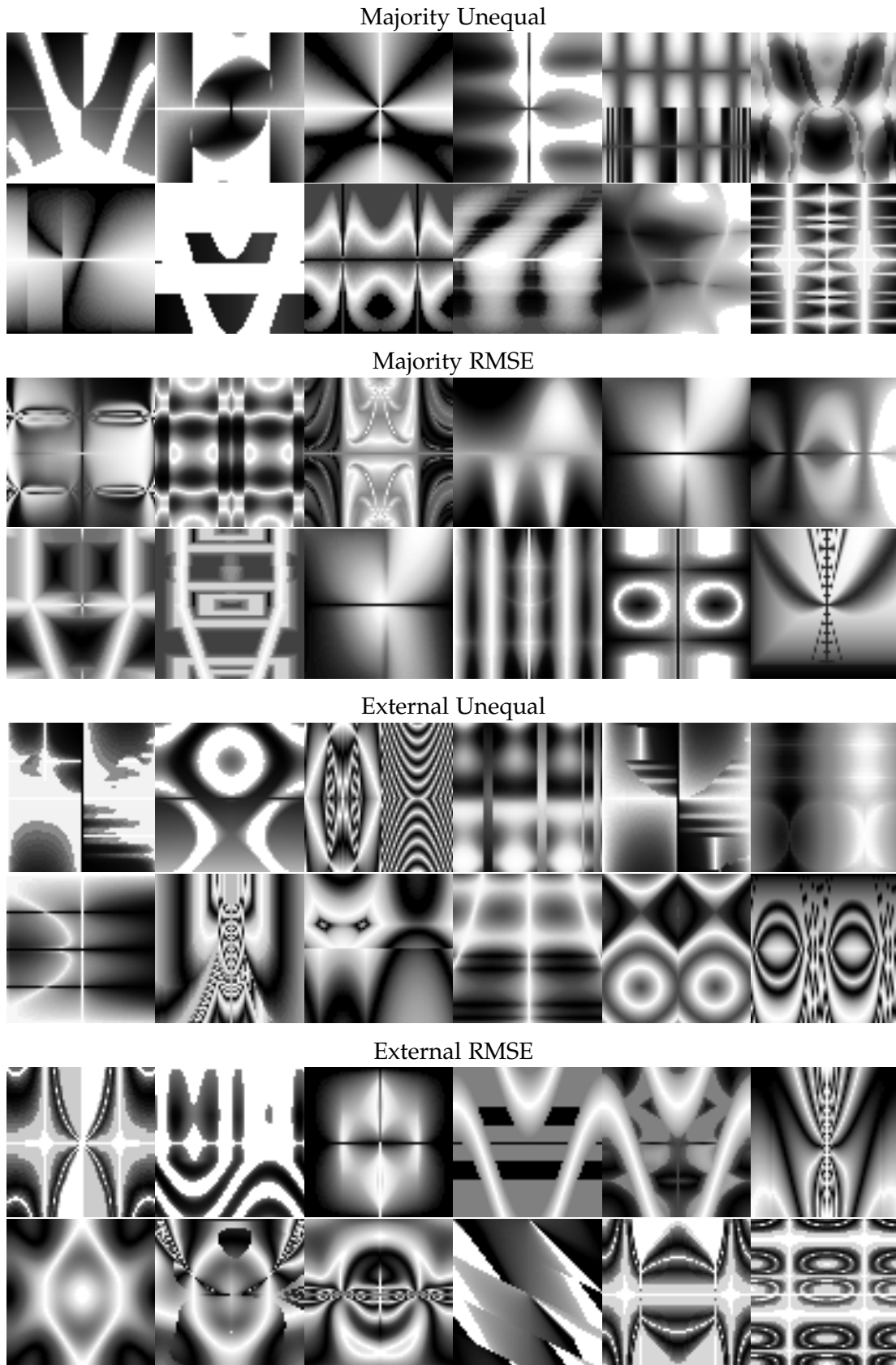


Figure 6.5: Samples of images generated that were added to the training dataset in the last iteration from each corresponding setup.

Table 6.4: Negative dataset's size along the iterations for all setups.

Iteration	Majority				External			
	Unequal		RMSE		Unequal		RMSE	
	size	added	size	added	size	added	size	added
(o) baseline	1905	-	1905	-	1905	-	1905	-
1	10927	9022	7684	5779	14611	12706	9914	8009
2	13395	2468	8730	1046	22428	7817	13515	3601
3	15495	2100	9924	1194	28693	6265	16967	3452
4	16815	1320	11281	1357	35939	7246	19475	2508
5	17610	795	12193	912	42861	6922	22616	3141
6	18717	1107	13196	1003	47890	5029	24703	2087
7	20184	1467	14281	1085	52498	4608	27857	3154
8	21442	1258	14927	646	59528	7030	30825	2968
9	22899	1457	15704	777	63637	4109	32699	1874

evolved instances are the same for all the setups but different number of instances are picked (as seen in subsection [6.1.3.1](#)). The *External Unequal* registers the highest value (14611), which means that in equal conditions for all setups, it selects more instances than the other three setups. This can be explained by the fact that the external classifier is not under selective pressure and, therefore, in the considered experimental conditions, it is more robust than the classifiers that are being trained. Regarding the filtering methods, *Unequal*, as expected, yields a higher number of additions than the *RMSE* method, since it only discards instances that are equal. On the other hand, the *RMSE* method discards images that are similar at the pixel level. Overall, these results show that: (i) the combination of different filtering and selection methods have distinct effects on the size of the dataset and; (ii) the supervisor's classifier used in the *External* setup identifies most of the generated instances as false alarms.

6.1.4.2 Classifier's Performance

Each of the trained classifiers is applied to the test datasets to assess the performance. We group the models per iteration and analyse the differences for each performance metric. The summary of the results obtained in the test datasets are depicted in Table [6.5](#).

As previously, to determine if statistical significant differences occur we use the Wilcoxon-Mann-Whitney test with a confidence level of 99% with Bonferroni correction for multiple comparisons. This implies setting the significance level to 0.01/11 for the tests involving comparisons with iteration 9, and 0.01/2 for all the remaining ones.

In Table [6.5](#) underlined values indicate that there is a statistically difference between the value and the corresponding one observed in the last iteration (9) of each corresponding setup, those in *italic* values indicate that there is a statistically difference between the value and the corresponding one observed with the previous iteration. In addition to the percentage of hits (%H), number of FA and percentage of correct (%C), we include the AVG (%C) which represents the average %C for the considered test datasets. Notice that the framework iterations start with the training of the initial classifiers (iteration 0), which are the same for all the setups.

In general, all of the classifiers show significant differences in performance from iteration 0 to iteration 1. Note that iteration 0 is the classifier trained with the base dataset. Most of the differences represent improvements over the initial classifiers, especially in terms of the number of FA and %C. This makes sense since we are improving the negative training dataset. The trade off is a relatively small decrease in terms of %H. The improvement in terms of AVG(%C) from the initial iteration to iteration 1 ranges from 11.3%, for *Majority Unequal*, to 13.3%, observed in the *External Unequal* setup. From there on the improvements in terms of AVG(%C) are smaller from iteration to iteration. Nevertheless, comparing the initial iteration with the final one reveals improvements ranging from 14.5% (*Majority Unequal*) up to 17.0% (*External RMSE*).

Next we will focus on the analysis of the performance obtained in each of the test datasets. In the Flickr dataset (composed only of negative instances) where only FA may occur, we see a significant decrease of FA along the iterations for all setups. The average number of FA observed in the baseline (i.e., iteration 0), 1326.4, is reduced to an average value of 135.5 in the case of *External RMSE* and to 244.2 in the case of the *Majority Unequal*. This results in a reduction of the original number of FA by 89.8% to 81.6%. This considerable decrease in the number of FA has consequences on the %C. When comparing the initial with the last iteration we can observe improvements of average %C ranging from 24.8% (*Majority Unequal*) up to 28.8% (*External RMSE*). These results highlight the ability of the approach in reducing the number of FA, which is arguably the most significant contribution.

When considering the Feret dataset, we observe a similar behaviour in what concerns the average number of FA: while in the baseline classifier obtains an average of 224.9 FAs, in the last iteration the average number of FA ranges from 24.5 (*External Unequal*) to 57.7 (*Majority Unequal*), that is a reduction by 89.9% to 74.4% of the original number of FA. As expected, the decrease in the number of FA has a potential cost: the decrease in the number of hits (%H). The decrease in the average %H from the initial iteration to iteration 1 ranges from 0.8% (*Majority Unequal*) to 1.3% (*External Unequal*). When comparing the initial iteration to the last one we observe decreases ranging from 0.9% (*Majority Unequal*) to 1.6% (*External Unequal*). As such, for the Feret dataset, the improvements in terms of FA appear to clearly outweigh the losses in terms of %H. This is confirmed by the analysis of the variation of average %C, where we can observe improvements ranging from 13.7% (*Majority Unequal*) up to 17.5% (*External Unequal*).

The CMU dataset was the one that posed more difficulties for our approach. As previously, the number of FA and %H decrease, while %C increases, but the differences are not as noticeable as in the previous datasets. In terms of FA we observe a reduction from 91.9 in the initial iteration to a number ranging from 10.0 (*External RMSE*) to 15.8 (*Majority Unequal*) in the last, which corresponds to a reduction of 89.2% and 82.8% of the initial number of FA. However, the decrease in %H is more severe than in previous cases, ranging from 9.1% (*Majority Unequal*) up to 11.3% (*External Unequal*), when comparing the initial with the last iteration. The combined effect results in improvements of %C that range from 4.7% (*Majority RMSE*) up to 5.6% (*External RMSE*).

Since we perform multiple framework iterations it is also important to assess if it is advantageous to perform more than one. In general, a brief perusal of Table 6.5, reveals that this is the case. For instance, if we consider the AVG(%C) we can observe statistically significant differences between the classifiers of iteration 1 and iteration 9. In fact, with the exception of *Majority Unequal*, there are statistical significant differences between the results of iterations 5 and 9, meaning that for those the results indicate that 9 framework iterations yield better results than 5. In general the results concerning %H and FA reveal the same pattern.

When we analyse the behaviour for each test dataset, we observe the same pattern for Flickr and Ferret. For the CMU dataset, since the improvements are smaller, the differences are not statistically significant, and we are unable to detect a meaningful pattern.

6.1.4.3 Comparison among setups

In this subsection we focus on the analysis of the impact of the select and filtering methods in the performance of the classifiers. For that purpose we analyse the performance among all setups, for all the validation datasets, at the end of the framework process, i.e., in the last iteration (9).

Table 6.6 summarises the significant statistical differences among setups in the last framework iteration. A “o” indicates that the difference is not statistically significant, a “+” indicates that significant differences exist and that the value is better when compared with the setup displayed in the corresponding row, and “−” indicates that significant differences exist and the value is worse when compared with the setup displayed in the corresponding row.

As can be observed in Table 6.6 the statistical significant differences of performance between *Majority* and *External* occur mostly in terms of FA and %C, where *External* setups tend to consistently obtain better results. In terms of %H, almost no statistical significant differences were observed among *Majority* and *External* setups. The exception is a significant difference between *External Unequal* and *Majority RMSE* in the Feret dataset, justifiable by the higher values obtained by the *External Unequal* setups.

When comparing the filter methods, we did not observe any statistical significant difference when comparing *Majority RMSE* with *Majority Unequal*, nor when compar-

Table 6.5: Percentage of hits (%H), number of false alarms (FA), percentage of correct instances (%C) and average percentage of correct instances (AVG(%C)) for each validation dataset, along the iterations for all the framework setups.

		Flickr		Ferret			CMU			AVG(%C)
		FA	%C	%H	FA	%C	%H	FA	%C	
Majority Unequal	(o) baseline	<u>1326.4</u>	<u>65.5%</u>	<u>96.1%</u>	<u>224.9</u>	<u>75.2%</u>	<u>56.8%</u>	<u>91.1</u>	<u>40.8%</u>	<u>60.5%</u>
	1	<u>478.3</u>	<u>83.2%</u>	<u>95.3%</u>	<u>96.2</u>	<u>85.7%</u>	<u>53.1%</u>	<u>31.0</u>	<u>46.6%</u>	<u>71.8%</u>
	2	<u>347.9</u>	<u>87.1%</u>	<u>94.9%</u>	<u>71.3</u>	<u>87.8%</u>	<u>49.5%</u>	<u>21.3</u>	<u>46.4%</u>	<u>73.8%</u>
	3	<u>340.1</u>	<u>87.1%</u>	<u>94.9%</u>	<u>69.4</u>	<u>88.0%</u>	<u>49.3%</u>	<u>22.6</u>	<u>46.3%</u>	<u>73.8%</u>
	4	<u>304.0</u>	<u>88.4%</u>	<u>95.0%</u>	<u>60.4</u>	<u>88.9%</u>	<u>49.7%</u>	<u>20.7</u>	<u>45.9%</u>	<u>74.4%</u>
	5	<u>251.1</u>	<u>90.1%</u>	<u>94.8%</u>	<u>55.8</u>	<u>89.2%</u>	<u>48.9%</u>	<u>15.9</u>	<u>46.4%</u>	<u>75.2%</u>
	6	<u>244.9</u>	<u>90.4%</u>	<u>95.0%</u>	<u>60.4</u>	<u>88.8%</u>	<u>48.4%</u>	<u>15.0</u>	<u>46.8%</u>	<u>75.3%</u>
	7	<u>250.6</u>	<u>90.1%</u>	<u>94.8%</u>	<u>56.0</u>	<u>89.2%</u>	<u>48.5%</u>	<u>16.4</u>	<u>46.0%</u>	<u>75.1%</u>
	8	<u>231.4</u>	<u>90.8%</u>	<u>94.7%</u>	<u>56.8</u>	<u>89.0%</u>	<u>46.8%</u>	<u>14.3</u>	<u>45.2%</u>	<u>75.0%</u>
	9	<u>244.2</u>	<u>90.3%</u>	<u>94.7%</u>	<u>57.7</u>	<u>88.9%</u>	<u>47.7%</u>	<u>15.8</u>	<u>45.8%</u>	<u>75.0%</u>
Majority RMSE	(o) baseline	<u>1326.4</u>	<u>65.5%</u>	<u>96.1%</u>	<u>224.9</u>	<u>75.2%</u>	<u>56.8%</u>	<u>91.1</u>	<u>40.8%</u>	<u>60.5%</u>
	1	<u>396.7</u>	<u>85.6%</u>	<u>95.0%</u>	<u>80.6</u>	<u>86.8%</u>	<u>51.5%</u>	<u>25.2</u>	<u>46.2%</u>	<u>72.9%</u>
	2	<u>349.4</u>	<u>86.9%</u>	<u>94.8%</u>	<u>74.1</u>	<u>87.3%</u>	<u>50.9%</u>	<u>24.5</u>	<u>46.0%</u>	<u>73.4%</u>
	3	<u>340.1</u>	<u>87.1%</u>	<u>94.9%</u>	<u>66.0</u>	<u>88.3%</u>	<u>50.5%</u>	<u>22.0</u>	<u>46.8%</u>	<u>74.1%</u>
	4	<u>277.3</u>	<u>89.2%</u>	<u>94.5%</u>	<u>60.5</u>	<u>88.5%</u>	<u>48.6%</u>	<u>17.6</u>	<u>45.4%</u>	<u>74.3%</u>
	5	<u>268.0</u>	<u>89.5%</u>	<u>94.7%</u>	<u>63.4</u>	<u>88.4%</u>	<u>49.0%</u>	<u>17.2</u>	<u>45.8%</u>	<u>74.6%</u>
	6	<u>232.3</u>	<u>90.7%</u>	<u>94.7%</u>	<u>51.6</u>	<u>89.5%</u>	<u>48.1%</u>	<u>15.4</u>	<u>46.0%</u>	<u>75.4%</u>
	7	<u>197.3</u>	<u>92.0%</u>	<u>94.8%</u>	<u>44.4</u>	<u>90.3%</u>	<u>47.2%</u>	<u>13.0</u>	<u>45.5%</u>	<u>75.9%</u>
	8	<u>222.4</u>	<u>91.1%</u>	<u>94.8%</u>	<u>45.5</u>	<u>90.2%</u>	<u>47.4%</u>	<u>14.6</u>	<u>45.5%</u>	<u>75.6%</u>
	9	<u>185.4</u>	<u>92.4%</u>	<u>94.5%</u>	<u>44.6</u>	<u>90.1%</u>	<u>46.2%</u>	<u>11.9</u>	<u>45.5%</u>	<u>76.0%</u>
External Unequal	(o) baseline	<u>1326.4</u>	<u>65.5%</u>	<u>96.1%</u>	<u>224.9</u>	<u>75.2%</u>	<u>56.8%</u>	<u>91.1</u>	<u>40.8%</u>	<u>60.5%</u>
	1	<u>327.5</u>	<u>87.7%</u>	<u>94.8%</u>	<u>73.8</u>	<u>87.4%</u>	<u>50.6%</u>	<u>21.4</u>	<u>46.3%</u>	<u>73.8%</u>
	2	<u>340.8</u>	<u>87.1%</u>	<u>95.4%</u>	<u>67.3</u>	<u>88.6%</u>	<u>51.1%</u>	<u>22.3</u>	<u>46.8%</u>	<u>74.2%</u>
	3	<u>239.7</u>	<u>90.4%</u>	<u>94.7%</u>	<u>52.2</u>	<u>89.5%</u>	<u>48.0%</u>	<u>15.8</u>	<u>45.4%</u>	<u>75.1%</u>
	4	<u>219.3</u>	<u>91.0%</u>	<u>95.0%</u>	<u>46.1</u>	<u>90.2%</u>	<u>47.8%</u>	<u>14.7</u>	<u>46.5%</u>	<u>75.9%</u>
	5	<u>229.6</u>	<u>90.7%</u>	<u>95.5%</u>	<u>40.2</u>	<u>91.5%</u>	<u>48.7%</u>	<u>16.2</u>	<u>46.0%</u>	<u>76.1%</u>
	6	<u>197.3</u>	<u>91.8%</u>	<u>95.3%</u>	<u>33.6</u>	<u>91.9%</u>	<u>47.5%</u>	<u>13.1</u>	<u>46.6%</u>	<u>76.8%</u>
	7	<u>174.8</u>	<u>92.7%</u>	<u>95.2%</u>	<u>30.7</u>	<u>92.1%</u>	<u>46.9%</u>	<u>13.2</u>	<u>46.5%</u>	<u>77.1%</u>
	8	<u>143.7</u>	<u>93.9%</u>	<u>95.2%</u>	<u>23.7</u>	<u>92.8%</u>	<u>45.4%</u>	<u>9.7</u>	<u>45.5%</u>	<u>77.4%</u>
	9	<u>148.6</u>	<u>93.7%</u>	<u>95.2%</u>	<u>24.5</u>	<u>92.7%</u>	<u>45.5%</u>	<u>10.8</u>	<u>45.6%</u>	<u>77.3%</u>
External RMSE	(o) baseline	<u>1326.4</u>	<u>65.5%</u>	<u>96.1%</u>	<u>224.9</u>	<u>75.2%</u>	<u>56.8%</u>	<u>91.1</u>	<u>40.8%</u>	<u>60.5%</u>
	1	<u>352.7</u>	<u>86.7%</u>	<u>95.2%</u>	<u>79.3</u>	<u>87.2%</u>	<u>51.2%</u>	<u>23.6</u>	<u>46.5%</u>	<u>73.5%</u>
	2	<u>293.1</u>	<u>88.8%</u>	<u>95.1%</u>	<u>62.2</u>	<u>89.0%</u>	<u>50.8%</u>	<u>19.4</u>	<u>46.5%</u>	<u>74.7%</u>
	3	<u>225.8</u>	<u>90.9%</u>	<u>94.6%</u>	<u>49.6</u>	<u>89.6%</u>	<u>48.7%</u>	<u>15.7</u>	<u>46.0%</u>	<u>75.5%</u>
	4	<u>203.8</u>	<u>91.7%</u>	<u>94.9%</u>	<u>45.9</u>	<u>90.3%</u>	<u>48.3%</u>	<u>14.7</u>	<u>46.3%</u>	<u>76.1%</u>
	5	<u>177.2</u>	<u>92.8%</u>	<u>94.9%</u>	<u>37.5</u>	<u>91.1%</u>	<u>46.9%</u>	<u>12.3</u>	<u>45.8%</u>	<u>76.6%</u>
	6	<u>169.7</u>	<u>92.9%</u>	<u>95.0%</u>	<u>32.8</u>	<u>91.7%</u>	<u>47.1%</u>	<u>11.2</u>	<u>46.1%</u>	<u>76.9%</u>
	7	<u>155.4</u>	<u>93.4%</u>	<u>94.7%</u>	<u>32.1</u>	<u>91.4%</u>	<u>46.3%</u>	<u>11.5</u>	<u>46.0%</u>	<u>77.0%</u>
	8	<u>137.7</u>	<u>94.1%</u>	<u>94.6%</u>	<u>27.7</u>	<u>91.8%</u>	<u>45.6%</u>	<u>9.8</u>	<u>45.3%</u>	<u>77.1%</u>
	9	<u>135.5</u>	<u>94.3%</u>	<u>94.7%</u>	<u>27.8</u>	<u>91.9%</u>	<u>45.9%</u>	<u>10.0</u>	<u>46.4%</u>	<u>77.5%</u>

Table 6.6: Statistical significant differences of: %H - percentage of Hits, FA - number of false alarms, %C - percentage of correct instances; among the framework setups in the last iteration for each test dataset: 1 - Flickr; 2 - Feret; 3 - CMU.

Measure	Setup		Majority						External					
			Unequal			RMSE			Unequal			RMSE		
			Dataset											
			1	2	3	1	2	3	1	2	3	1	2	3
%H	Majority	Unequal	n.a.	n.a.	n.a.	n.a.	o	o	n.a.	o	o	n.a.	o	o
		RMSE	n.a.	o	o	n.a.	n.a.	n.a.	n.a.	+	o	n.a.	o	o
	External	Unequal	n.a.	o	o	n.a.	-	o	n.a.	n.a.	n.a.	n.a.	o	o
		RMSE	n.a.	o	o	n.a.	o	o	n.a.	o	o	n.a.	n.a.	n.a.
FA	Majority	Unequal	n.a.	n.a.	n.a.	o	o	o	+	+	o	+	+	+
		RMSE	o	o	o	n.a.	n.a.	n.a.	o	+	o	+	+	o
	External	Unequal	-	-	o	o	-	o	n.a.	n.a.	n.a.	o	o	o
		RMSE	-	-	-	-	-	o	o	o	o	n.a.	n.a.	n.a.
%C	Majority	Unequal	n.a.	n.a.	n.a.	o	o	o	+	+	o	+	+	o
		RMSE	o	o	o	n.a.	n.a.	n.a.	o	+	o	+	+	o
	External	Unequal	-	-	o	o	-	o	n.a.	n.a.	n.a.	o	o	o
		RMSE	-	-	o	-	-	o	o	o	o	n.a.	n.a.	n.a.

ing *External RMSE* with *External Unequal*. This indicates that the observed differences among setups are most likely due to the selection methods and that, in the considered experimental conditions, none of the considered filtering methods offers a competitive advantage over the other.

6.1.5 Adversarial versus Non-Adversarial Augmentation

In this subsection we focus on analysing if there is an advantage in evolving instances that explore shortcomings, i. e. adversarial instances, and using them for training purposes, as we do in our framework, or if these results could also been obtained by the addition of non-evolved, and hence not adversarial, negative instances.

In order to do this, we trained the same number of classifiers as we did for all the framework setups, with the same number of negative instances per iteration (see Table 6.7), but without including evolved instances. Instead, we employ the DA technique proposed by Sung and Poggio (1995) to increase the number of negative instances by randomly selecting parts of non-face images, classifying them, and adding the misclassified parts to the training dataset. This is a main approach used when training Haar cascade classifiers and is included by default in OpenCV.

We refer to these setups as *Control*. After training all *Control* classifiers we tested them with the same test datasets. Table 6.7 summarises the results obtained by the *Control* classifiers.

Table 6.7: Percentage of hits (%H), number of false alarms (FA), percentage of correct examples (%C) and average percentage of correct examples (AVG(%C)) for each validation dataset, along the iterations for all the *Control* setups.

		Flickr		Feret			CMU			AVG(%C)
		FA	%C	%H	FA	%C	%H	FA	%C	
Control Majority Unequal	(o) baseline	<u>1326.4</u>	<u>65.5%</u>	<u>96.1%</u>	<u>224.9</u>	<u>75.2%</u>	<u>56.8%</u>	<u>91.1</u>	40.8%	<u>60.5%</u>
	1	802.9	76.1%	95.5%	147.0	81.1%	54.2%	53.2	43.7%	66.9%
	2	772.9	76.8%	95.5%	140.6	81.7%	53.8%	47.3	44.8%	67.8%
	3	782.8	76.7%	95.5%	148.2	80.9%	53.8%	50.1	43.4%	67.0%
	4	738.1	77.5%	95.4%	135.2	81.9%	53.6%	45.6	44.8%	68.1%
	5	717.5	78.1%	95.4%	131.1	82.5%	53.5%	45.6	44.0%	68.2%
	6	751.4	77.1%	95.4%	140.5	81.6%	54.1%	49.3	43.9%	67.6%
	7	639.8	79.7%	95.5%	124.8	83.0%	53.1%	43.7	43.6%	68.8%
	8	713.2	78.2%	95.4%	140.3	81.5%	52.9%	47.0	43.3%	67.7%
	9	724.7	77.8%	95.5%	128.1	82.8%	53.3%	46.0	44.4%	68.3%
Control Majority RMSE	(o) baseline	<u>1326.4</u>	<u>65.5%</u>	<u>96.1%</u>	<u>224.9</u>	<u>75.2%</u>	<u>56.8%</u>	<u>91.1</u>	40.8%	<u>60.5%</u>
	1	712.1	78.2%	95.4%	126.5	82.8%	53.7%	46.3	44.1%	68.3%
	2	746.7	77.2%	95.5%	133.2	82.3%	53.7%	48.4	44.7%	68.1%
	3	649.9	79.7%	95.3%	122.8	83.0%	53.4%	42.1	44.8%	69.2%
	4	812.6	75.5%	95.6%	149.2	80.9%	54.3%	52.1	44.3%	66.9%
	5	781.7	76.6%	95.3%	137.3	82.0%	54.3%	49.2	44.2%	67.6%
	6	778.0	76.6%	95.5%	143.5	81.4%	54.2%	48.3	43.9%	67.3%
	7	884.8	74.4%	95.6%	152.2	80.7%	54.8%	55.6	43.5%	66.2%
	8	711.4	78.3%	95.3%	129.0	82.4%	53.9%	46.8	44.2%	68.3%
	9	748.8	77.4%	95.5%	131.8	82.3%	53.4%	46.2	43.6%	67.8%
Control External Unequal	(o) baseline	<u>1326.4</u>	<u>65.5%</u>	<u>96.1%</u>	<u>224.9</u>	<u>75.2%</u>	<u>56.8%</u>	<u>91.1</u>	40.8%	<u>60.5%</u>
	1	744.4	77.4%	95.4%	141.2	81.4%	54.1%	48.2	44.1%	67.6%
	2	710.8	78.3%	95.4%	134.2	82.0%	52.9%	43.8	44.3%	68.2%
	3	760.4	77.1%	95.5%	136.7	82.1%	53.7%	48.1	44.2%	67.8%
	4	682.2	78.7%	95.5%	120.8	83.4%	53.6%	45.0	44.9%	69.0%
	5	835.2	75.4%	95.6%	165.5	79.6%	54.6%	55.4	44.6%	66.5%
	6	769.4	76.7%	95.6%	140.1	81.8%	54.7%	51.3	45.1%	67.9%
	7	774.7	76.7%	95.7%	144.7	81.5%	54.6%	48.8	44.9%	67.7%
	8	803.3	76.2%	95.6%	146.2	81.1%	54.0%	49.8	44.3%	67.2%
	9	754.7	77.1%	95.6%	135.3	82.2%	53.6%	48.8	43.9%	67.8%
Control External RMSE	(o) baseline	<u>1326.4</u>	<u>65.5%</u>	<u>96.1%</u>	<u>224.9</u>	<u>75.2%</u>	<u>56.8%</u>	<u>91.1</u>	40.8%	<u>60.5%</u>
	1	751.8	77.1%	95.4%	141.1	81.5%	53.8%	50.0	44.5%	67.7%
	2	801.0	76.1%	95.4%	149.0	80.8%	54.0%	53.0	43.8%	66.9%
	3	730.2	77.7%	95.5%	132.6	82.4%	53.9%	47.5	44.8%	68.3%
	4	763.4	76.9%	95.4%	142.9	81.3%	53.2%	48.4	43.7%	67.3%
	5	747.8	77.6%	95.4%	140.0	81.5%	53.6%	49.2	44.6%	67.9%
	6	747.6	77.3%	95.5%	131.6	82.4%	54.1%	47.7	44.7%	68.1%
	7	771.1	76.7%	95.6%	146.2	81.2%	53.9%	50.2	44.3%	67.4%
	8	708.9	78.3%	95.4%	128.2	82.6%	54.0%	45.0	45.2%	68.7%
	9	776.3	76.7%	95.6%	142.1	81.5%	53.6%	50.0	44.1%	67.4%

As previously we indicate statistical significant differences with the previous iteration using *italic*, and with the last using underlined. In addition, **bold** indicates statistical significant differences between the results obtained in one iteration by the classifiers of our framework confronted against the result of the *Control* classifiers in the same iteration.

One can observe that there are statistical significant differences between the classifiers of the initial iteration and the ones of iteration 1 for all considered metrics, with the exception of %C in the CMU dataset. The same occurs when comparing the initial iteration with the last one. Interestingly, from the initial iteration onwards, there are no statistical significant differences between any other iteration with the results of the last iteration. Thus, while it is advantageous for **EFFECTIVE** to perform more than one iteration, when using the control setups no significant further improvements of performance are gained after iteration 1. Thus, while the **EC** runs are able to systematically create novel adversarial instances that contribute to the improvement of the performance of the classifiers, the advantages of non-adversarial instances, by selecting and adding misclassified parts of non-face images, are exhausted when a sufficiently large number of such image parts is included in the training dataset. Depending on the setup, the increases of AVG(%C) from the initial to the last iteration range from 6.9% *Control External RMSE* to 7.8% *Control Majority Unequal*.

More importantly, the comparison of the results obtained with the *Control* setups with those obtained using our framework, reveals the advantages of **EFFECTIVE**. As it can be observed the differences in terms of AVG(%C) are always statistically significant. For the Flickr dataset, statistical significant differences exist both in terms of FA and %C. The same happens for the Feret dataset. However, in this case, one can also observe some statistical significant differences in terms of %H, where the framework setups tend to perform worse (although statistically significant, the differences in %H range from 0.1% to 1.1%). It is interesting to notice that, in most cases, the *External Unequal* setup appears to achieve an improvement of FA without incurring in statistical significant losses of performance in terms of %H.

On the downside, the results regarding the CMU dataset tend to be inconclusive. That is, our approach tends to perform significantly worse in terms of %H and significantly better in terms of FA. In terms of %C, although the values obtained by our approach tend to be superior, the differences are not statistical significant in most cases. This result was, to some extent, expected since it was the dataset in which our framework showed the least improvements over the original classifiers. Overall, we consider that these results demonstrate the advantages of our approach over a standard technique for **DA** (i.e. based on the work of Sung and Poggio, 1995).

Table 6.8 summarises the significant statistical differences among framework and control experiments at the end of the framework process, i.e., in the last iteration (9). As previously: a “o” indicates that the difference is not statistically significant; a “+” indicates that the result obtained by the framework is better and the difference is

Table 6.8: Statistical significant differences of: %H - percentage of Hits, FA - number of false alarms, %C - percentage of correct examples; between the framework and control setups in the last iteration for each validation dataset: 1 - Flickr; 2 - Feret; 3 - CMU.

	Measure	Setup	Framework Setups													
			Majority						External							
			Unequal			RMSE			Unequal			RMSE				
			Dataset													
			1	2	3	1	2	3	1	2	3	1	2	3		
Control Setups	H	Majority	Unequal	n.a.	-	-	n.a.	-	-	n.a.	o	-	n.a.	-	-	
			RMSE	n.a.	-	-	n.a.	-	-	n.a.	o	-	n.a.	-	-	
		External	Unequal	n.a.	-	-	n.a.	-	-	n.a.	o	-	n.a.	-	-	
			RMSE	n.a.	-	-	n.a.	-	-	n.a.	o	-	n.a.	-	-	
	FA	Majority	Unequal	+	+	+	+	+	+	+	+	+	+	+	+	
			RMSE	+	+	+	+	+	+	+	+	+	+	+	+	
		External	Unequal	+	+	+	+	+	+	+	+	+	+	+	+	+
			RMSE	+	+	+	+	+	+	+	+	+	+	+	+	+
	C	Majority	Unequal	+	+	o	+	+	o	+	+	o	+	+	+	+
			RMSE	+	+	o	+	+	o	+	+	o	+	+	+	+
		External	Unequal	+	+	o	+	+	o	+	+	o	+	+	o	+
			RMSE	+	+	o	+	+	o	+	+	o	+	+	+	+

statistically significant; a “-” indicates that the result obtained by the framework is worse and the difference is statistically significant. This summary further reinforces the previous analysis, highlighting the differences between methods.

6.2 EXPANDING THE POSITIVE DATASET

In the previous Section we explored the expansion of the negative dataset. We instantiated **EFFECTIVE** to evolve instances classified as false positives, performed multiple framework iterations and analysed the results of the evolutionary process and the performance of the classifier. Based on the results in test dataset, lower values of FA than the baseline classifier were attained by the framework classifiers, which resulted in an increase of the classifiers’ overall performance but at the cost of %H. In this Section, we test the **EC** engine described in Section 4.5 to generate instances classified as false negatives and use them to expand the positive dataset. We are interested in testing if the expansion of the positive dataset using **EFFECTIVE** increases the performance of the classifier in terms of %H.

Table 6.9: Framework parameters.

Parameter	Setting
Framework iterations (N)	10
Classifiers per iteration (C)	30
EC runs per classifier (E)	1
Minimum candidates per EC run	50
RMSE threshold	0.10

6.2.1 Framework Changes

The **CS** module was maintained for this set of experiments and can be consulted in Subection 4.1.1. In this case, the objective is to evolve false negatives. Therefore, in this experiment, candidate instances are images that are not classified as faces, but that should have been classified as faces. To fulfil this requirement we resort to the **EC** engine that generates photorealistic faces, described in Section 4.5. The Supervisor module is the *External RMSE*. The choice was made based on the overall results attained in Section 6.1. The Supervisor's classifier was maintained. In contrast with the approach of Section 6.1, the Supervisor selects instances that are classified as faces.

6.2.2 Experimental Setup

The overall framework parameters are presented in Table 6.14, which yields 300 distinct Haar cascade classifiers. The dataset used is the same as described in Section 5.2 (1905 images from different sources). This contrasts with the experiment of Section 4.5 that uses a different positive dataset (composed of either 200 or 500 faces from FACITY).

Table 6.10: **EC** parameters.

Parameter	Setting
Number of generations	50
Population size	100
Elite size	1
Tournament size	2
Crossover operator	uniform crossover
Crossover rate	0.8
Mutation operator	gene replacement
Mutation rate per gene	0.15

In the experiments for expanding the negative dataset (Section 6.1) an expression based **GP** engine is used (see Subsection 4.1.2.1). The representation of the **GP** engine

is more flexible, easing the process of generating suitable individuals for the problem. On the other hand, with the eXploit-Faces **EC** engine representation adapted to the problem of generating false negatives, we are more restricted and it becomes harder to generate candidate instances. The settings for the **GA** engine for each **EC** run are presented in Table 6.10. These are based on the results obtained in Section 4.5.2. An adjustment was made to the population size. The adjustment was made in order to increase the number of individuals of the population so it becomes easier to reach the minimum candidates per **EC** run quota. These adjustments were determined empirically. The fitness function was maintained from the previous experiment and is described in Section 4.5.2.

6.2.3 Experimental Results

First, we start by describing the results of the evolutionary process. We show that it is possible to guide evolution with the current fitness scheme and parameters, and that the results change from one framework iteration to the next. We start by analysing the evolutionary process for each framework iteration. Afterwards, we discuss the assessment of the classifiers, by performing a statistical analysis of the results of each setup.

Similar to the experiments presented in Section 6.1, we rely on statistical tests to make inferences about the experimental results. First we apply the Lilliefors test (Lilliefors, 1967), then we use the Kruskal-Wallis between the groups to assess if statistically significant differences exist. When differences exist, we rely on post-hoc tests, using Wilcoxon-Mann-Whitney test (Wilcoxon, 1945) to perform pairwise comparisons. All the tests are performed with a confidence level of 99%, i.e., the p-value for rejecting the null hypothesis has to be less than 0.01. We use the Bonferroni correction ($\alpha/\text{textrnumberofcomparisons}$) in the cases where multiple comparisons were performed (Bonferroni, 1935).

The training phase is the same of the Section 6.1.3.1. At each iteration of the framework, 30 distinct classifiers are trained. A single **EC** run is performed per classifier, which results in 30 distinct **EC** runs. Each **EC** run evaluates the individuals with a different classifier to assign fitness. In this instantiation the **EC** run must generate 50 candidate instances, in order for an **EC** run to be valid. In this experiment, a candidate instance is an instance classified as not containing a face. The Supervisor selects and filters the candidate instances and the resulting set of instances are added to the training dataset completing a framework iteration.

Figure 6.6 shows the evolution of the fitness of the best individual of each generation (Max) and of the average fitness (Avg), during the initial framework iteration. To improve the readability, the fitness values were normalised to the $[0, 1]$ interval by

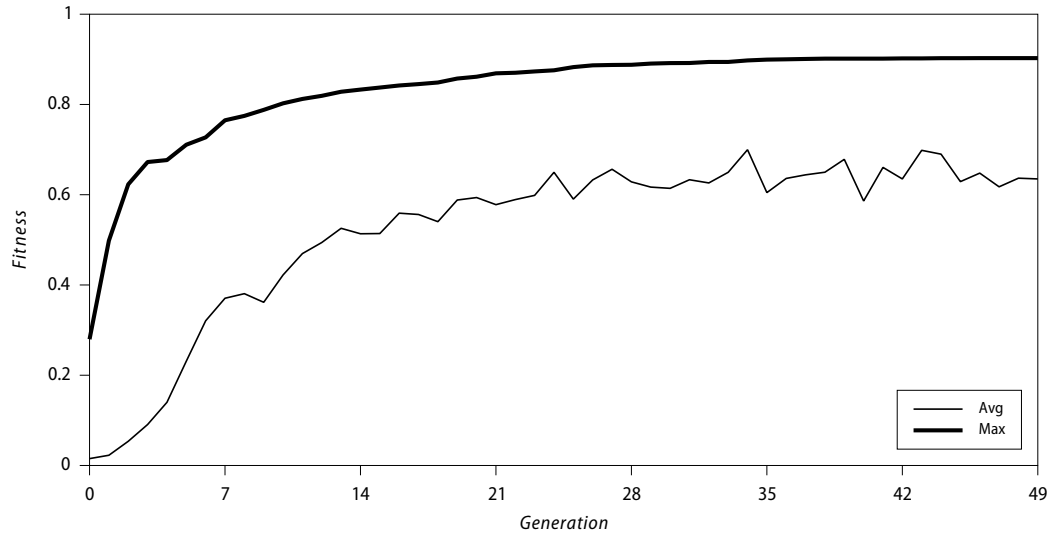


Figure 6.6: Evolution of the average and maximum fitness across generations for the initial framework iteration, considering the valid the [EC](#) runs. The results are averages of 30 independent runs.

dividing the raw fitness, calculated according to the fitness function by the average maximum fitness found in the course of the framework iteration³.

The fitness increases rapidly in the first 7 generations, and steadily, although at a lower pace, from there on. This is an indication that the fitness function is capable of guiding the evolutionary process, leading it to the discovery of images classified as members of the negative class.

Figure [6.7](#) shows the evolution of the fitness of the best individual of each generation (Max) and of the average fitness for the final framework iteration for the valid [EC](#) runs. When comparing the maximum and average fitness curves of both iterations we can observe that the curves grow at different rates. In the initial iteration we see a fast increase in the first 10 generations and a slower rate after, stabilising after the 30th generation. The average fitness of the population per generation reaches high values (0.7) close to the average maximum of the 30 runs (0.9). In the last iteration the [EC](#) struggles to get to the maximum value, with small increases along the generations and unable to reach the maximum value reached in the initial iterations. The charts indicate that becomes harder to the [EC](#) engine to find instances classified as false negatives as the number of iterations increases.

To analyse the impact of the addition of the evolved instance to the dataset, we present in Table [6.11](#), for each setup and for each iteration, the total number of [EC](#) runs

³ Similar to the experiments for expanding the negative dataset, it is important to notice that the raw fitness values resulting from two different classifiers are not directly comparable, since the number of stages and inner structure tends to be different.

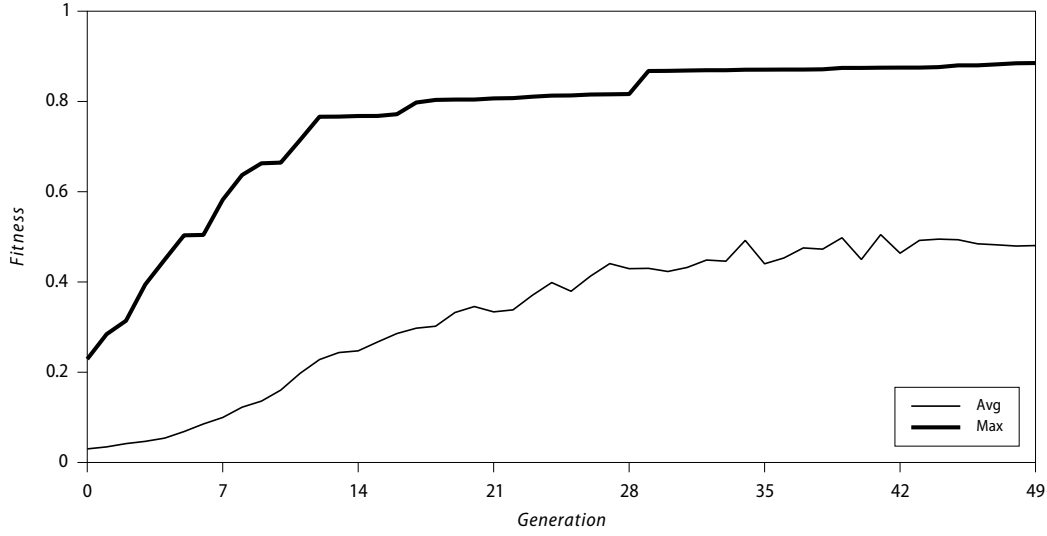


Figure 6.7: Evolution of the average and maximum fitness across generations for the last framework iteration, considering the valid the EC runs. The results are averages of 30 independent runs.

performed in order to reach 30 valid runs, the average number of detections per EC run, and the average number of detection per valid EC run.

For the analysis of the evolution of the total number of EC runs per framework iteration, we performed a non-parametric chi-squared method for statistical proportion analysis. This analysis reveals that there are statistically significant differences⁴ between the proportion of valid EC runs observed during the initial iteration, and all remaining ones. Statistically significant differences between the proportion of valid EC runs of a given iteration and the previous one are indicated in *italic*. Statistically significant differences between one iteration and the last one are depicted in underline. The differences from iteration per iteration only exists in iteration 2 (from 31 to 47) but stabilises from there on. Nevertheless it suffers some fluctuations in terms of values. It seems that it does not need several EC runs to find the necessary number of candidate instances. As for the statistical differences between one iteration and the last iteration only in the first two framework iterations occurs. This copes with the results attained with the charts.

As for the analysis of the number of candidate instances, we perform the Wilcoxon-Mann-Whitney test with a confidence level of 99% with Bonferroni correction for multiple tests, which implies setting significance level to $0.01/8$ for the tests involving comparisons with iteration 8, and $0.01/2$ for all the remaining ones. The results concerning the number of candidate instances follow the previous analysis. The differences from consecutive iterations only occur until iteration 2. The differences from a iteration with

⁴ with a confidence level of 99%

Table 6.11: Number of EC runs necessary to find 30 valid runs, average and median number of candidate instances per EC run, and average and median number of detections per valid EC run.

Setup	Iter.	EC runs	Avg candidates all runs	Median candidates all runs	Avg candidates valid runs	Median candidates valid runs
Expanding Positive	0	30	<u>3382.50</u>	<u>3583.50</u>	<u>3382.50</u>	<u>3583.50</u>
	1	31	<u>1922.68</u>	<u>2055.00</u>	1986.77	2055.50
	2	47	1097.60	1106.00	1716.90	1807.00
	3	34	1484.65	1716.50	1682.60	1825.00
	4	39	1206.28	1392.00	1566.67	1645.50
	5	39	1219.62	1275.00	1585.50	1643.00
	6	42	1193.79	1141.00	1669.60	1695.50
	7	39	1350.64	1245.00	1755.83	1693.00
	8	49	925.76	747.00	1512.07	1572.50

the last iteration also hold a similar behaviour for the results with all EC runs. In the valid EC runs only in iteration 0 we have a statistically significant difference.

Figure 6.8 shows an example from the last population from each CS model's EC run. Interestingly we see almost the same faces with different facial attributes. Since in the initial iterations the datasets of the classifiers do not have these examples, this could mean that these faces are uncommon on the base dataset yielding a higher fitness. In contrast, Figure 6.17 shows a more diversified sample of faces. In this iteration it seems that the classifier struggles with some lighting conditions (e. g., second on the second row of Figure 6.17). We have more atypical combinations, using babies and mixing woman eyebrows with man faces (first on the second row of Figure 6.8). Based on these image results, fitness and detection, we can say that the classifiers are still able to find instances that are not classified as faces.

In Table 6.12 the dataset's size and the number of instances added to the positive dataset throughout the iterations are presented. The results reinforce the observations performed for the EC runs analysis and fitness charts. In the first iteration the highest number of instances are added to the dataset when compared with the rest of the iterations. This can be explained by the fact that the dataset used by the EC engine to generate instances, contains examples that differ from the instances of the baseline classifier's dataset. After adding the instances of the first iteration, it becomes harder for the EC engine to find instances that the classifier does not classify as faces. We can also observe that, based on the average number of candidates per EC run (Table 6.11), the Supervisor, with a *External RMSE* strategy, tends to filter several of the generated



Figure 6.8: Instances collected by the *External RMSE* Supervisor of the last population of each classifier in the first iteration, not considered to be faces.



Figure 6.9: Instances collected by the supervisor representative of the last population of each classifier in the last iteration, not considered to be faces by the classifier.

Table 6.12: Positive dataset's size along the iterations for all setups.

Iteration	Expanding Positive	
	size	added
0	1905	-
1	8580	6675
2	9368	788
3	10039	671
4	10673	634
5	10896	223
6	11392	496
7	11626	234
8	11969	343
9	12415	446

instances. As seen in Figure 6.8 and Figure 6.17 one of the reasons for such instances to be filtered is their similarity.

The trained classifiers are used in the test datasets to assess the performance. We group the models per iteration and analyse the differences for each performance metric. The Supervisor's classifier is also included for analysis. The summary of the results obtained in the test datasets are depicted in Table 6.13. As previously, to determine if statistical significant differences occur we use the Wilcoxon-Mann-Whitney test with a confidence level of 99% with Bonferroni correction for multiple comparisons. This implies setting the significance level to 0.01/11 for the tests involving comparisons with iteration 9, and 0.01/2 for the comparison of consecutive iterations. In Table 6.13 underlined values indicate that there is a statistically difference between the value and the corresponding one observed in the last iteration (9) of each corresponding setup, those in *italic* values indicate that there is a statistically difference between the value and the corresponding one observed with the previous iteration. We include the AVG (%C) which represents the average %C for the considered test datasets.

Overall, there are significant differences in the %H from iteration 0 to iteration 1. The differences to the last iteration exist for all the metrics of iteration 0 in the Feret test dataset and for %H of iteration 1 of the CMU. Since we are improving the positive dataset, the results make sense. Although there are no statistically significant differences, for all the test datasets, in the last framework iteration the %C is higher than the baseline in all test datasets except the CMU (-0.3 difference). Compared with the Supervisor's classifier, only in the Flickr the result is lower. In terms of AVG(%C) this experiment results show a higher and statistically significant result when compared

Table 6.13: Percentage of hits (%H), number of false alarms (FA), percentage of correct instances (%C) and average percentage of correct instances (AVG(%C)) for each validation dataset, along the iterations for all the framework setups.

Setup	Iteration	Flickr		Feret			CMU			AVG(%C)
		FA	%C	%H	FA	%C	%H	FA	%C	
Expanding Positives	baseline(o)	1326.4	65.5%	<u>96.1%</u>	<u>224.9</u>	<u>75.2%</u>	56.8%	91.1	40.8%	<u>60.5%</u>
	1	1249.9	66.6%	96.7%	160.7	81.1%	<u>52.9%</u>	92.6	39.3%	62.3%
	2	1329.2	65.2%	97.0%	149.3	82.4%	54.3%	95.6	40.2%	62.6%
	3	1248.9	66.5%	97.3%	138.4	83.7%	53.8%	91.2	40.2%	63.5%
	4	1370.6	64.6%	97.2%	151.9	82.5%	54.4%	99.6	39.0%	62.0%
	5	1312.9	65.5%	97.3%	145.6	83.2%	53.9%	96.6	38.8%	62.5%
	6	1279.1	66.0%	97.4%	136.7	83.9%	53.8%	94.7	40.1%	63.3%
	7	1311.7	65.5%	97.3%	143.0	83.4%	54.2%	95.2	39.6%	62.9%
	8	1299.4	66.1%	97.3%	140.0	83.5%	54.5%	94.3	40.9%	63.5%
	9	1197.1	67.7%	97.1%	129.8	84.5%	53.5%	89.0	40.5%	64.2%
	Supervisor	401.0	83.9%	42.7%	521.0	41.9%	31.5%	244.0	21.5%	49.1%

to the baseline. The overall results suggest that using **EFFECTIVE** with this **EC** engine we are able to increase the %H.

Next we will focus on the analysis of the performance obtained in each of the test datasets. In the Flickr dataset, although not statistically significant, a decrease of 129.3 in terms of FAs occurs from iteration 0 to iteration 9, resulting on an increase of %C of 2.2%. This behaviour is explained by the pressure applied to the FA rate during training of the classifier (see Section 4.1.1). That is, we are adding more positive instances to the dataset while maintaining the number of negative examples, the minimum FA rate and %H rate. At each iteration, during training the classifier must detect more instances as faces while maintaining the same number of FAs which indirectly forces the classifier to be more robust. Nevertheless is not a difference that competes with the the Supervisor's classifier, which obtains a better result with a difference of 16.2.

In the Feret dataset we observe an increase in terms of %H and a decrease of FAs. The increase of %H, ranging from 0.5% to 1.3% along the iterations, is explained by the addition of the evolved instance to the positive dataset. The total decrease that ranges from 64.2 to 95.1 in terms of FA, follows the same behaviour that occurred in the Flickr test dataset. The values of %H and FA resulted in an overall, and statistically significant, increase of 9.3% of %C when comparing the last iteration with the baseline. When compared with the Supervisor's classifier, the result of the framework's classifiers is higher in terms of %C, ranging from 39.2% (Iteration 1) to 42.6% (Iteration 9).

The CMU continues to be a challenging dataset. When compared with the Supervisor's classifiers, the framework's classifiers obtains a better result in every iteration, resulting in differences that range from 17.8% to 19% in terms of %C. The only statisti-

cal difference for %H is obtained in iteration 1 but the value is lower than the baseline classifier. Only iteration 8 holds a better result in terms of %C by 0.1% but not statistically significant. We see variations in %H along the iterations, from 52.9 to 54.5 but always lower than the baseline %H. One explanation for this behaviour is the following: the face crop box of our instances differ from the ones of the CMU's groundtruth. I. e., the preparation of the dataset positive dataset was manually performed by us and the CMU dataset was already annotated by the creators of the dataset (see Section 5.2.0.1). Since we are obtaining programmatically the new instances, the crop of the face in the image is also automatic and tends to follow the manual cropping performed by us in the preparation of the dataset. This can affect the positioning of the face detection box, resulting on missing the groundtruth hit, creating an FA and resulting in incorrect classified instance. This behaviour is less noticeable for the Feret because the examples were manually cropped and thus, the box has less probability to be misplaced.

6.3 EXPANDING THE POSITIVE AND THE NEGATIVE DATASETS

After performing the experiments of Sections 6.1 and 6.2 we show that we are able to expand the negative and positive dataset, respectively. In the experiments for expanding the negative dataset, the classifiers trained with the expanded negative dataset showed a decrease in the number FAs in the test datasets. When expanding the positive dataset, the classifiers trained with the expanded positive dataset revealed an increase in the %H in the test datasets. In this Section, we are going to perform a new set of experiments where we expand both datasets per iteration. The idea is to test if we are able to train classifiers that when tested in the test datasets will show a decrease in the number of FAs and, at the same time, an increase the %H. We start by defining the framework changes for this set of experiments in Section 6.3.1. Then, we present the experimental setup in Section 6.3.2. Finally, we present the assessment results in Section 6.3.3.

6.3.1 Framework Changes

We use the same EC described in Section 4.1.2.2 engine to evolve false negative instances. However we use NORBERT as the EC engine to evolve false positive instances. Based on the results attained in Section 6.1 we use *External RMSE* as the Supervisor to filter and select instances from the two distinct EC engines. In this instantiation for each iteration, EC runs from both engines are performed. Each EC engine has a Supervisor that selects and filters the instances that are going to be added to the dataset. In this way, a framework iteration is completed when the EC runs from both engines end and when the corresponding Supervisors updated each corresponding dataset with the set of instances.

Table 6.14: Framework parameters.

Parameter	Setting
Framework iterations (N)	10
Classifiers per iteration (C)	30
EC runs per classifier (E)	1
Minimum candidates per EC run	300 (<i>Abstract</i>) and 50 (<i>PhotoRealistic</i>)
RMSE threshold	0.10

Table 6.15: *Abstract* engine parameters.

Parameter	Setting
Population Size	100
Number of generations	50
Crossover probability	0.8
Mutation operators	sub-tree swap, sub-tree replacement, node insertion, node deletion, node mutation
Initialisation method	ramped half-and-half
Initial maximum depth	5
Mutation max tree depth	3
Function set	+, -, *, /, min, max, abs, sin, cos, if, pow, mdist, warp, sqrt, sign, neg
Terminal set	x, y, scalar random constants
Phenotype width (pixels)	64
Phenotype height (pixels)	64

6.3.2 Experimental Setup

The **EFFECTIVE** parameters are presented in Table 6.14. The settings for the **EC** engine for each **EC** run are presented in Tables 6.15 and 6.16. From here on we will refer to the **EC** engine that expands the negative dataset as the *Abstract*. Also, we will refer to the **EC** engine that expands the positive dataset as the *PhotoRealistic*.

Note that both engines present different characteristics than the **EC** engines used in Sections 6.1 and 6.2. Since the *Abstract* engine uses **NORBERT**, the results are not directly comparable with the ones using **NEvAr**. Regarding the *PhotoRealistic*, although we are using the same **EC** engine as in Section 6.2, we made alterations to a parameter for the generator of photorealistic faces. In the previous experiment we exchanged, from one individual to another, pairs of eyes and eyebrows. In this experiment we do not pair both the eyes or the eyebrows. Based on the annotated dataset of 200 individuals this means that we have $200^7 \approx 1.28 * 10^{16}$ different composite faces to explore. The rationale is the following: by working with atypical faces forces the classifier to be more robust to detect faces in the training stage. As a downside, the training can

Table 6.16: *PhotoRealistic* engine parameters.

Parameter	Setting
Number of generations	50
Population size	100
Elite size	1
Tournament size	2
Crossover operator	uniform crossover
Crossover rate	0.8
Mutation operator	gene replacement
Mutation rate per gene	0.15

leave the classifier exposed to more shortcomings. Nevertheless, we wanted to push the framework to test the limits of the approach and evaluate if it is still able to adapt from iteration to iteration to the positive and negative shortcomings. In terms of fitness function, we maintain the same fitness function that is described in Subsection 4.5.2.

The Supervisor is the *External RMSE* but operates differently for the *PhotoRealistic*. With the *PhotoRealistic* engine the Supervisor must invert the condition of the selection mechanism. It must select instances that the classifier classified as negative but for the Supervisor classified as positive.

6.3.3 Experimental Results

In this Subsection we follow the same structure of Subsections 6.1 and 6.2. We start by analysing the evolutionary process, analysing the progression of the EC runs in terms of fitness, number of EC runs, and candidate instances. Then we analyse the dataset size per iteration and performance of the classifiers per iteration. For the analysis of the results we rely on the same statistical tests done in Subsections 6.1 and 6.2.

At each iteration of the framework, 30 distinct classifiers are trained. A single EC run is performed per classifier, per EC engine, which result in 60 distinct EC runs. Each EC run evaluates the individuals with a different classifier to assign fitness. As shown in 6.3.2, in this instantiation, different conditions are imposed to the *Abstract* and *PhotoRealistic*. The EC run of *Abstract* must generate 300 candidate instances, in order for the EC run to be valid. With *PhotoRealistic* it must generate 50 candidate instances to be valid. A candidate instance of *Abstract* is an instance classified as containing a face. In *PhotoRealistic* a candidate instance is an instance classifier as not containing a face. The Supervisor also operates in a different way for each EC engine. At each framework iteration it selects and filters: (i) candidate instances from *Abstract* to expand the negative dataset; and (ii) candidate instances from *PhotoRealistic* to expands the positive dataset.

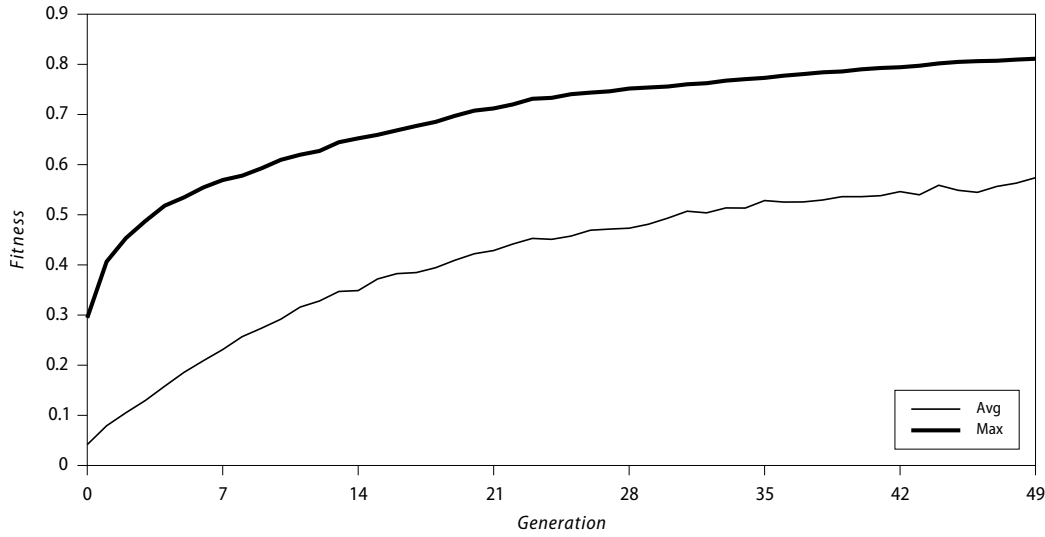


Figure 6.10: Evolution of the average and maximum fitness of *Abstract* across generations for the initial framework iteration, considering only the valid [EC](#) runs. The results are averages of 30 independent runs.

For this particular experiment we have to analyse both [EC](#) engines in terms of fitness. To improve the readability, the fitness values were normalised to the $[0, 1]$ interval by dividing the raw fitness, calculated according to the fitness function by the average maximum fitness found in the course of the framework iteration.

Starting with *Abstract*, in [Figure 6.10](#) shows the evolution of the fitness of the best individual of each generation (Max) and of the average fitness (Avg), during the initial framework iteration. The behaviour is similar to the one observed in [Section 6.1](#). A similar behaviour can be seen in [Figure 6.12](#) while using *PhotoRealistic*. In both cases, these results suggests that we are able to guide evolution with the design fitness functions.

In [Figure 6.13](#), we can observe some of the instances generated by the *Abstract* engine in the initial iteration. We can observe similarities with the results attained in [Section 6.1](#) and from [Chapter 4](#). Several of the images are not really suggestive of faces, and tend to be symmetrical and noisy images.

The average fitness results in the last iteration for *Abstract* can be viewed in [Figure 6.14](#). Once again, there are some similarities with the results attained in the evolution of false positives instances experiment. Nevertheless, the patterns seem to be more evocative of faces than the ones attained in [Section 6.1](#). The fitness results can be seen in [Figure 6.16](#). Based on the results, we can notice the curve has not stabilise near the maximum value. As seen in [Section 6.1](#), this suggests that in the last iteration it becomes harder to evolve instances classified as false positive than in the initial iteration.

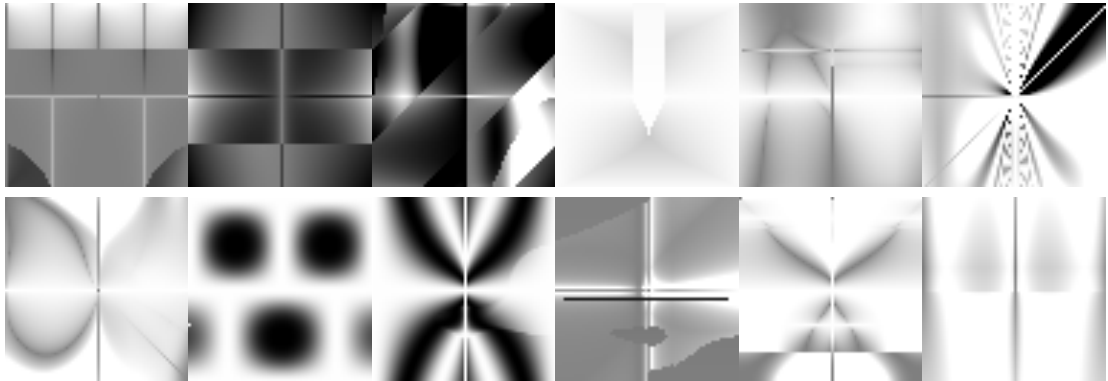


Figure 6.11: Examples collected by the supervisor of the *Abstract EC* engine. They are representative of the last population of each classifier in the first iteration, considered to be faces by the classifier.

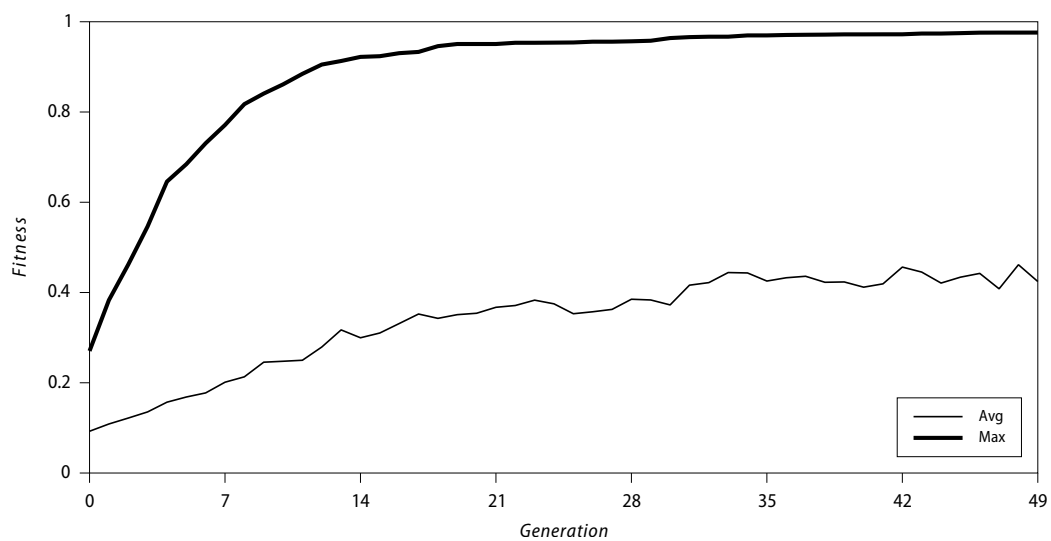


Figure 6.12: Evolution of the average and maximum fitness of *PhotoRealistic* across generations for the initial framework iteration, considering only the valid *EC* runs. The results are averages of 30 independent runs.



Figure 6.13: Examples collected by the supervisor of the *PhotoRealistic* **EC** engine. They are representative of the last population in the first framework iteration, considered to be faces by the classifier.

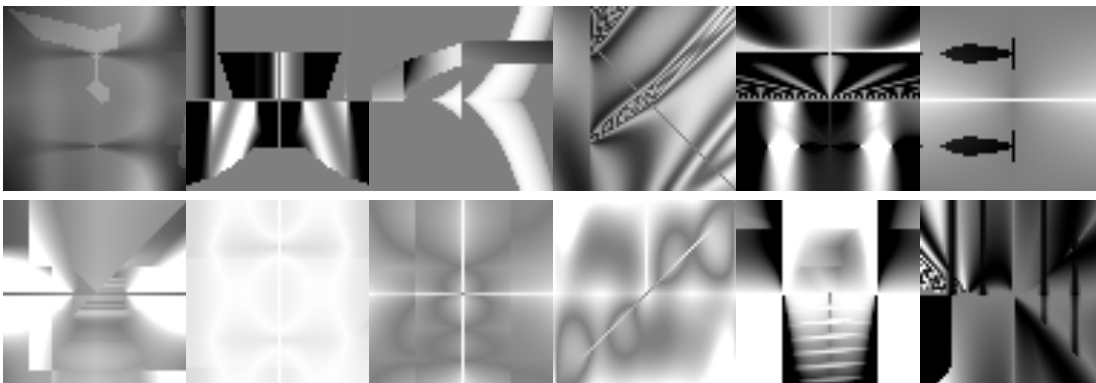


Figure 6.14: Examples collected by the supervisor of the *Abstract* **EC** engine. They are representative of the last population of each classifier in the last framework iteration, considered to be faces by the classifier.

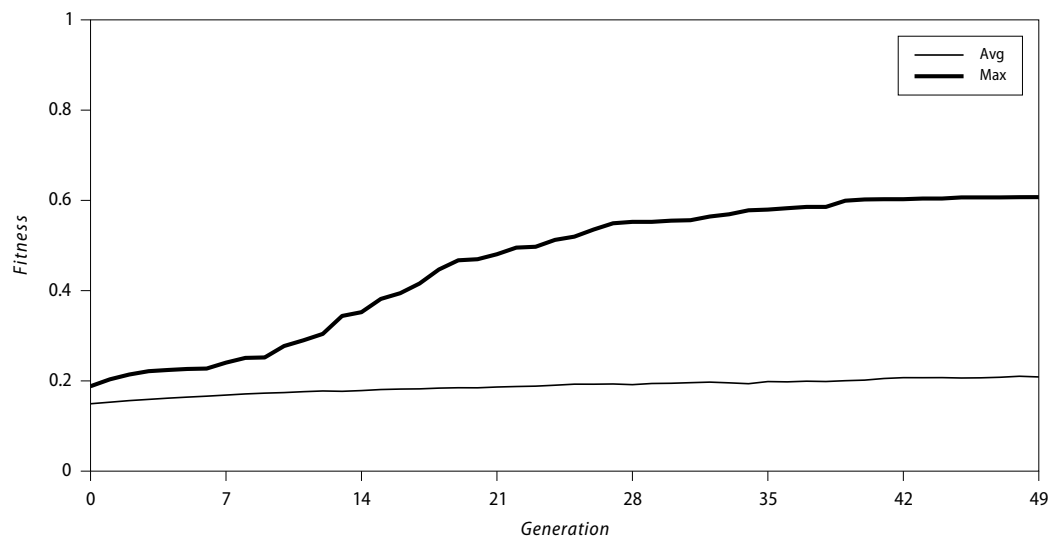


Figure 6.15: Evolution of the average and maximum fitness of the *PhotoRealistic* across generations for the last framework iteration, considering the valid EC runs. The results are averages of 30 independent runs.

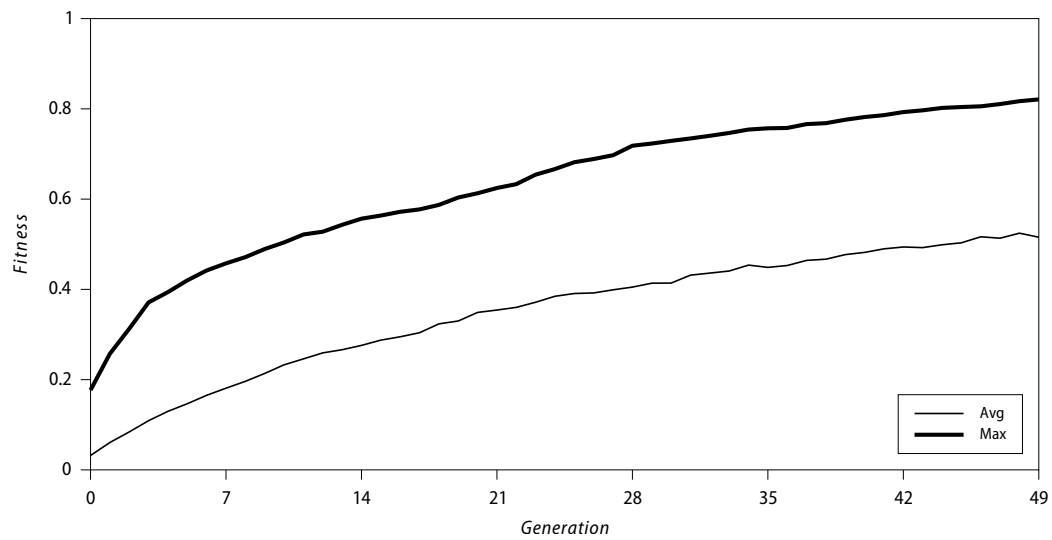


Figure 6.16: Evolution of the average and maximum fitness of the *Abstract* across generations for the last framework iteration, considering the valid EC runs. The results are averages of 30 independent runs.



Figure 6.17: Examples collected by the supervisor of the *PhotoRealistic*. They are representative of the last population of each classifier in the last framework iteration, not considered to be faces by the classifier.

Statistically significant differences between the proportion of valid EC runs of a given iteration and the previous one are indicated in *italic*. Statistically significant differences between one iteration and the last one are depicted in underline. As for the analysis of the number of candidate instances we perform the Wilcoxon-Mann-Whitney test with a confidence level of 99% with Bonferroni correction for multiple tests, which implies setting significance level to $0.01/8$ for the tests involving comparisons with iteration 8, and $0.01/2$ for all the remaining ones.

In Table 6.17 we have the values related with the EC runs valid runs and candidate instance generation. The results attained for the *Abstract* are similar to the ones observed in Section 6.1. It becomes harder to evolve candidate instances from iteration to iteration. For all EC runs we have statistically significant differences to the last iteration from iteration 0 to iteration 3. For the valid runs we only have differences to the last iteration in iteration 0 and 1. The number of EC runs needed to find the necessary candidate instances, when compared with the last iteration, increase steadily but the increase is only statistically significant for iterations 0, 1 and 3. This result suggests that from iteration to iteration the *Abstract* finds the task of finding valid EC runs harder (supported with the Median candidates column). However when *Abstract* finds a valid EC run it generates a high number of candidates (almost triple of the minimum required). This supports the idea that the representation allows the EC to easily generate candidate instances for the negative class.

The results *PhotoRealistic* differ from the ones obtained in the Section 6.2. The most noticeable difference is the increase of number of EC runs needed, average number of candidates for all the runs and for the valid ones. These results strongly suggest that after iteration 1 it becomes really hard to generate false negatives. In terms of results, since the endgame is to get an improved version of the classifier the behaviour

Table 6.17: Number of EC runs necessary to find 30 valid runs, average and median number of candidate instances per EC run, and average and median number of detections per valid EC run.

Setup	Iter.	EC runs	Avg candidates all runs	Median candidates all runs	Avg candidates valid runs	Median candidates valid runs
<i>Abstract</i>	0	<u>30</u>	<u>2558.97</u>	<u>2517.50</u>	<u>2558.97</u>	<u>2517.50</u>
	1	<u>31</u>	<u>1436.81</u>	<u>1516.00</u>	<u>1475.90</u>	<u>1524.50</u>
	2	57	<u>605.93</u>	<u>365.00</u>	1004.70	1034.00
	3	<u>45</u>	<u>696.07</u>	<u>738.00</u>	1026.80	1081.00
	4	57	491.79	356.00	890.13	792.00
	5	65	504.38	205.00	1037.93	1051.00
	6	76	440.43	65.00	1062.27	1056.00
	7	71	416.00	88.00	927.20	918.00
	8	87	362.97	86.00	960.97	824.00
<i>PhotoRealistic</i>	0	<u>30</u>	<u>2326.47</u>	<u>2414.00</u>	<u>2326.47</u>	<u>2414.00</u>
	1	<u>77</u>	<u>314.75</u>	<u>46.00</u>	<u>782.97</u>	<u>750.50</u>
	2	3900	3.50	0.00	191.87	146.50
	3	6627	<u>2.47</u>	<u>0.00</u>	182.73	103.50
	4	<u>2173</u>	<u>5.57</u>	<u>0.00</u>	251.77	132.50
	5	5202	2.92	0.00	269.90	170.50
	6	4179	<u>3.44</u>	<u>0.00</u>	248.17	130.00
	7	6966	1.79	0.00	125.17	96.50
	8	5946	1.94	0.00	137.33	103.50

Table 6.18: *Abstract* and *PhotoRealistic* dataset's size along the iterations for all setups.

Iteration	Evolving Abstract and PhotoRealistic			
	size	added to Negative	added to Positive	added total
0	1905	-	-	-
1	21823	14203	5715	19918
2	33432	8904	2705	11609
3	38737	4733	572	5305
4	43078	3783	558	4341
5	46102	2244	780	3024
6	48806	2141	563	2704
7	51238	1900	532	2432
8	53387	1999	150	2149
9	56443	2599	457	3056

registered in terms of candidate instances in the *PhotoRealistic* engine is a desirable one, arguably indicating that it became more robust.

In Table 6.18 we show the progression of the dataset size along the framework iterations. It copes with the results attained in terms of fitness and EC run stats. In the first iteration a considerable amount of instances are added to the positive and to the negative dataset (14203 negatives, 5715 positives). In the second iteration we also have a lot of instances added to the datasets. From the iteration 2 onwards there is a significant drop of instances added per iteration. The numbers for the positive instances decrease significantly confirming that it becomes hard to evolve instances that explore shortcomings of the classifier in terms of false negatives. The addition of negative instances also reduces from iteration 0 to 4 but then it maintains a quota of 2000 instances per framework iteration.

The performance of the classifiers of this instantiation is presented in Table 6.19. In Flickr we have a consistent reduction of FA and consequently increase of %C along the iterations. The first three framework iterations (0 to 2) show significant differences when compared with the iteration 9. The same behaviour is observed for the differences in consecutive iterations. The reduction of FA ranges from 1063.4 to 1154.8. The value is not the lowest value registered when comparing with the other experiments but it is lower than the values of the baseline classifier and of the Supervisor's classifier (ranging from 138 to 229.4 less FAs).

The performance of the classifiers along the iterations for the Feret dataset suggest that the behaviour we were seeking in this set of experiments occurs. The %H consistently improves with significant differences for the last iteration from iteration 0 to 5,

Table 6.19: Percentage of hits (%H), number of false alarms (FA), percentage of correct examples (%C) and average percentage of correct examples (AVG(%C)) for each test dataset, along the iterations for the *Abstract* and *PhotoRealistic* setup.

		Flickr		Feret			CMU			AVG(%C)
		FA	%C	%H	FA	%C	%H	FA	%C	
Abstract and PhotoRealistic	baseline(0)	1326.4	65.5%	96.1%	224.9	75.2%	56.8%	91.1	40.8%	60.5%
	1	263.0	89.9%	95.7%	40.0	91.7%	49.4%	20.6	44.8%	75.4%
	2	181.9	92.8%	97.2%	24.7	94.7%	52.7%	10.0	48.9%	78.8%
	3	207.2	91.7%	97.3%	32.2	94.0%	52.9%	15.0	48.0%	77.9%
	4	174.7	92.9%	97.4%	23.4	95.1%	52.0%	11.2	48.4%	78.8%
	5	171.6	93.1%	97.7%	20.7	95.6%	51.9%	11.9	48.9%	79.2%
	6	192.5	92.2%	97.9%	23.9	95.4%	52.1%	13.4	49.0%	78.9%
	7	179.5	92.7%	97.9%	24.0	95.4%	52.1%	13.6	47.9%	78.7%
	8	177.4	92.7%	97.9%	19.9	95.9%	52.8%	13.9	49.0%	79.2%
	9	171.8	92.9%	98.2%	20.6	96.1%	52.8%	12.2	48.7%	79.3%
Supervisor		401.0	83.9%	42.7%	521.0	41.9%	31.5%	244.0	21.5%	49.1%

the FA decrease with significant differences from iteration 0 to 3 to the last iteration. Consequently, the %C increases consistently with significant differences to the last iteration from 0 to 4. The significant differences from one iteration to the next occurs only once from 0 to 1 with the exception of FA which occur from 0 to 2. When comparing the framework's classifiers with the baseline classifier, the values of the %H range from -0.4 to 2.1 difference; the FA reduction range from 184.9 to 205 ; and %C range from 16.5 to 20.9 difference.

In the CMU dataset regarding the %H we observe a similar behaviour from the last Section, suggesting that the positioning of the detection box could be a cause for this behaviour. In terms of FA, the values significantly decrease in a similar way that was observed in Section 6.1. The %C values stabilise around the 49%, which is the highest value achieve in all the three experiments. In terms of significant differences to the last iteration only iteration 0 and 1 registered such differences. The analysis from iteration per iteration was registered from iteration 0 to 1 and 1 to 2. From iteration 1 to 2 we see that both %H and FA both improved at the same time and was found statistically significant. From there on, some adjustments occur until reaching the last iteration.

Overall we observe that while using **EFFECTIVE** to generate instances classified as false positives and false negatives, by expanding the negative and positive datasets, we are increasing the %H and decreasing significantly the FAs. Moreover, the results attained show that the framework classifiers yield a better overall results since the first iteration, when compared to the baseline classifier and when compared with the Supervisor's classifier.

6.4 SUMMARY

In this Chapter, the framework was used to assess and improve the performance of classifiers in a face detection problem and tested using several test datasets. First, we used **EFFECTIVE** to perform several framework iterations expanding the negative dataset by evolving false positives. The experimental results show statistically significant improvements in performance over the baseline classifiers regarding FA and, conversely, small losses of performance regarding %H. Since the improvements in terms of false alarms (FA) tend to be large (suppression of up to 89.9% of the baseline's false alarms) and the losses in terms of hits tend to be small (the worst case is a 10.3% difference in terms of percentage of hits), these two opposing forces result in a clear and statistically significant improvement of performance – differences up to 17.0% – in terms of the average number of correctly classified instances across test sets (AVG(%C)).

The results show that it is advantageous to perform multiple iterations and that the performance of the classifiers tends to improve as the number of iterations increases. The results suggest that the Supervisor has a significant impact on the performance of the classifiers, indicating that adding instances criteriously contributes to better classifiers. Afterwards, we performed a control experiment. Using the standard approach of collecting negative instances to train cascade classifiers, we trained classifiers with the same number of instances from the previous experiments but without using instances generated by **EFFECTIVE**. The results show a significant improvement of the results against all test setups, showing that the evolutionary instances are relevant and impact the performance of the classifiers.

Then we tested expanding the positive dataset, using the eXploit-faces **EC** engine (see Section 4.5), with multiple framework iterations and the *External RMSE* Supervisor. The **EC** engine was able to generate false negatives that led to a better performance than the baseline due to the increase in %H (1.0% in the Feret) and, due to properties of the training, a small decrease in terms of FA. This conjunction of results led to an increase of 3.7 regarding AVG(%C) when compared with the baseline classifier.

Finally we performed tests where we expand the negative and positive dataset. We used an **EC** engine to generate false positives, and another **EC** engine to generate false negatives. The approach was tested using the setup that attained the best results, the *External RMSE*. The results of expanding the negative and the positive dataset simultaneously led to an increase in the maximum value of performance of 79.3 regarding AVG(%C), a 18.8% performance increase when compared with the baseline and a 30.0% increase when compared with the Supervisor's classifier. The results show that by expanding both the positive and negative dataset we diminish the FAs and increase in terms of %H, which leads to an increase in terms of %C and, consequently, an increase in terms of AVG(%C). Overall, the approach showed its viability to assess and improve the classifier performance.

In this Chapter we test the approach in a **CC** context to create a system that promotes style change. This work was built upon previous ones of Machado and Cardoso (1997), Machado et al. (2007a,b), and Romero et al. (2003, 2012) incorporating ideas and parts of the work done in Correia et al. (2013b), Machado et al. (2015b), Vinhas et al. (2016). Furthermore, most of the work described in this Chapter is featured in a book Chapter (Correia et al., 2017).

McCormack (2007) posited that the development of Aesthetic Judgement Systems (**AJS**) is one of the most significant challenges in the field of **CC**. Over the course of the years, two main approaches to address the challenge emerged: the development of hardwired fitness measures that try to encapsulate some aesthetic principle and the use of **ML** techniques to learn aesthetic models (Romero et al., 2012).

As we stated in previous works (Machado and Cardoso, 1997; Machado et al., 2007a,b; Romero et al., 2003, 2012), the long-term goal is the development of Artificial Artists (**AA**s) that display the full range of abilities of human artists. In this context, the ability to learn aesthetic models is indispensable, since it gives the system the ability to experience, assess and react, not only to its artistic production but also to the artworks of other, artificial or human, artists (Machado and Cardoso, 1997). Furthermore, it also creates the preconditions that allow the system to be inspired by other artists, to detect trends, and to innovate and deviate deliberately.

The ability to consistently generate innovative and adequate artefacts is a key trait of creative, human or computational, agents. In this Chapter, we present an **AA** that is characterised by the ability to build its aesthetic model from a set of examples, and by its perpetual quest for novelty and innovation through style variation and change.

The approach resorts to an expression-based evolutionary art engine and several classifiers, in this case, **ANN**s. The **ANN**s are trained to discriminate among the artistic production of the system and that of famous artists. The evolutionary engine is used to generate images that the **ANN**s do not recognise as being products of the system, and that, as such, novel concerning its previous artistic practice. During the evolutionary runs, we also promote the discovery of a diverse set of imagery by taking phenotype similarity into account when assigning fitness.

When a set of evolutionary runs is concluded, the novel imagery they produced is added to the training set, enlarging the area of the search space covered by the system, and the **ANN**s are retrained. This leads to a refinement of the classifiers, which, in turn, forces the evolutionary algorithm to explore new paths and styles to break with its past. Thus, the consecutive discovery of new styles is attained through the revision

and refinement of aesthetic criteria for novelty of the [AA](#), while variation within style is achieved by promoting phenotype diversity.

The research presented in this Chapter builds upon our previous efforts on the same topic (e. g., Correia et al. [\(2013b\)](#), Machado et al. [\(2007a,b\)](#), and Romero et al. [\(2012\)](#)) expanding previous approaches by:

- Performing in each framework iteration a set of parallel evolutionary runs instead of a single one;
- Considering phenotype similarity to promote the discovery of a wide set of diverse images in the course of each evolutionary run;
- Using classifiers with access to a larger number of image features;
- Using a significantly larger set of training examples;
- Using an archive to summarise the innovative imagery produced by the system.

The experimental results obtained across several iterations are presented and analysed, showing the ability of the system to consistently produce novel imagery and to identify atypical images without human intervention. We consider that the results obtained in the course of the first iteration are evocative of images produced by user-guided evolution. Furthermore, we claim that the images evolved in the last of the presented iterations are of a significantly different nature, breaking the mould with the previous artistic production of the [AA](#). As such, we hypothesise that a limited form of h- and t-creativity (Boden, [2004](#)) may have been attained.

This Chapter is structured as follows: we start with the instantiation of the [EFFECTIVE](#) framework for this work [7.1](#); this is followed by the presentation and analysis of the experimental results (Section [7.2](#)) and; drawing conclusions and summarising the results of this Chapter (Section [7.3](#)).

7.1 INSTANTIATION

The architecture proposed by Romero et al. [\(2003\)](#) argues that an [AA](#) should be composed of two main modules: a creator and a critic. We employ [EFFECTIVE](#) using a general purpose expression-based evolutionary art tool as the creator and an [ANN](#) as the critic. Thus, in our scenario, the [EFFECTIVE](#) fits the role of an [AA](#), who, based on its judgment and its past experience, iteratively learns from its inspiration and produced work, evolving its craft along its existence. We consider the role of the Supervisor as the creator's *selective memory* component, i. e., managing the generated images selecting the images that will improve its knowledge and past experience. The whole process of [EFFECTIVE](#) allows the creator to learn from the past experience, generating different artworks per iteration and improving upon previous ones.

Before diving into the details of the instantiation of the framework, a succinct description on how it operates in this particular scenario follows:

1. A set of *external* images is selected; In the case of this Chapter, this set is composed of famous artworks, representing a source of inspiration for the AA;
2. A set of *internal* images is selected; in this case, the EC engine is used to randomly create a set of images, thus creating a sample of the type of imagery the evolutionary engine tends to produce;
3. The ANN is trained to distinguish among *internal* and *external* images;
4. A new set of evolutionary runs is started; The output of the ANN is used to assign fitness; Images classified as *external* have higher fitness than those classified as *internal*; additionally, phenotype diversity is also taken into consideration;
5. During the course of each evolutionary run, an archiving module keeps track of the artistic production of the AA, storing images that are both: classified as external; and diverse from other images classified as external evolved during the course of the run;
6. When the set of evolutionary runs is concluded, the Supervisor module gathers and merges the archives resulting from each evolutionary run;
7. The consolidated archive is added to the set of *internal* images;
8. The process is repeated from step 3.

One of the key aspects of this approach is the definition of two classes of images. The first class contains *external imagery*. Images that were not created by the CP system and that are usually considered interesting or of high aesthetic value. Conceptually, the external set should be seen as an inspiration for the AA. It provides a stable attractor that is meant to ensure that the evolved imagery tends to incorporate aesthetic qualities recognised by humans. The second class contains *internal imagery*, it is composed of images generated by the evolutionary engine, and describes the previous artistic production of the AA. For the purpose of the present work, this class represents undesirable imagery, since we are interested in innovation through style change.

In the present case, the task of the evolutionary module is to evolve images that the ANN classifies as external. This may be accomplished by evolving images that are:

1. Similar to those belonging to the *external* set;
2. Different from the set of internal images (e.g., images that are entirely novel, hence dissimilar from both sets).

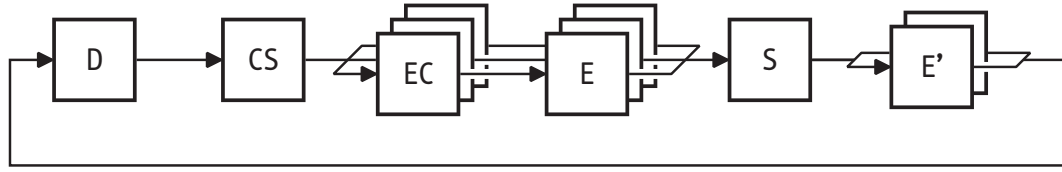


Figure 7.1: Schematic of the EFFECTIVE Framework instantiation.

Note that in this context, the concept of similarity and dissimilarity is deeply connected to the features that serve as input to the ANN. Therefore, it may deviate from human perception.

The approach relies on promoting competition between the evolutionary engine and the **CS**. In each iteration, the evolutionary engine must evolve images that are misclassified by the **CS**; otherwise, no progress is achieved. By assigning fitness using a classifier and valuing examples that belong to a predefined class, the approach evolves several misclassified examples Machado et al. (2012b). These examples can be potentially useful for improving the performance of the **CS**.

The systematic expansion of the internal set and the subsequent retraining of the **ANN** causes an arms-race between generator and classifier. As such, from iteration to iteration, the evolutionary engine is forced to explore new paths, which results in stylistic change and the expansion of the diversity of the artistic production of the system.

As pointed out by Machado et al. (2007a), in the long run, there are two possible final scenarios, which correspond to natural termination criteria for the approach: (i) the evolutionary engine becomes unable to find images that are classified as external; (ii) the **ANN** becomes unable to discriminate between internal and external imagery. The first outcome reveals a weakness of the **EC** engine, which can be caused by a wide variety of factors (deceptive fitness landscape, incorrect parametrisation, lack of computational resources, etc.). In the second outcome, there are two possible sub-scenarios: (ii.a) the images created by the **EC** engine are similar to some of the external images, which implies that the **EC** and the **CS** are performing flawlessly; (ii.b) the images created by the **EC** engine are stylistically different from the external imagery, which indicates a flaw of the **CS**.

EFFECTIVE is instantiated for this scenario with one classifier system, an evolutionary engine and a supervisor as can be seen in Figure 7.1. It starts by training a classifier with an initial dataset. Then E parallel evolutionary runs are started. When all evolutionary runs are finished, the supervisor gathers the individuals and decides which ones are going to be added to the dataset. This cycle is iteratively repeated until a termination criterion is met. The global parameters of the framework are presented in Table 7.1.

Table 7.1: Global parameters of the framework.

Parameter	Setting
Classifiers per iteration (C)	1
EC runs per classifier (E)	30
Adequacy threshold	0.5
Dissimilarity threshold	0.01

7.1.1 Classifier System

The **CS** is composed of a Feature Extraction module and an **ANN**. The classifier participation in the approach is crucial for several reasons: it evaluates the images that are generated by the evolutionary engine; its performance dictates the number of examples that are added and/or deleted before retraining the classifier.

In this work, the **CS** is trained under certain conditions before it is used to assign fitness during the evolutionary runs. On each training phase, the **ANN** is trained with the full dataset. If training is entirely successful, meaning that the **ANN** can adequately discriminate between the internal and external sets, we proceed to the evolutionary runs. However, if false externals exist, i. e., if there are human-produced artworks being classified as evolved images, these images are removed from the external dataset, and a new training attempt is made. Training is only concluded when no external images are classified as being internal.

The removal of these images has two motivations. First, from the perspective of the classifier, one can consider that the style these images embody has already been explored. As such, they should no longer be classified as external. Second, from a more pragmatic perspective, these images tend to be atypical compared with the rest of the images of the external dataset, removing them arguably simplifies the task of the classifier, which may, in turn, result in classifiers that provide fitness landscapes that are more favourable for the evolutionary engine.

The existence of false externals, i. e., evolved images classified as human-made, does not have a direct solution. Deleting them would solve nothing. Instead, they remain in the internal dataset. Future iterations are likely to explore the same shortcoming of the classifier, increasing the number of examples of the same style present in the internal dataset, and forcing, due to the increased cardinality of the subset, the classifier to learn that such images are internal.

7.1.1.1 Feature Extraction

In our approach, the **ANNs** do not have direct access to the images. Instead, each image is described by a set of image features, which serve as input to the **ANN**. As such, we developed a Feature Extractor used to extract relevant features from each image.

The pipeline of the feature extractor is the following: the input image is resized to 128×128 pixels; converted to the Hue Saturation Value (HSV) colourspace, and a copy of each image channel is stored for further computation; several preprocessing operations are computed on demand, depending on the feature to be extracted, e. g., applying Canny filter to the image and extracting information from the edges. In total, the feature extraction process yields a total of 120 features, which are later used as input for the classifier.

A thorough description of the feature extractor is outside the scope of this Chapter. Therefore, we present a brief description of the features extracted, indicating bibliographic references that may provide to the interested reader a complete description. Most of the features implemented originate from previous work concerning the aesthetic analysis of images Datta et al. (2008). The features collected were inspired by the work of Datta et al. (2006), Li and Chen (2009), Faria et al. (2013), Romero et al. (2003), Heijer (2012) and, based on our previous work (Correia et al., 2013b; Machado et al., 2007a,b).

To make the description of the feature set tractable, we introduce a taxonomy (refer to Table 7.2). Some of the features could be classified into several categories; in these cases, we followed the literature consensus for the feature's category. When selecting and developing this group of features, our goal was to cover several aspects of the images' style and aesthetics.

Although most of the features are implementations based on the state of the art, we have also introduced some new features in this work. These are briefly described in the following paragraphs.

As the name suggests, the edge density feature captures information regarding the number of edges present in the image. We compute it by applying a Canny filter to the image and counting the percentage of pixels that correspond to edges, i. e., white pixels.

We also introduce the Palette analysis features, which are intended to provide additional information regarding the image's colour palette. The core idea is to analyse the contrasting colours present in the image (Machado et al., 2015d). First, we apply a colour quantisation algorithm to reduce the number of colours using k-means clustering. The colour occurrences are counted and sorted in descending order. We compute the distances among the colours of the resulting image using the HSV space, as follows: considering two colour vectors (H, S, V) to represent the colour, the distances in the S and V colour components are calculated using the Euclidean norm; for the H channel, which is circular, we use the formula $\text{dist}(a, b) = \min(|a - b|, |a - \text{MAX} - b|, |b - \text{MAX} - a|)$ to compute the distance, where a and b are two colours and MAX is the maximum value of H. After calculating the distances, we discard the colours that are closer to each other than a predetermined threshold. This results in n colours, which we consider the image's palette. With the palette, we calculate a frequency histogram of the colours. Afterwards we compute the following metrics: number of palette colours;

Table 7.2: The proposed feature taxonomy is composed of five categories: colour, complexity, composition, saliency and texture.

Category	Features	Reference
Colour	Palette analysis	
	Average of pixel values	Datta et al. (2006), Machado et al. (2007b), and Romero et al. (2003)
	Standard deviation pixel values	Datta et al. (2006), Machado et al. (2007b), and Romero et al. (2003)
	Weber contrast	Faria et al. (2013)
	Michelson contrast	Faria et al. (2013)
	Warm and cool colors	Faria et al. (2013)
	Contrasting colors	Faria et al. (2013) and Li and Chen (2009)
	Background simplicity	Datta et al. (2006) and Rubner et al. (2000)
Complexity	Fractal dimension	Machado et al. (2007a) and Romero et al. (2003)
	Zipf size	Machado et al. (2007a) and Romero et al. (2003)
	Zipf rank	Machado et al. (2007a) and Romero et al. (2003)
	JPEG and fractal compression	Correia et al. (2013b), Machado et al. (2007a), and Romero et al. (2003)
	Horizontal and vertical symmetry	Heijer (2012)
Composition	Liveliness	Heijer (2012)
	Edge density analysis	
	Lighting	Li and Chen (2009)
	Hue Count	Datta et al. (2006) and Li and Chen (2009)
	Blur analysis	Li and Chen (2009)
	Rule of thirds	Datta et al. (2006)
	Edge distribution	Datta et al. (2006)
	Spatial frequency	Faria et al. (2013)
Saliency	Subject size	Faria et al. (2013)
	Tamura contrast	Tamura et al. (1978)
	Tamura coarseness	Tamura et al. (1978)

Table 7.3: Parameters related to the [ANNs](#) and their training.

Parameter	Setting
Initialisation of weights	random, $[-0.1, 0.1]$ interval
Learning function	backpropagation
Tolerance threshold	0.3
Learning rate	0.3
Momentum	0.2
Epochs	1000

percentage of occurrences; the mode, minimum value and maximum value for each component of the colour; histogram's linear regression and error; average distance to the next colour; average and standard deviation of the differences between the histogram's bins; components of the maximum and minimum distance from one colour to the others. We make the same analysis by considering the purity of the colours, which translates to only considering the S and V components of the image's channels to compute the metrics, ignoring the H channel.

7.1.1.2 Artificial Neural Network

The choice of an [ANN](#), as described in Section [4.1.1](#), is justified by the success of this approach in previous works of related nature (Correia, [2009](#); Correia et al., [2013b](#); Machado et al., [2007a](#)).

The [ANN](#) receives as input the feature vector. The output indicates its confidence in classifying the input instance as belonging to either the internal or the external class. To avoid a "binary" output, i. e., both neurons returning either 0 or 1, which would result in an unsuitable fitness landscape, we employ a tolerance threshold during the training stage. It translates into a modification of the training algorithm, where, during the backpropagation of the error, if the difference between the output of the network and the desired output is below the tolerated threshold, then the error is propagated back as zero (no error). The parameters of the [ANN](#) are summarised in Table [7.3](#).

7.1.2 Initial Datasets

The initial sets of external and internal images play an important role in the performance of our system. We use an external set containing 26238 images including works of artists such as Cézanne, de Chirico, Dalí, Gauguin, Kandinsky, Klee, Klimt, Matisse, Miró, Modigliani, Monet, Picasso, Renoir, van Gogh. The images were gathered from different online sources. The rationale was to collect a varied set of artworks. Although we avoided repetitions, it is relatively common for an artist to paint several versions of

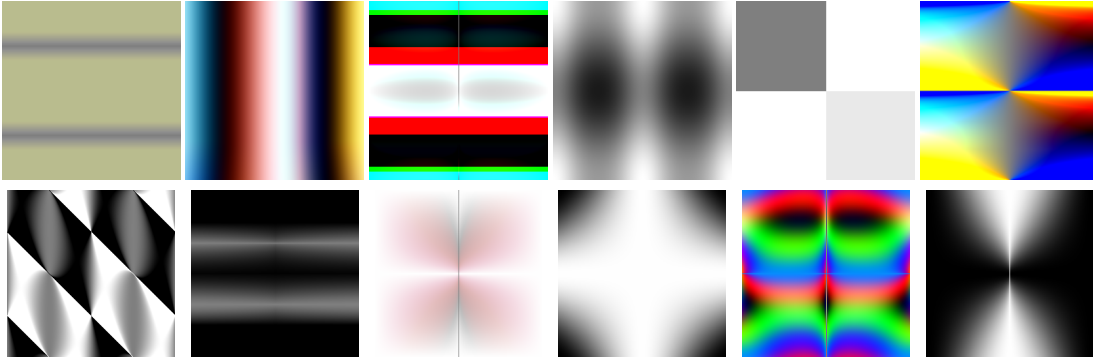


Figure 7.2: Samples of the internal dataset.

the same theme. In these cases, and to prevent the subjectivity of deciding what was sufficiently different, we decided to include the different variations.

The set of internal images is created using the evolutionary engine, described in the next Subsection, to generate 30 initial random populations of size 1600. These images are added to the internal dataset until the same amount of examples exist in the two datasets. Although the images were created randomly, some of the phenotypes may appear more than once. Figure 7.2 presents samples of the images belonging to the internal dataset, illustrating the type of imagery that the EC engine produced in these circumstances.

7.1.3 Evolutionary Engine

For this work we used the NORBERT as the EC engine (Vinhas, 2015) defined in Section 4.1.2.1. Similarly to previous works, in this instantiation the fitness of the individuals is given by the output of the CS, more precisely, by the ANN's output as described in Subsection 7.1.1.2. Figure 7.3 presents some examples of images generated with NORBERT from trial runs.

We employ a phenotype diversity mechanism by using a novelty search algorithm, designed to evolve a diverse set of adequate images. The fundamental goal of this algorithm is to generate a broader set of images than the set that would be created by a traditional fitness-based EA. In essence, it is a method capable of evolving images according to two criteria that are chosen automatically by analysing the quality of the images produced in each generation. One criterion is to look for the best images according to a fitness function and the other consists in taking novelty and fitness as two different objectives to be maximised. The reason why novelty is not considered alone is that prior tests have shown how big the search space is and, consequently, how

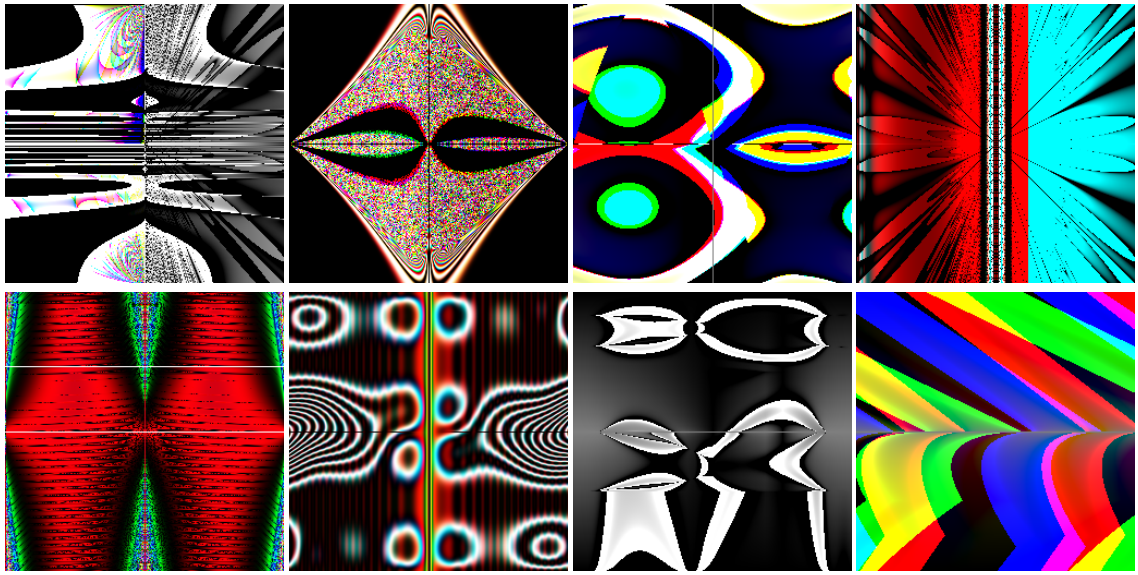


Figure 7.3: Images generated from trial runs of this instantiation using [NORBERT](#).

difficult it is to discover suitable images (Vinhas, [2015](#)). Similar behaviour has occurred when using a single criterion or considering both fitness and novelty (Vinhas, [2015](#)).

The algorithm's flowchart is similar to the traditional EA one, differing only in two main aspects: (i) the creation of an archive to store the most novel solutions and, (ii) a customised selection mechanism, which is able to consider single or multiple objectives using a tournament based strategy. The algorithm's flow is shown in [Figure 7.4](#), and can be summarised as follows:

1. Randomly initialise the population;
2. Render the images (phenotypes) from the individuals' genotypes;
3. Apply the fitness function to the individuals;
4. Select the individuals that meet the criteria to be in the archive (archive assessment);
5. Select the individuals to be used in the breeding process. The individuals are picked using one of the following criteria: (i) according to their fitness, as a standard EA; (ii) taking into account both the fitness and the novelty metric, which is computed using the archive members;
6. Employ genetic operators to create the new generation of solutions, that will replace the old one;
7. Repeat the process starting from step 2, until a stop criterion is met.

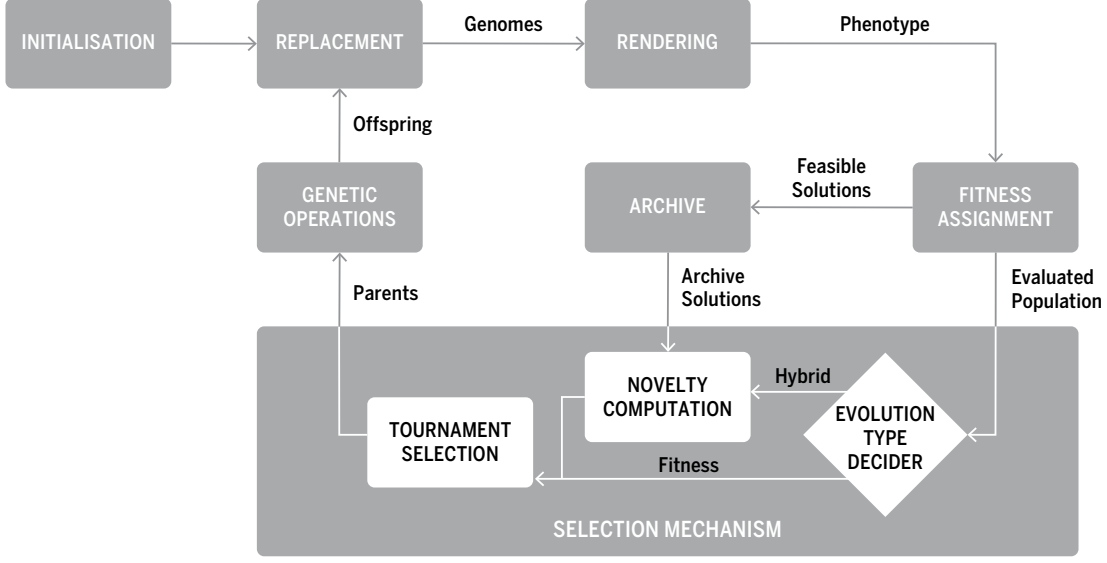


Figure 7.4: Flow of the proposed hybrid algorithm.

7.1.3.1 Archive Assessment

In this work, the archive has an unlimited size, and it plays an important role because it is used to evaluate our solution and prevents the algorithm from exploring areas of the search space already visited. The idea is that the archive should represent the spectrum of images found to date, and for this reason, the bigger the archive is, the more the algorithm can generate suitable and diverse images. Whereas in the previously mentioned works the archive size is limited, we opted for not restricting it.

At this stage, a candidate individual has its fitness assigned, and it has to meet two requirements in order to be added to the archive: (i) its fitness must be greater than or equal to an adequacy threshold f_{\min} ; (ii) it needs to be different from those that already belong to the archive. This process is performed by computing the average dissimilarity between the candidate and a set of k -nearest neighbours. When the average dissimilarity is above a predefined dissimilarity threshold, dissim_{\min} , the individual is added to the archive. The values for f_{\min} and dissim_{\min} are presented in Table

7.1.1.

The dissimilarity metric for an image i is computed as:

$$\text{dissim}(i) = \frac{1}{\max_{\text{arch}}} \sum_{j=1}^{\max_{\text{arch}}} d(i, j), \quad (7.1)$$

where \max_{arch} is a predefined parameter which represents the number of most similar images to consider when comparing with image i , and $d(i, j)$ is a distance metric that measures how different two images (i and j) are. From this dissimilarity measure there

are two exceptions that should be highlighted. If there are no entries in the archive, the first individual that has a fitness above f_{\min} is added. Moreover, if the number of archive entries is below \max_{arch} , Equation (7.1) is used with the number of archive entries instead of \max_{arch} .

For archive assessment, we resorted to an image similarity metric. Similarity metrics provide us with a notion of distance between a pair of images. The development of image distance metrics is a relevant area of research with several applications. A revision of the state of the art is beyond the scope of this Chapter. To the interested reader, we suggest consulting the works of Wang et al. (2005a) and Goshtasby (2012). Images distance metrics typically involve pixel-based operations that can be less or more elaborated. Among the available state of the art options, we chose to employ the Normalised Cross-Correlation (NCC), which can be calculated, for two images X and Y with a m by n size, in the following way:

$$\text{NCC}(X, Y) = \frac{\sum_{i=1}^{m \times n} X_i Y_i}{\sqrt{\sum_{i=1}^{m \times n} X_i^2 \sum_{i=1}^{m \times n} Y_i^2}}, \quad (7.2)$$

where X_i and Y_i correspond to the pixels of images X and Y , respectively.

NCC similarity outputs a value in the interval $[0, 1]$, where 1 indicates the best match. This measure, besides providing a fast calculation, is deemed more robust than most metrics for noisy scenes Nakhmani and Tannenbaum (2013). It suits our needs, in the sense that our approach involves a considerable quantity of images, and it can minimise the impact of noisy images on our dissimilarity assessment. As such, we use as distance metric $d(i, j) = 1 - \text{NCC}(i, j)$.

7.1.3.2 Selection Mechanism

The selection mechanism is important to shape how evolution will proceed, depending on the results obtained in a given generation. Our novelty approach has a customised selection mechanism that can switch between a fitness-based strategy and a hybrid mechanism that considers both fitness and novelty. It starts as a fitness guided evolution; however, that can change according to a decision rule, which is described as:

$$\begin{cases} \text{change_to_fitness,} & \text{adequate}_{\text{inds}} < T_{\min} \\ \text{change_to_hybrid,} & \text{adequate}_{\text{inds}} > T_{\max}, \end{cases}$$

where $\text{adequate}_{\text{inds}}$ is the number of individuals of the current generation that have a fitness above the threshold f_{\min} ; T_{\min} is the threshold used to verify if evolution should be changed to fitness, and T_{\max} is used to verify if it should be changed to hybrid.

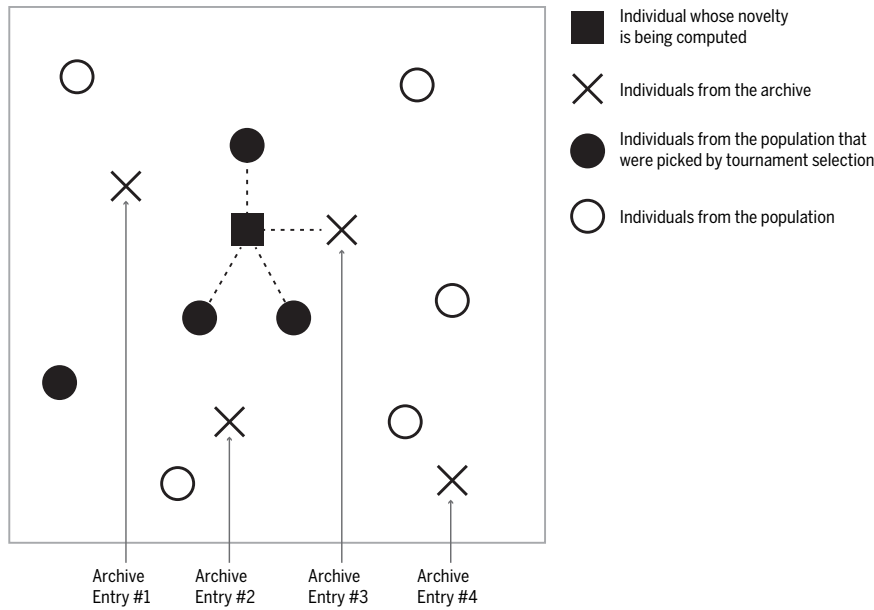


Figure 7.5: Novelty computation for an individual.

In fitness guided evolution, the tournament selection is based on the fitness values of the candidate solutions, as in a standard [EA](#). If hybrid evolution is chosen, it is necessary to compute the novelty of each selected individual, and perform a Pareto-based tournament selection, using the novelty and fitness of each selected individual as two different objectives to maximise.

The novelty computation process is inspired by Lehman and Stanley's work 2008, with one small change: the k most similar images are considered from the set of the selected individuals and the archive, instead of considering the whole population and the archive. An example of this novelty computation is illustrated in [Figure 7.5](#); considering $k = 4$ and a tournament size of 5, the dashed lines denote the chosen individuals to compute novelty, and it is possible to see that from the 4 nearest individuals picked, 3 were chosen from the tournament while the remaining one was chosen from the archive.

At this stage, each selected individual has a fitness and novelty value, and there is the need to determine the winner of the tournament. This process is inspired by multi-objective EAs, namely the Pareto-based approaches, which select the best individuals based on their dominance or non-dominance when compared to other individuals. In this work, the hybrid tournament selection determines the non-dominant solutions by comparing, among the selected individuals, both fitness and novelty. After computing the set of non-dominant individuals, we have the so-called Pareto front. The tourna-

Table 7.4: Parameters of the GP engine.

Parameter	Setting
Population size	100
Number of generations	50
Crossover probability	0.8 (per individual)
Mutation probability	0.05 (per node)
Mutation operators	sub-tree swap, sub-tree replacement, node insertion, node deletion and mutation
Initialisation method	ramped half-and-half
Initial maximum depth	5
Mutation max tree depth	3
Archive assessment width	32 px
Archive assessment height	32 px
T_{\min}	5
T_{\max}	15
Function set	$+$, $-$, \times , $/$, min, max, abs, neg, warp, sign, sqrt, pow, mdist, sin, cos, if
Terminal set	x , y , random constants

ment winner will be selected by randomly retrieving one of the solutions of the Pareto front.

The settings of the GP engine and the archive assessment for each EC run are presented in Table 7.4.

7.2 THE EXPERIMENTAL RESULTS

In this Section we present the experimental results obtained using our approach. As previously stated, one of the key characteristics of our approach is its iterative nature. In each iteration we perform 30 evolutionary runs, and once these runs end, the “external” images produced by the system, i. e., the images that expand the range of the artistic production of the system, are added to the internal set and the ANN retrained, promoting the discovery of novel images in subsequent iterations.

We are, therefore, primarily interested in analysing the differences, in terms of produced imagery, that occur from iteration to iteration. It is impossible to show all the

Table 7.5: Statistics regarding evolutionary process across iterations. The results pertain 30 independent evolutionary runs for each framework iteration.

Framework iteration	Evolved external images	Images added	Seeds with ev. external	Avg. generations for ev. external
1	38283	30110	30	3.63
2	1426	250	22	18.22
3	816	195	17	20.52
4	752	178	24	25.33
5	366	57	10	29.3
6	1105	433	21	20.24
7	620	131	22	31.64
8	191	31	7	23.86
9	422	115	21	24.90
10	692	62	20	28.5
11	374	126	22	32.27
12	267	101	11	31.09
13	842	352	17	21.76

images produced in the course of the evolutionary runs. Even if we only presented the images classified as external, this would imply presenting 38283 images for the first iteration alone. As such, we will present a synthesis of the results, which aims to convey the key experimental findings. We divide our analysis into Subsections as follows: first, we present and examine the results concerning the evolution of fitness throughout iterations; next, we will inspect the images produced; finally, we analyse the classifier’s training and performance in each iteration.

Although we present results concerning 13 iterations of the framework, it is important to stress that further iterations are still being performed. Therefore, the process is not concluded, and all evidence indicates that a significantly higher number of iterations would be necessary before a *breakdown* of the EC engine or classifier takes place.

7.2.1 Analysis of the Numeric Results Concerning Evolution

Table [7.5](#) depicts a series of statistics concerning the evolutionary process across iterations, namely: the total number of images evolved in the course of the 30 evolutionary

runs of each iteration that were classified as external (*Evolved external*); the number of these that were added to the internal set used to train the classifier guiding the next iteration after supervision (*Images added*); the number of seeds in which the EC engine was able to find at least one image classified as external (*Seeds with ev. external*); the average number of generations necessary for finding an image classified as external (*Avg. generations for ev. external*). This average is calculated taking only into account the seeds where at least one image classified as external was found.

As it can be observed, a striking number of images classified as external was found in the course of the first iteration, 38283, which corresponds to an average of 1276.1 per evolutionary run. All of the evolutionary runs were able to find “external” images and, on average, they took 3.63 generations to find the first image classified as external.

Although this number is somewhat surprising, it is far from being unexplainable. In essence, this result means that it is easy for the system to break from its “past” and produce novel imagery.

The initial set of internal images was created by randomly generating genotypes and their corresponding phenotypes. As such, the images of the initial internal dataset did not undergo evolution. By supplying an aesthetic model, and a mechanism that steers evolution towards regions of the search space that were not covered by the initial dataset, we are fundamentally changing the nature of the images that the system tends to produce. When confronted by images that are novel, and that probably do not fit in either of the categories (internal or external), the classifier is forced to make a choice, eventually classifying some of these new images as external. Once such image is found, evolution quickly explores and exploits such type of imagery, leading to the discovery of a high number of images classified as external.

On a second stage, the phenotype diversity mechanisms “kicks in”, contributing to the discovery of a diversified set of images classified as external. The importance of the phenotype diversity mechanism can be verified by the fact that out of the 38283 classified as external, 30110 were added to the internal dataset. Thus, only 8173 of the evolved images classified as external, roughly 21%, was considered similar to the ones already in the archive of their corresponding evolutionary runs and, therefore, discarded. This result shows that the phenotype diversity mechanism is able to prevent stagnation of the evolutionary runs and convergence to a fixed type of image.

After the “explosion” of novelty that occurs in the first iteration, the task of the evolutionary engine and, as will be seen, of the classifier, becomes increasingly harder and an abrupt decrease of productivity is verified. In the course of the 30 generations of the second iteration, the EC engine found 1426 images that were classified as external. Although this is still an impressive number, it pales in comparison with the numbers observed in the first iteration. This increase in difficulty can also be observed by the increase in the average number of generations necessary to find an external image 18.22 and by the fact that only 22 out of the 30 evolutionary runs were able to find images classified as external. The chart presented in Figure 7.6 concerning the evolu-

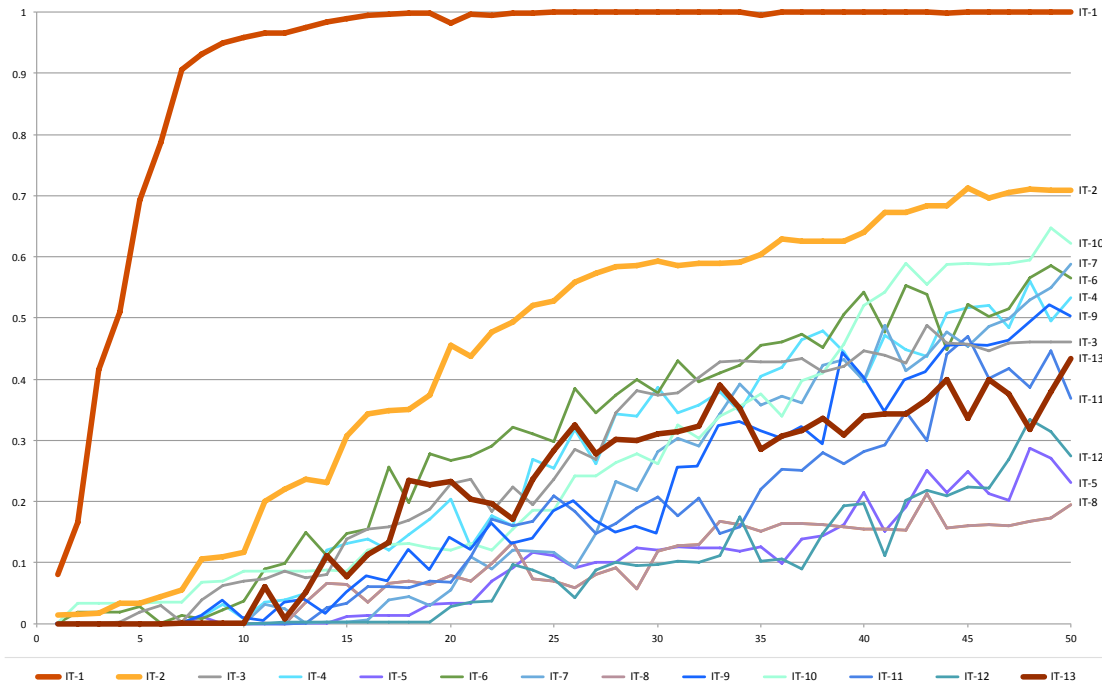


Figure 7.6: Evolution of the fitness of the best individual of each generations. Results are averages of 30 independent evolutionary runs for each iteration.

tion of the fitness of the best individual of each generation across iterations, further highlights the differences in the difficulty of the task of the EC engine in the first and second iteration.

Out of the 1426 external images found in the course of the second iteration, 250 were added to the internal dataset, since the remaining 1176 were considered sufficiently similar by our archiving algorithm to these 250. This illustrates a well-known fact concerning novelty search algorithms (as defined by Lehman and Stanley 2008): as optimising fitness becomes harder, it becomes significantly more difficult to find solutions that are both novel and fit. In other words, although the phenotype diversity mechanisms are activated and contribute to the diversity of the population, finding images that are simultaneously novel, in relation to the ones evolved in the course of the evolutionary run, and adequate, i. e., classified as external, becomes increasingly difficult.

As the number of iterations increases, and as the internal dataset becomes larger, one would expect an increasing difficulty in finding images classified as external (and also an increasing difficulty in learning to differentiate between the two sets). Although this tends to be true, it is not always the case. As Figure 7.6 illustrates, although there is a clear differentiation among the lines representing the evolution of fitness of the first two iterations and the remaining ones, and although these differences are statistically

significant, the same does not happen in the remaining iterations. The explanation for this fact is twofold: (i) the number of images added in each iteration is not sufficient to make the task visibly harder; (ii) the training of the classifier includes a stochastic component, and as such, even if trained with the same datasets, different classifiers may induce different fitness landscapes with different difficulties.

7.2.2 *Analysis of the Visual Results*

Next we make an analysis of the visual results, i. e., the images produced in the course of the 13 iterations. The complexity of the setup, and the vast number of images classified as external that were evolved make this analysis particularly hard. Furthermore, and although we will try to be as objective as possible, the analysis entails a degree of subjectivity that cannot, and perhaps, should not, be avoided. We divide this analysis in three Subsections, focusing, respectively, on the analysis of the visual results of the first iteration, intermediate iterations, and thirteenth iteration.

7.2.2.1 *First Iteration*

We begin by trying to convey what happens within each of the 30 evolutionary runs of the first iteration. For this purpose, Figure 7.7 depicts the fittest individual from each of the 50 generations of a typical evolutionary run of the first iteration. As it can be observed, the fittest images of the first two generations are entirely amorphous. By the third generation, the EC engine finds the first image classified as external. From this point onwards, the phenotype diversity mechanism kicks in, promoting the discovery of images that are, simultaneously adequate, i. e., classified as external, and different from the ones previously evolved in the course of this specific run. This mechanism does not produce immediate effects regarding the fittest image of the generations, but it prevents the algorithm from converging, and creates the conditions for the discovery, within the evolutionary run, of different images that are also classified as external. As such, the apparently abrupt changes that can be observed in Figure 7.7 result mainly from a progressive evolutionary process that promotes the diversity of the population.

In Figure 7.8 we present a sample of the images classified as external evolved in the course of the same run of Figure 7.7. Since we were unable to find a reasonable algorithm for automatically sampling the set of evolved images convincingly, this and other samples presented in this Chapter were selected by hand, trying, in all cases, to maximise the diversity of the sample and make it as representative as possible. As it can be observed, the diversity of the populations and the images being classified as external are more extensive than what Figure 7.7 suggests, showing the adequacy of the phenotype diversity mechanisms.

Figure 7.9 depicts the fittest individual of each of the 30 evolutionary runs of the first iteration. All of these images have been classified as external. There are, at least,

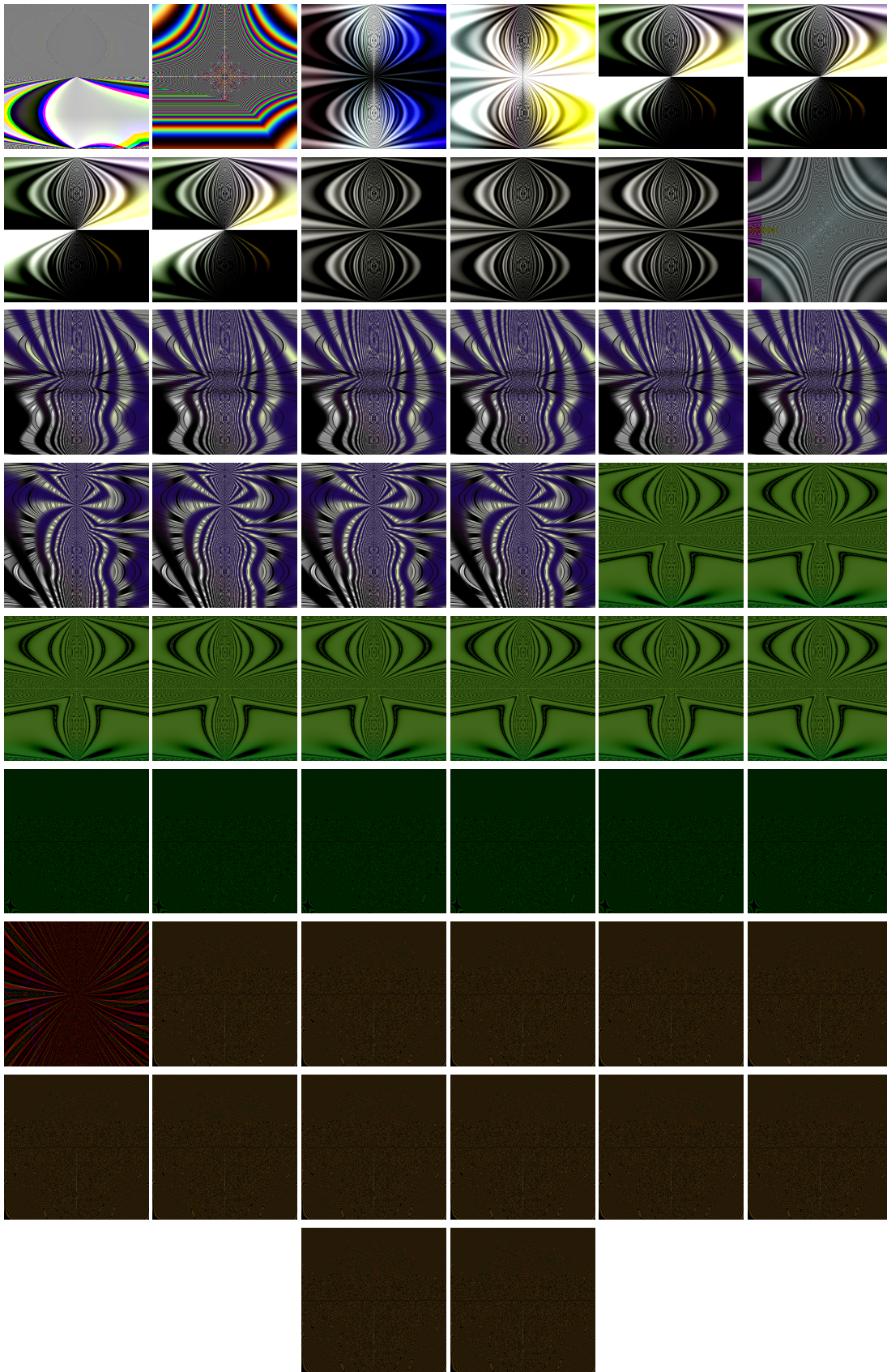


Figure 7.7: Fittest individual from each population of a typical evolutionary run of the first iteration. The image in the upper-left corner corresponds to population 0; remaining images in standard reading order.

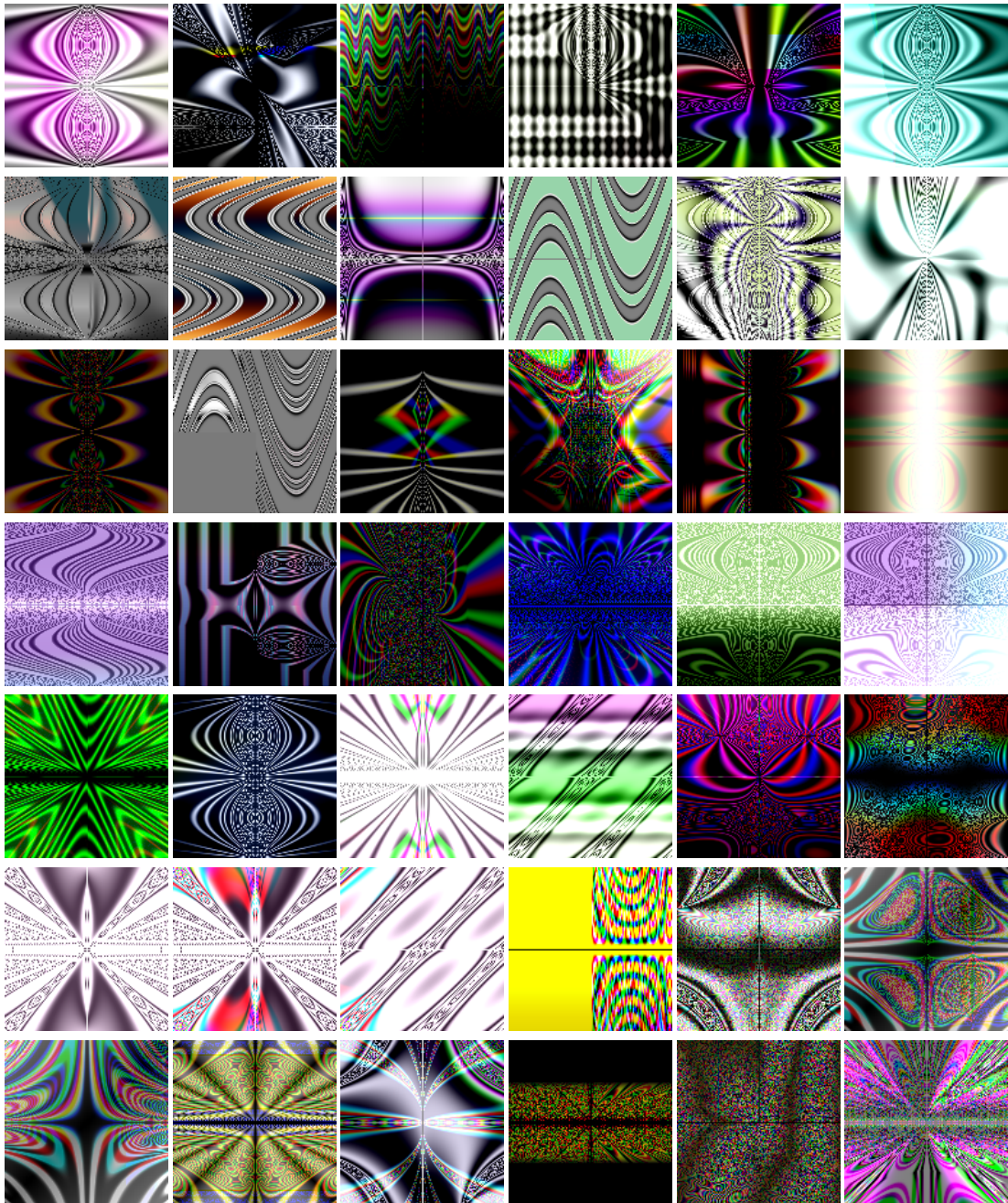


Figure 7.8: Samples of the images classified as external, generated throughout the course single typical evolutionary run of the first iteration.

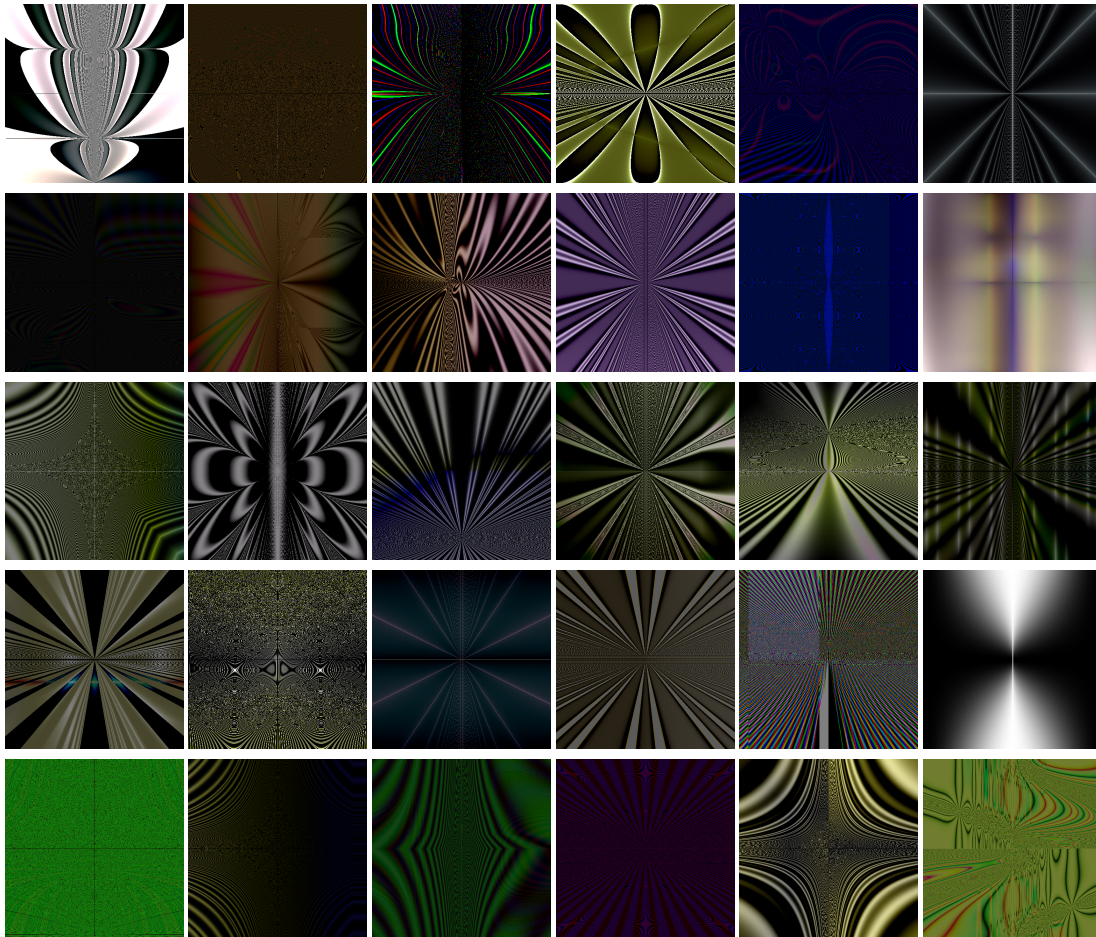


Figure 7.9: Fittest individuals of the last generation of each of the 30 seeds of the first iteration.

three predominant traits: most of the images tend to be dark and with low contrast; several exhibit a star-like shape; many of them include some noise. In some cases, the contrast is so low that the images appear to be of a uniform colour for the human eye; however, a colour adjustment and equalization operation will reveal the hidden structure. Regarding this point, it is relevant to point out that several of the features that serve as input to the ANN are invariant regarding contrast among colours, so these results also highlight the differences in the perception of images between humans and ANNs. It also appears to be safe to state that several of the runs converged to the same type of imagery, which is an expected result. The runs are performed in parallel, and the classifier, which ultimately defines the fitness landscape, is common to all. Therefore, the fitness landscape has the same local and global optimum, optima with a larger basis of attraction are bound to be explored more often. Additionally, each evolutionary run has its archive and no access to the archives of others; therefore, the phenotype diversity mechanisms cannot avoid imagery being explored in other evolutionary runs – they only operate within the production of a specific run.

Figure 7.10 presents a sample of the 38283 images evolved in the course of the first iteration and classified as external. Obviously, the visual inspection of 38283 and the selection of a representative set sufficiently small to present in this Chapter is close to impossible. Nevertheless, we believe that the selected samples illustrate the diversity of images classified as external that were evolved throughout this iteration.

Based on the results presented, we believe it is safe to claim that the images classified as external are substantially different from the ones belonging to the initial dataset. On the other hand, it is also safe to state that they are substantially different from the external dataset composed of human-made artworks. In a nutshell, the EC engine is producing images that are distinct from both initial datasets, and that the classifier, which is forced to classify them into one of these two sets, identified as external. We also believe that it is safe to claim that these images are novel concerning the ones previously produced by the EC system (i. e., the initial set of internal images), not only from a computational perspective but also to the human eye.

In our subjective opinion, several of these images are aesthetically interesting and appealing. Considering our background and experience using user-guided evolutionary art systems, it is relevant to make the following observation: these images are in many ways similar to the ones we evolved through user-guided evolution. Anecdotal evidence of this fact is that, when confronted with Figure 7.10, one of the authors asked “Why do we include user-guided images?”. Proving that this resemblance is real is beyond the scope of the present Chapter, nevertheless, even without strong evidence to make this claim, we consider this one of the most unexpected, and possibly relevant, results of this Chapter.

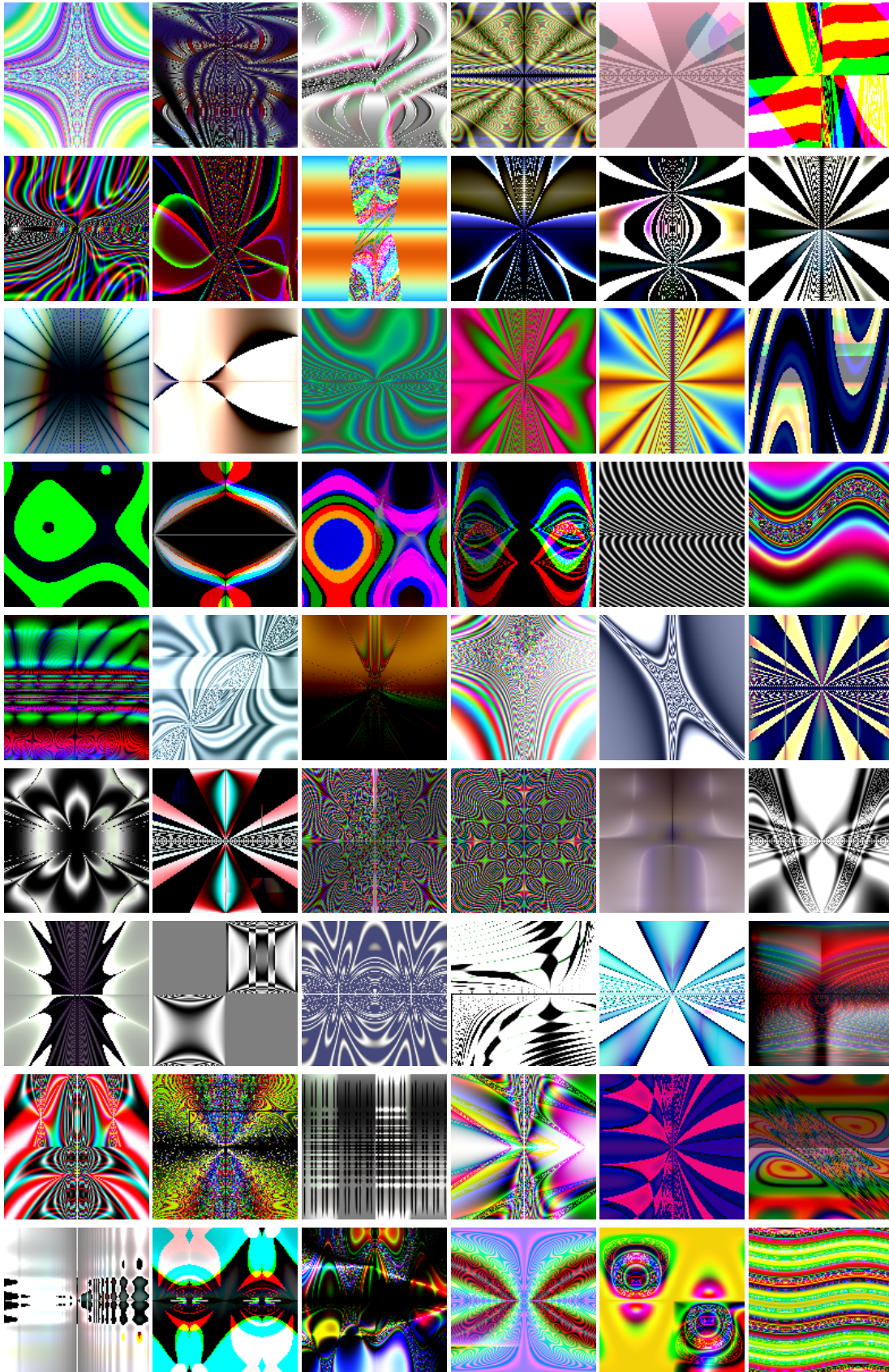


Figure 7.10: Samples of the images classified as external, generated throughout the course of the 30 evolutionary runs of the first iteration.

7.2.2.2 *Intermediate Iterations*

In this Subsection we make an overview of the visual results obtained from the second to the twelfth iterations. These results are illustrated by the samples of the images classified as external presented in Figures 7.11 to 7.21. It is important to remember that, in most cases, for each of the images presented in the figures a significant number of images of similar nature was evolved throughout the corresponding run.

Rather than making a detailed analysis, we will focus on highlighting some of the most striking results obtained in each iteration, identifying, whenever possible, trends that emerge in several runs and that, as such, represent optima with a large basis of attraction for the classifier being used in that particular interaction.

In the course of the second iteration, the EC engine evolved 1436 images classified as external. These images result from 22 of the 30 runs. The images presented in the figures are ordered by the evolutionary run. As such, two similar images presented side by side typically indicate that they were evolved in the same run, similar images that are not adjacent to each other typically indicate the rediscovery of the same type of imagery in two different runs.

A brief scrutiny of the images presented in Figure 7.11 reveals that most of the evolutionary runs converged to different imagery, but also the recurrence of some themes. Among these, we highlight the stripped star-like shapes, which also emerged in the first iteration, and that continues to be present, although “rendered” in a different style. One of the interesting results concerns the evolution of several “minimalistic” images (e. g., the two rightmost images of the first row and the leftmost images of the fifth row), which occurs in several runs. Although they appear minimalistic, this type of image is particularly hard to evolve, and their simplistic nature contrasts with the size of their genotypes. In fact, an inspection of the learning process after the second iteration appears to indicate that the emergence of these images is deeply related to the presence in the initial dataset of external imagery that is also minimalistic and monochromatic (see Subsection 7.2.3). In fact, the use of a reduced colour palette occurs in several of the evolutionary runs. This is consistent with the colour schemes used in many of the images belonging to the external dataset and contrasts with the typical imagery produced by the EC engine. More importantly, considering the nature of this Chapter, the appearance of the evolved images classified as external appears, in most cases, to be different from the initial dataset of external images and the images evolved in the course of the first iteration.

Analysing the images produced in the course of the third iteration, from which a sample is presented in Figure 7.12, one can observe the same overall patterns: most runs tend to converge to different types of images; most evolved novel imagery in relation with the previous production of the system; there are some recurring themes, namely the star-like images, which are “rendered” in different styles. The emergence of images with strong and contrasting colours (magenta, green, yellow, white, black) occurs in several evolutionary runs. This type of imagery is highly atypical of the

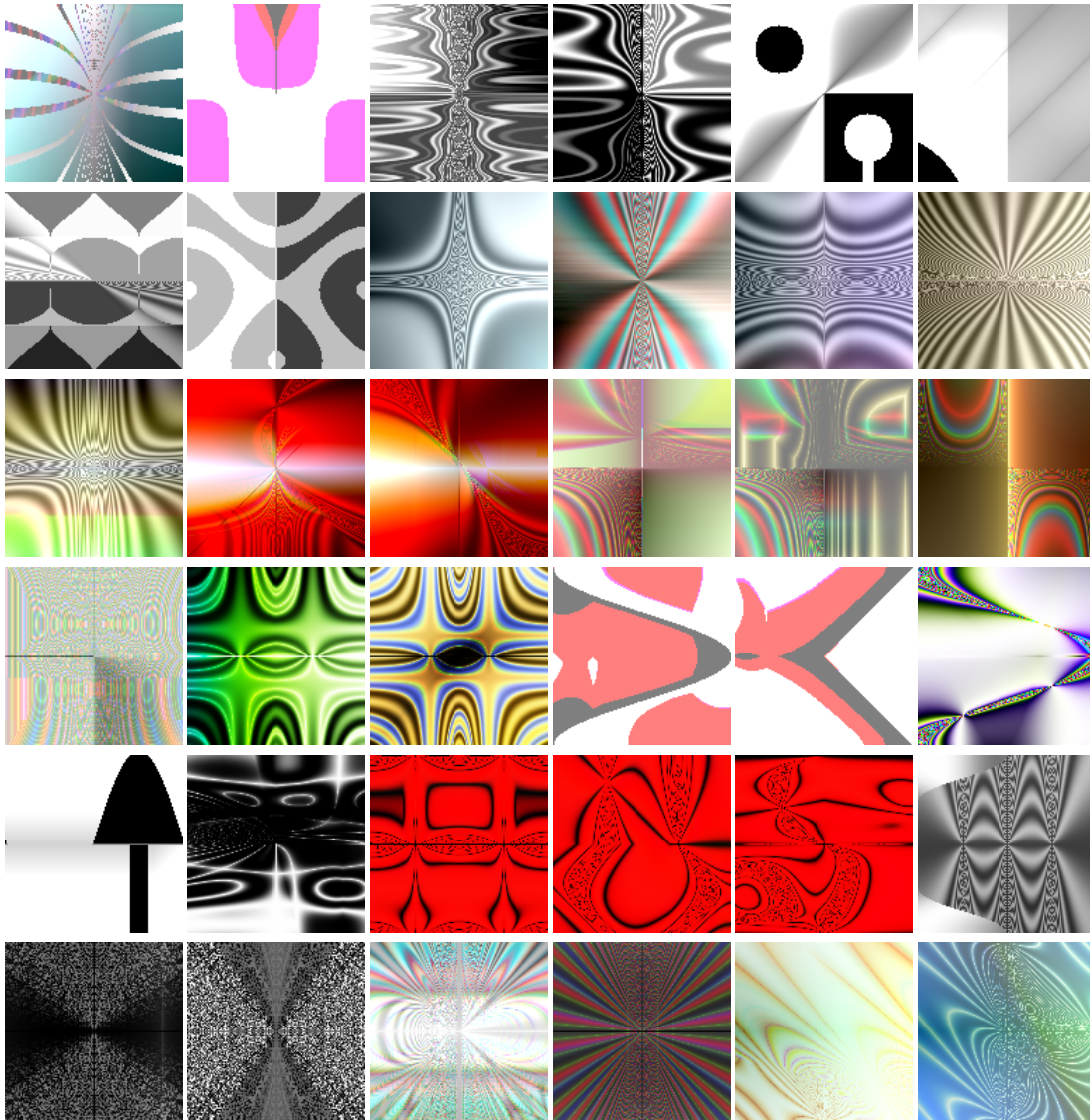


Figure 7.11: Samples of the images classified as external, generated throughout the course of the 30 evolutionary runs of the second iteration.

EC engine and matches the chromatic characteristics of several of the artworks of the external set.

As we will see when analysing the results of other iterations, the emergence of graphic elements such as lines, points and planes, also characterises some of the evolved images. Although these are usually considered graphic primitives for humans, the EC engine has no explicit way of creating such elements. As such, their emergence is deeply linked with the fitness landscape induced by the classifiers.

Much of what was stated regarding the images evolved in the third iteration also applies to the ones evolved in the fourth (see Figure 7.13). Many of the images are characterised by the emergence of organic lines and planes. Others appear to be composed of multiple layers with transparencies (e.g. leftmost image of the third row).

In the fifth iteration, the **EC** engine experienced difficulties in finding images classified as external. Only 10 of the 30 evolutionary runs found such images and, on average, these took 29.3 generations to evolve. In total, 366 images classified as external were evolved, a number that is reduced to 57 by our archiving algorithm. For these reasons, the diversity of the images presented in Figure 7.14 is not as broad as in previous iterations. The feature common to all of these images is the presence of “noise” patterns. It is also interesting to notice that a vast percentage of the images is monochromatic and with intricate detail. In several of the cases (e. g., the black and white images of the first and last row) the lines are discontinued, in the sense that they emerge from the arrangement of several white or grey dots that are not connected.

Although the productivity of the **AA** during the fifth iteration was not high, the addition of these images to the internal dataset, coupled with the removal of some of the external images (see Subsection 7.2.3), appears to cause profound changes in the classifier. There is a burst of productivity in the course of the sixth iteration, 1105 images of which 433 are added to the archive, a number that is only surpassed by the first iteration. As Figure 7.15 illustrates, this burst of productivity coincides with a change of style in comparison with the previous iterations. This sudden increase in productivity can be explained by the performance of the classifier and will be discussed in Subsection 7.2.3.

Productivity decreases during the seventh iteration, see Figure 7.16, and reaches an all-time low in the eighth iteration (refer to Figure 7.17). Generally speaking, one can state that the images classified as external evolved in the course of the seventh iteration correspond to variations in the style of themes already explored in previous iterations, almost as if the **AA** further refined and included additional detail to previously explored images. The ones evolved in the eighth iteration appear, in our eyes, to be of the same style as images evolved in some of the previous iterations. The classifier is not able, even during training, to fully discriminate among the internal and external datasets. As previously explained, this opens the door for the repetition of styles and imagery that was not sufficiently explored in previous iterations. Our analysis indicates that this is what happened in the course of the eighth iteration, the AA artist

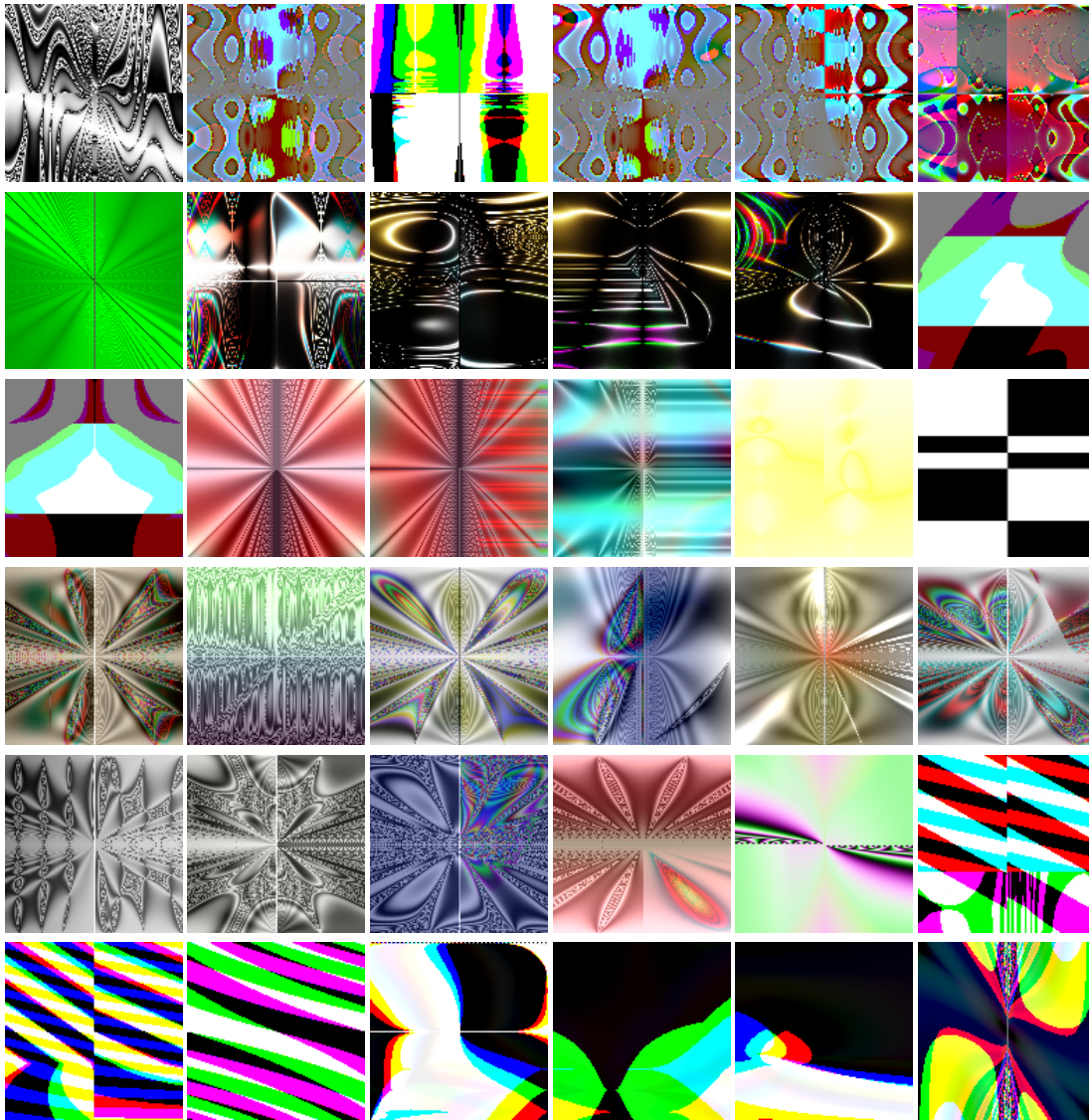


Figure 7.12: Samples of the images classified as external, generated throughout the course of the 30 evolutionary runs of the third iteration.

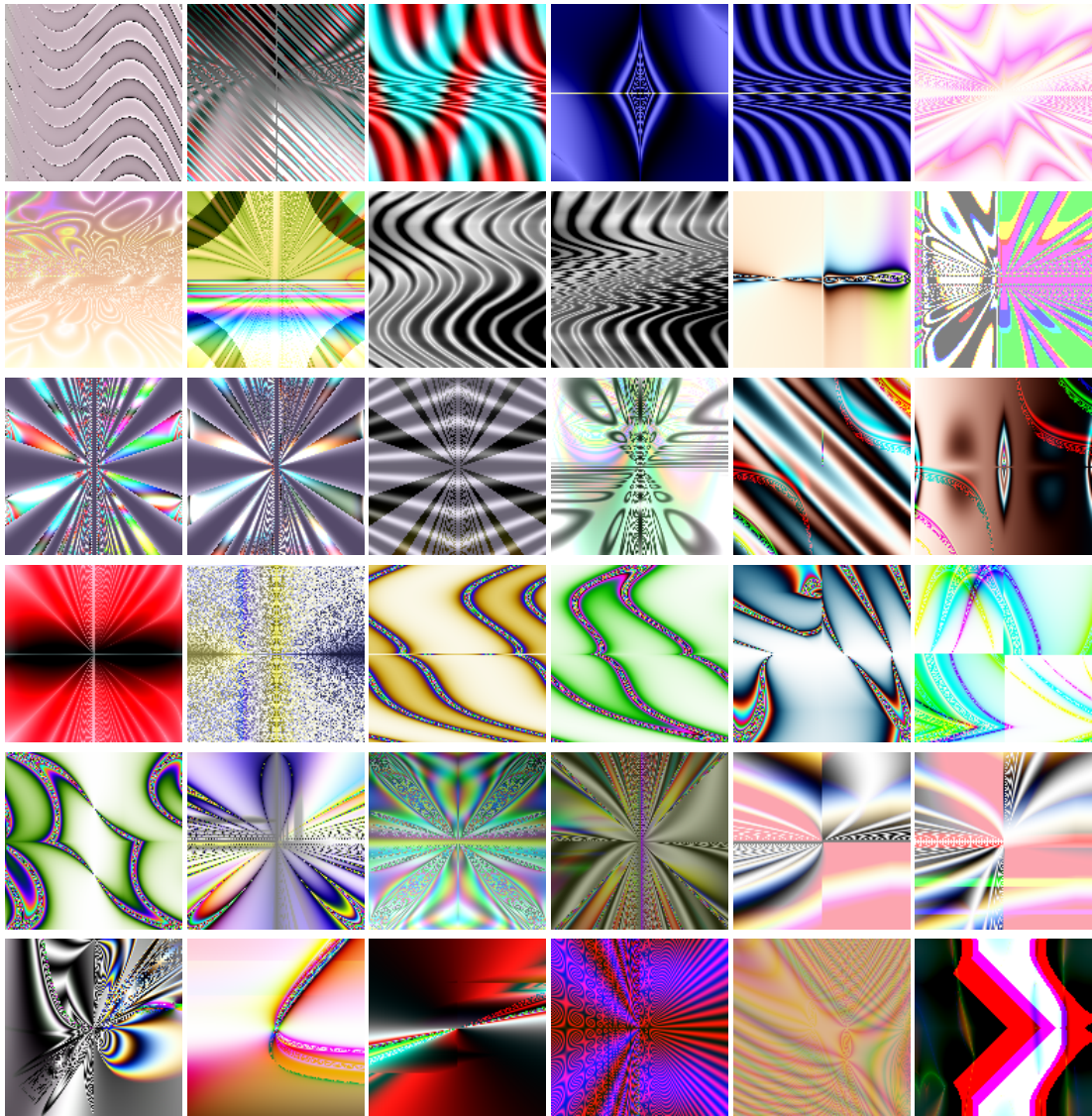


Figure 7.13: Samples of the images classified as external, generated throughout the course of the 30 evolutionary runs of the fourth iteration.

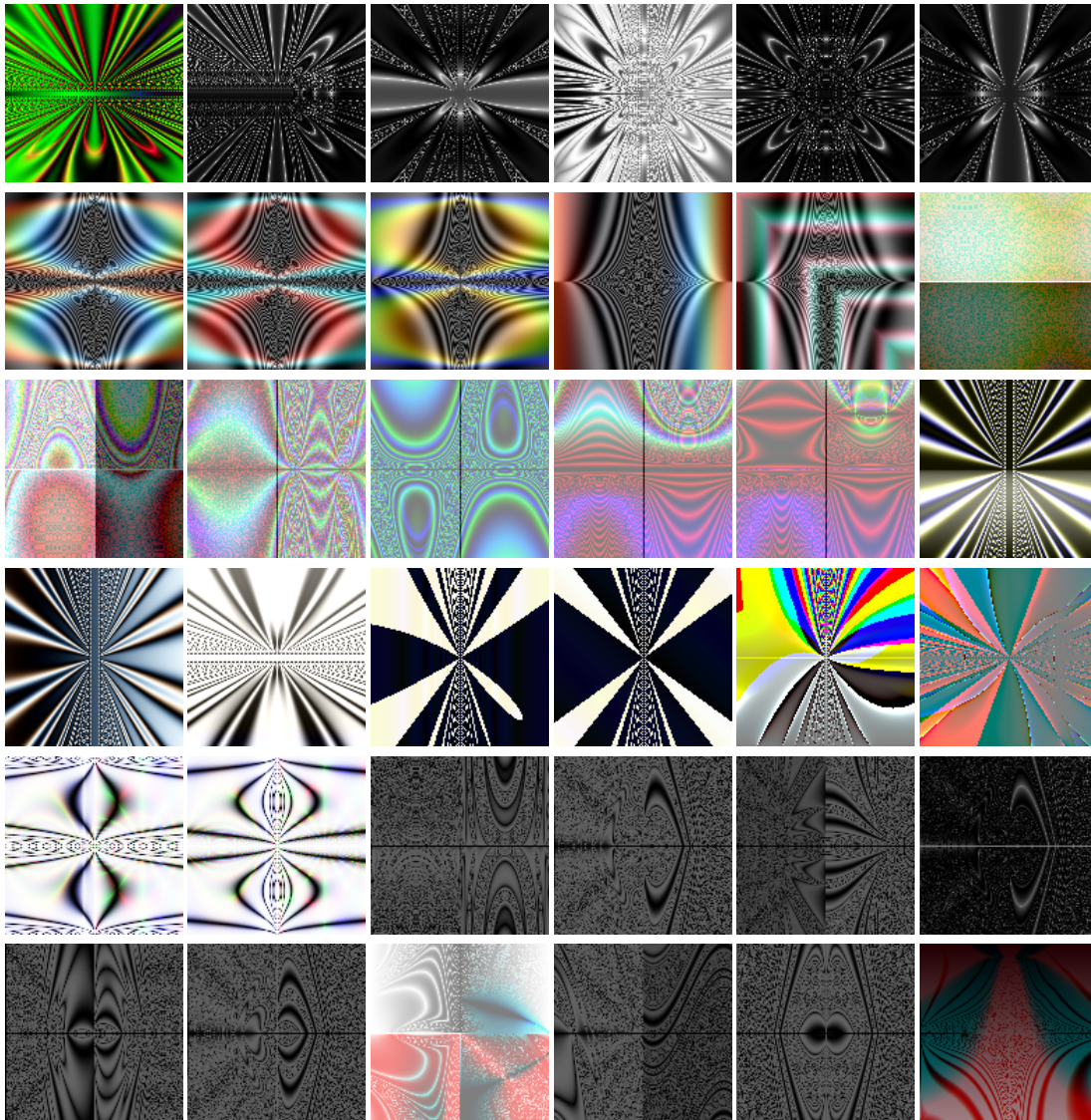


Figure 7.14: Samples of the images classified as external, generated throughout the course of the 30 evolutionary runs of the fifth iteration.

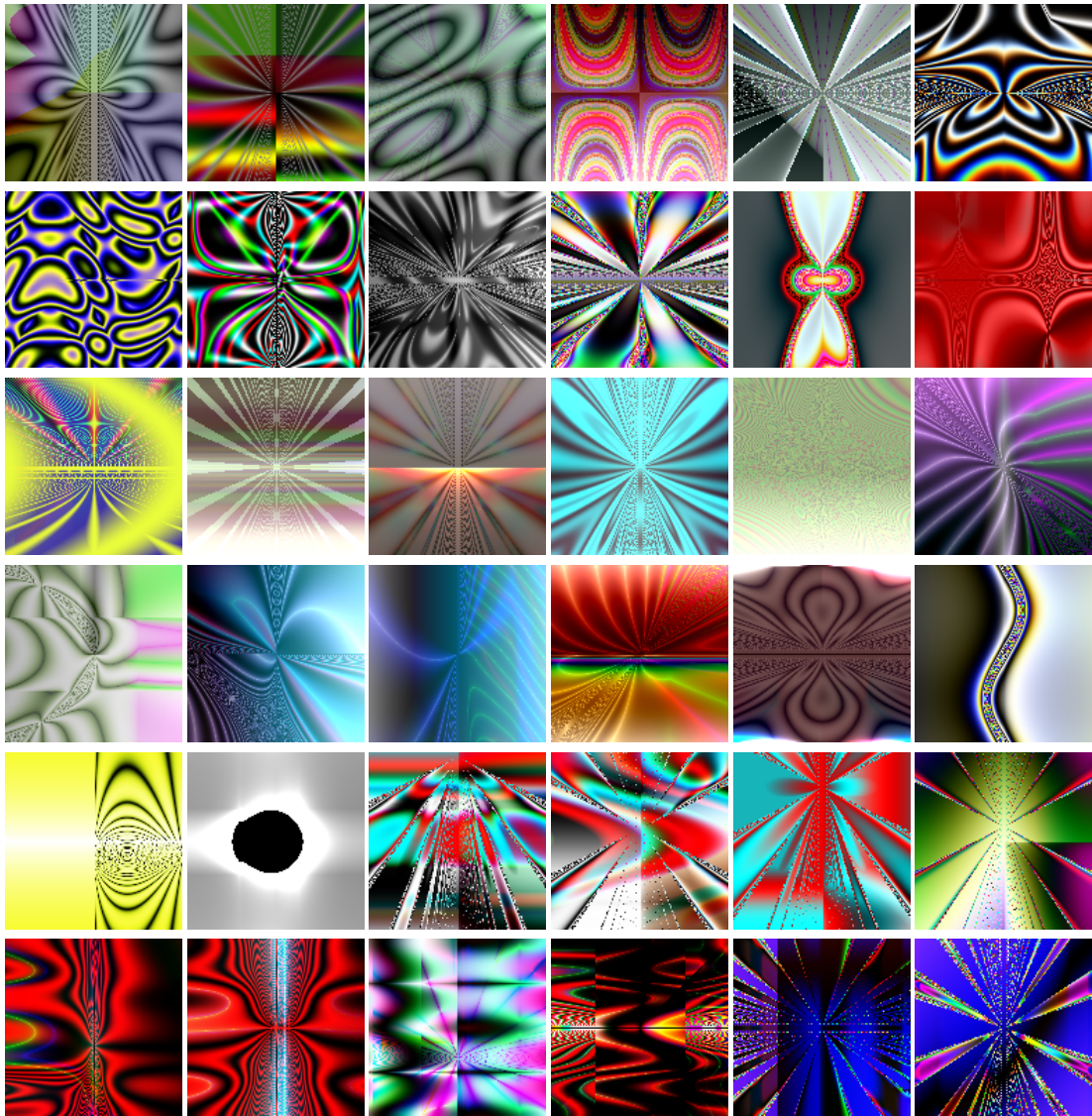


Figure 7.15: Samples of the images classified as external, generated throughout the course of the 30 evolutionary runs of the sixth iteration.

explored styles that, although already present, were not sufficiently explored. As the cardinality of such images increases the **CS** is “forced” to recognise such images as internal and, therefore, the **EC** engine will no longer be able to explore them in future iterations.

As previously, the changes to the datasets gave rise to a classifier that bases its assessments on different premises, inducing a different fitness landscape, which happens to be more prone to evolution. In the ninth iteration, the **EC** engine evolved a total of 422 images, of which 115 were archived, finding images classified as external in 21 of the 30 runs. As can be observed by inspecting Figure 7.18, there is a mixture between new and old themes and styles. Interestingly, several images that are evocative of landscapes (three leftmost images of the third row) were evolved.

The tenth iteration was one of the most productive ones regarding the total number of images classified as external, 692, but of these only 62, less than 10% made it to the archive. As it can be observed in Figure 7.19, several runs converged to the same type of imagery, reducing the overall diversity and productivity of the set. Visually, we identify three main styles which emerge in several runs: the black and white minimalist images; images that appear to have a white transparent layer (e.g., the five leftmost images of the first row, and also the two rightmost of the last row); The images exploring a combination of magenta and green. Like in several of the previous iterations, the star-like shape continues to be one of the favourite “themes” of the **AA**.

The lack of diversity of the tenth iteration contrasts with the visual diversity of the eleventh. Although only 374 images classified as externals were found, 126 of these images were archived. On average it took 32.27 generations to find the first image classified as external. Although there is some stylistic agreement among several evolutionary runs (see Figure 7.20), the overall diversity is significantly higher than in the previous iteration. The purely black and white images disappear from this iteration onwards, likely due to the combination of two factors: the inclusion of several of these images in the internal dataset and, most importantly, the removal of a large number of strictly black and white images from the external dataset.

The twelfth iteration is among the least productive ones, only 267 images classified as external were found and only 11 of the 30 runs found such images. In spite of this lack of productivity, visible in Figure 7.21, some of the evolutionary runs were able to find novel imagery that contrasts both in terms of style and theme with the previous artistic production of the system.

7.2.2.3 Thirteenth Iteration

The thirteenth and last iteration presented in this Chapter corresponds to the burst of novelty and productivity of the system. Although a similar burst occurred in the sixth iteration, the nature of the burst appears significantly different. In this case, the increased productivity is coupled with significant stylistic variations and may be seen as a moment where the **AA** actually “broke the mould”.

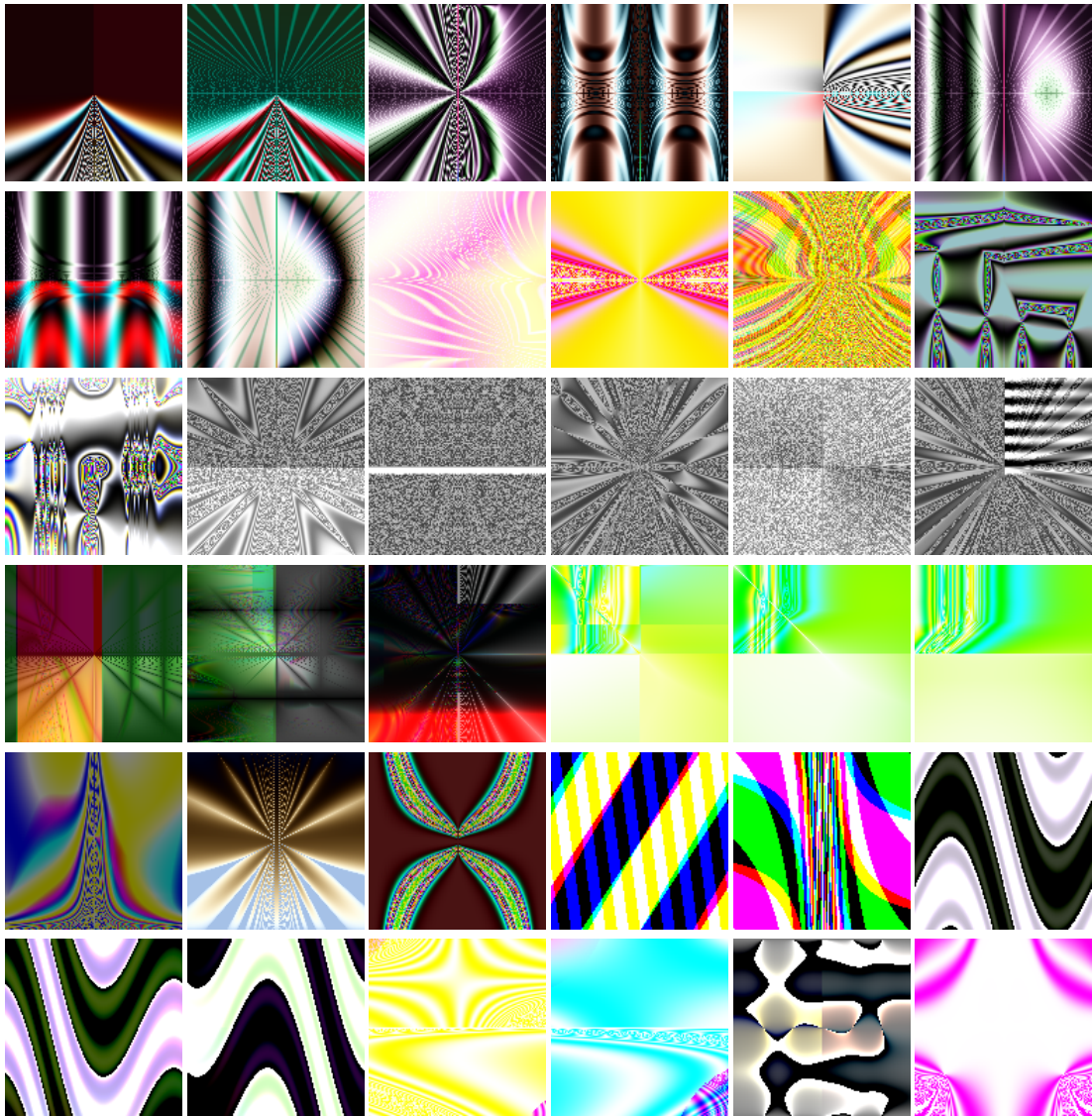


Figure 7.16: Samples of the images classified as external, generated throughout the course of the 30 evolutionary runs of the seventh iteration.

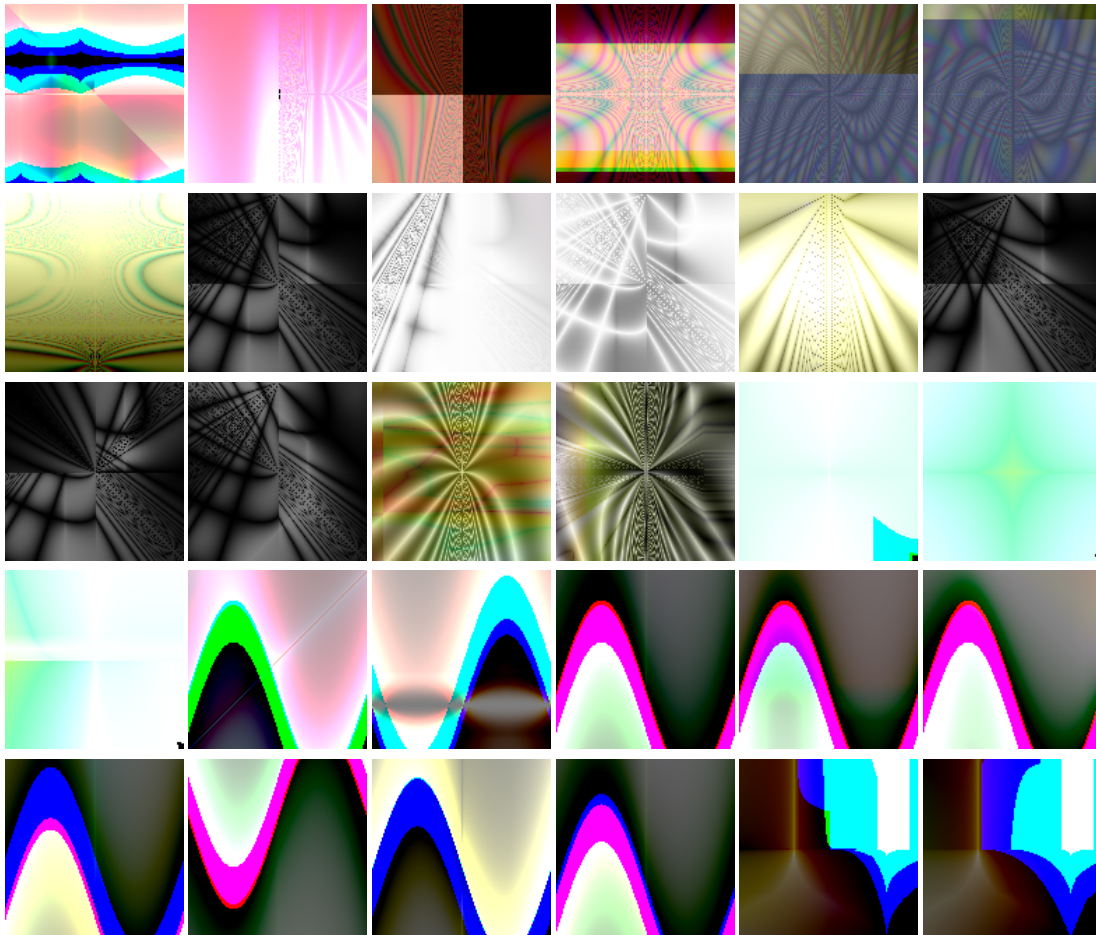


Figure 7.17: Samples of the images classified as external, generated throughout the course of the 30 evolutionary runs of the eighth iteration.

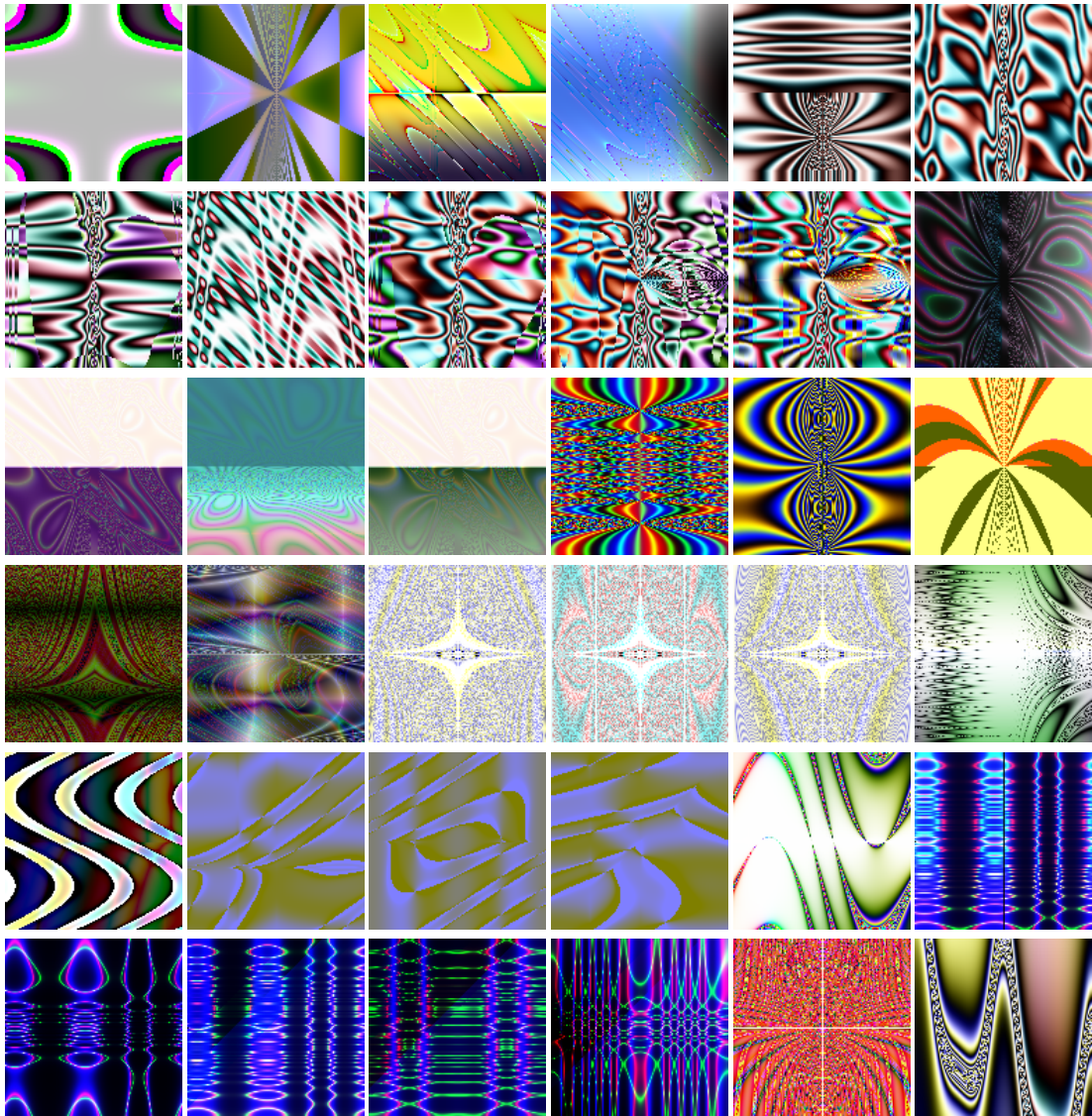


Figure 7.18: Samples of the images classified as external, generated throughout the course of the 30 evolutionary runs of the ninth iteration.

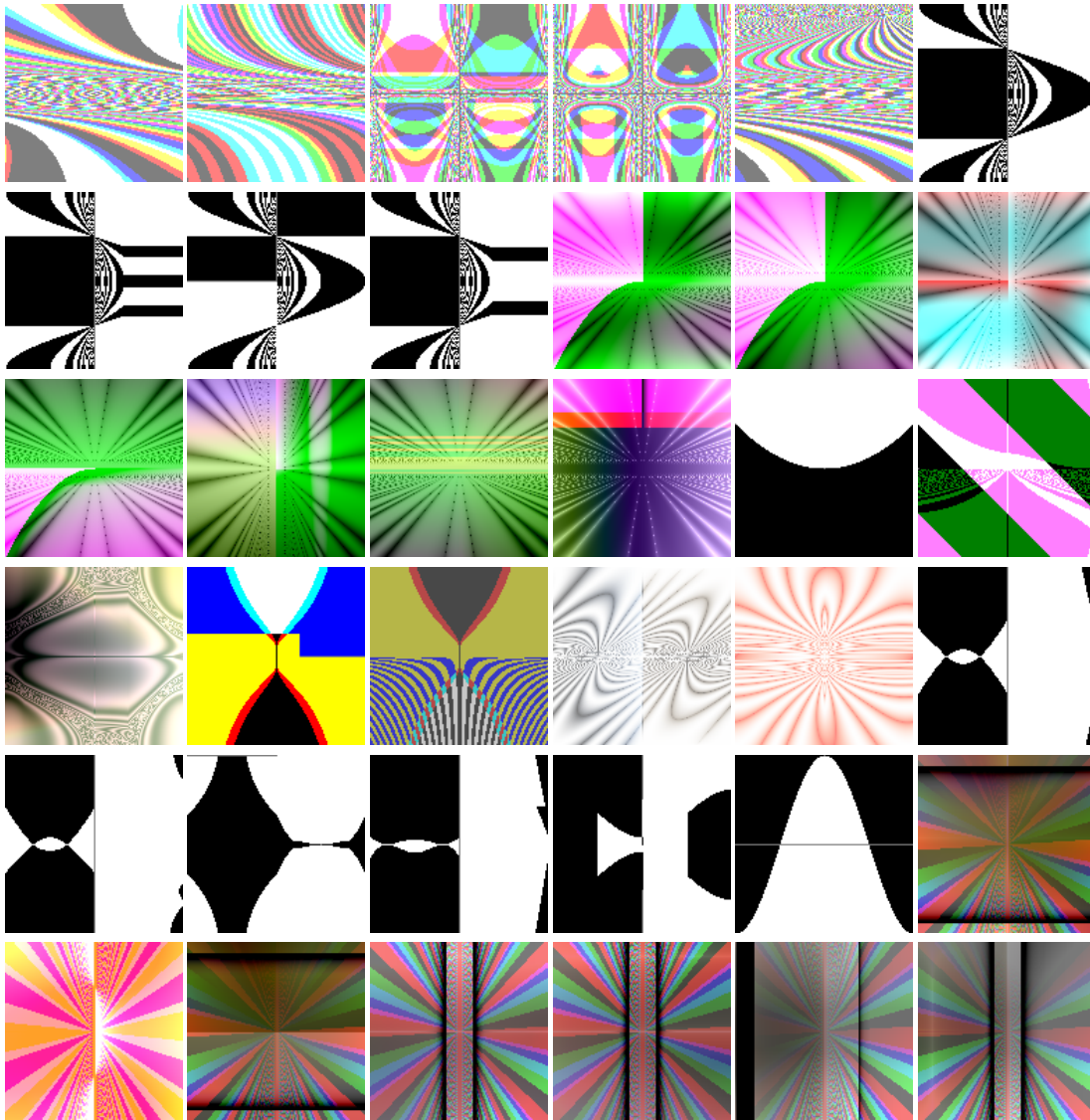


Figure 7.19: Samples of the images classified as external, generated throughout the course of the 30 evolutionary runs of the tenth iteration.

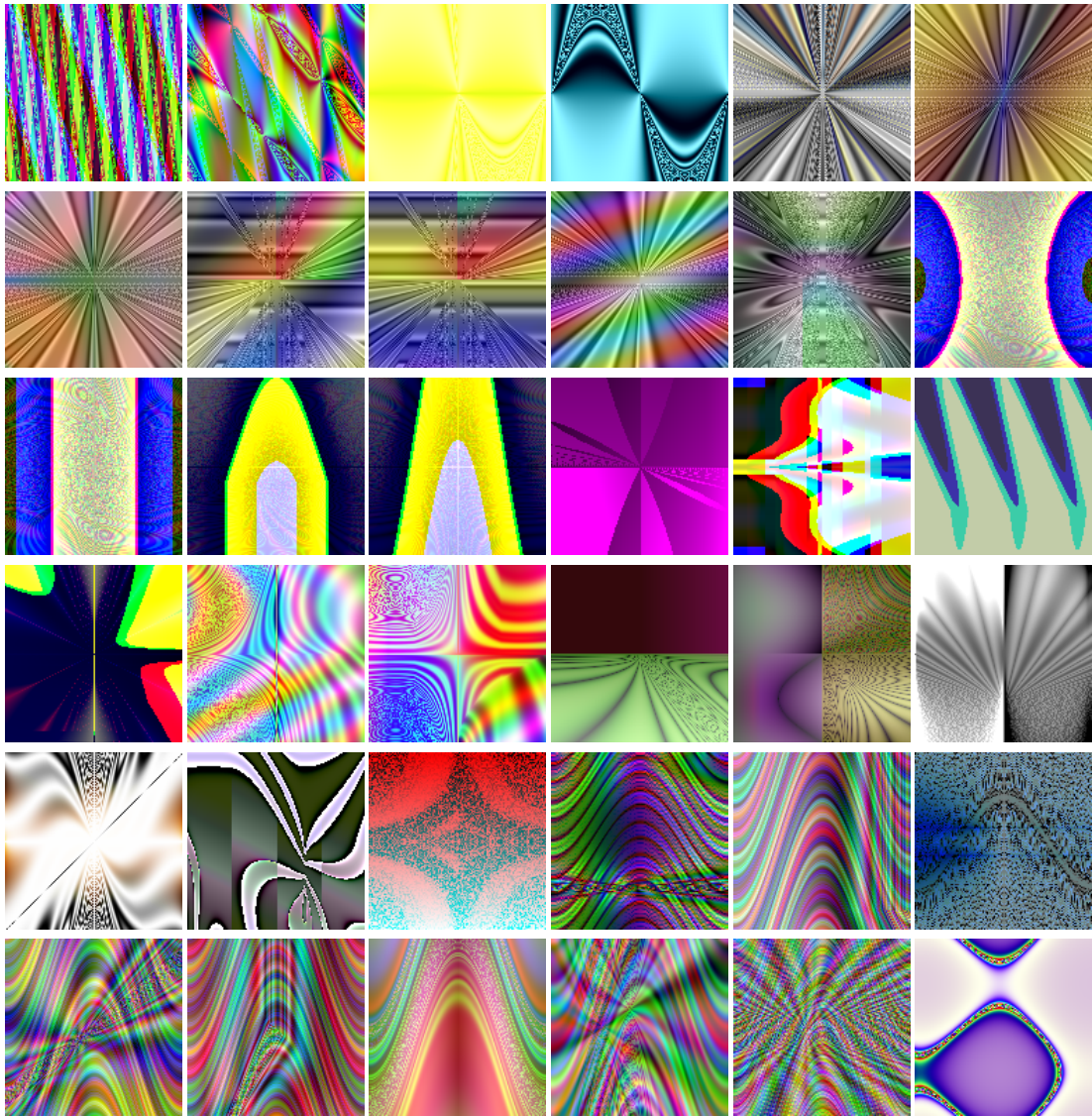


Figure 7.20: Samples of the images classified as external, generated throughout the course of the 30 evolutionary runs of the eleventh iteration.

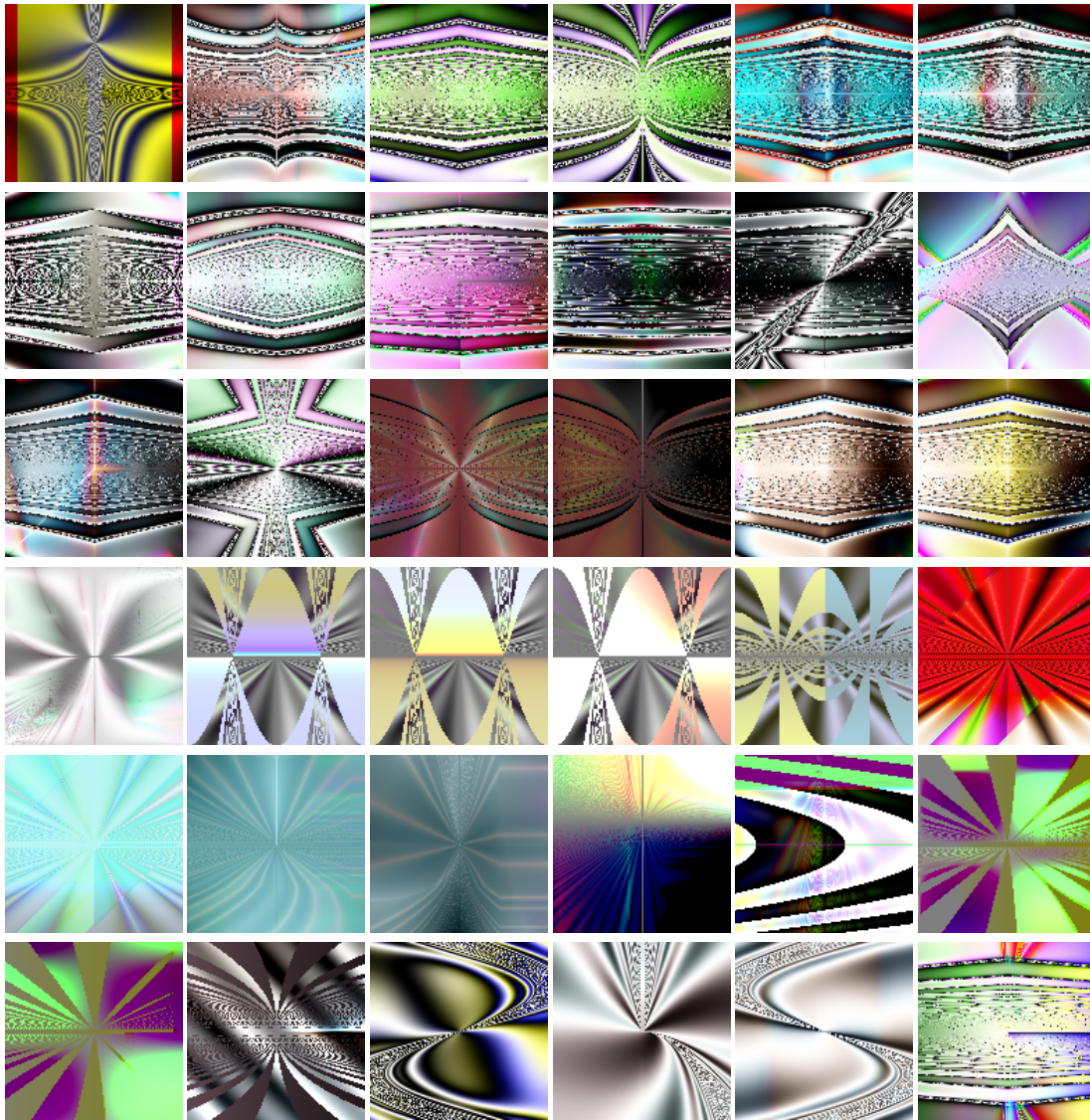


Figure 7.21: Samples of the images classified as external, generated throughout the course of the 30 evolutionary runs of the twelfth iteration.

As we will see in the next Section, while the increase of productivity in the sixth iteration seems to be linked with shortcomings of the classifier, here it appears to be linked with significant changes to the way the classifier system differentiates between the class of internal and external imagery. Thus, while in the other intermediate iterations the AA seems to be making minor stylistic variations of images that it has already produced, and opportunistic exploitations of shortcomings of the classifier, what happens in the thirteenth iteration seems to be rather different, resulting from profound changes of the aesthetic model, caused by the cumulative revision of the internal and external set. Making an analogy, this can be seen as an “Eureka” moment, where the system discovers substantially different styles, expanding and enriching the range of its artistic production.

As it can be observed by the sample of the images presented in Figure 7.22, although there are some recurring themes, the detail of “execution” of the images of the thirteenth generation classified as external is a lot higher than in previous iterations. The images seem to be more elaborate, detailed, and refined when compared with previous iterations. At the same time, some novel ornamentation techniques, such as the one depicted in the three rightmost images of the first column, were discovered, and some novel themes seem to emerge. The exploration of “light” (see, e.g. the leftmost image of the third row and the rightmost image of the fifth row) also emerges as visible and distinctive traits.

We considered the images resulting from the first iteration comparable to the ones evolved through user-guided evolution. Although in fairness, the same could be stated for a significant portion of the images evolved in the course of the thirteenth, it is equally fair to state that some of the runs created imagery that is stylistic dissimilar from what we have evolved through user-guided evolution or other means. Thus, many of these images strike us not only as novel when compared with the previous artistic production of the AA but also as novel and surprising with our own experience and production.

7.2.3 Classifier’s Training Results

In this Subsection, we make an overview of the results pertaining the training of the classifier. As previously mentioned (see Section 7.1.1), when an iteration has concluded the images classified as external, evolved in the course of the iteration, they are gathered by the supervisor and added to the internal dataset. This is followed by several training attempts, which may imply removing images from the external dataset. Training is concluded when no external images are classified as being internal. The existence of internal images classified as external implies that the EC engine may revisit previous styles, but, when this occurs, the consequent increase of the number of images of those styles present in the internal dataset will eventually force the classifier to recognise such images as internal.

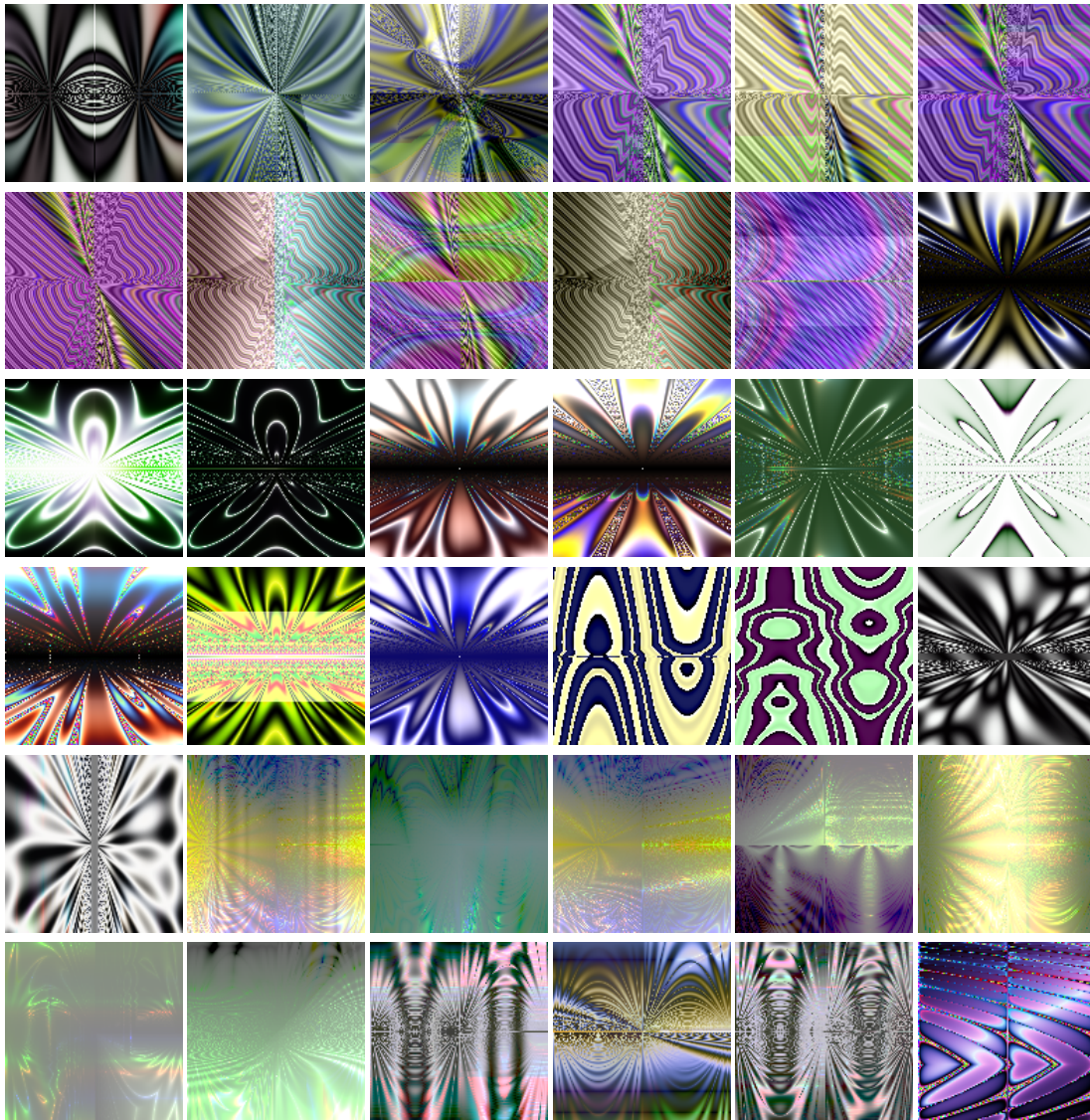


Figure 7.22: Sample of images classified as external, generated throughout the course of the 30 evolutionary runs of the thirteenth iteration.

Table 7.6 presents a summary of pertinent statistics regarding the training phase. It details, per iteration: the number of training attempts necessary to reach a classifier without false internals (*attempts*); the total number of false externals identified in the course of these attempts (*False Externals*); the number of false externals and internals after the training attempts are concluded (*CS False External* and *CS False Internal*, respectively), which reflects the ability of the classifier that is going to be used to guide the following iteration to discriminate between the sets; the size of the external and internal datasets after training is concluded (*Total External* and *Total Internal*).

As it can be observed, the training of the classifier for the first iteration required two attempts. One external image - a black and white photograph of a detail of a painting - was removed from the external dataset. Unfortunately, due to copyright issues, this and other external images cannot be replicated in the Chapter. After the second attempt, no external images were classified as internal, but one internal image, a black and white star-like shape was deemed as external. The existence of this image, and the difficulty in classifying it may, at least partially, explain the recurrence of such theme in several iterations.

As previously mentioned, the first iteration generated a wide and varied number of images. As a consequence, 30110 images have been added to the internal dataset and the training of the classifier that guided the second iteration took a significantly larger number of attempts; in total 68 external images have been removed. These are mostly black and white engravings, and quite interestingly, some images of mathematical objects, an artwork of M. C. Escher, which also has a mathematical appearance, and a cartoon image. In what concerns the engravings, we believe that these images were removed by two main reasons: (i) with the resolution that the EE processes these images, they could easily be confused with images produced by the EC engine; (ii) more importantly, these images tend to be atypical in relation to the other images belonging to the external dataset, which makes them harder to classify. In what concerns the images of mathematical objects, they seem to be computer-generated and, therefore, the confusion with the images produced by the AA is natural. After the removal of these images, the classifier can completely distinguish between the two sets.

The same overall trend occurs in iterations 2 to 3, although the number of attempts varies (4 and 7, respectively) the types of external images being excluded is the same, including engravings, M. C. Escher artworks, black and white drawings, photographs of sculptures, and minimalistic paintings (some of them by Kazimir Malevich). The reasons for their misclassification are the same: they are either atypical in relation with the rest of the external set or, confusable with computer-generated imagery, i.e., similar in style with the images the EC engine is prone to create.

The fourth generation provoked few changes on the external dataset, removing only 11 images. While 10 of these are black and white drawings, the remaining one is notable since it is the first Mondrian removed from the external set. The images produced in the fifth generation provoked the exclusion of 18 images from the external dataset,

among which two by Matisse and two by M.C. Escher. The most relevant issue concerning the training after the fifth iteration is that the resulting classifier, which will guide evolution in the sixth iteration, misclassifies 28 internal images. This gives the EC engine a large degree of freedom to explore previously visited imagery, which explains the burst of productivity observed in the sixth iteration. The exploitation of these “shortcomings” leads to the generation of images that, once added to the internal dataset prevent their future exploitation.

In the seventh iteration, a total of 248 external images were removed from the external dataset, these include Black and white engravings, several M. C. Escher artworks (14 to be precise), several Mondrian paintings, and numerous line drawings. The eighth iteration, the least productive of all, provoked the removal of 31 external images, that tend to be of the same type as the ones previously identified. After these removals, the classifier is, again, able to fully distinguish between the two sets.

The ninth and tenth iterations caused the removal of few external images, 8 and 6, respectively. Confirming our previous observation that, although these iterations were productive, they were not particularly fruitful regarding the novelty of the evolved imagery. As mentioned previously, although 692 images classified as external were evolved in the tenth iteration, only 62 of these made it to the archive.

These numbers contrast with the ones of the eleventh and twelfth iteration where, respectively, 47 and 63 were deleted. These include several Picasso, Dalí, Paul Klee, Mondrian, and Mark Rothko paintings, as well as several line drawings. The lack of texture, at low resolution, appears to be the binding trait of these paintings.

As mentioned previously, in our opinion, the thirteenth iteration is different from the others in the sense it corresponds to a pronounced shift in style. As such, it is particularly interesting to inspect what kind of changes to the external dataset the evolved imagery induces. In total 64 external images were removed. In previous iterations most of the removed images were black and white drawings or engravings, this is not the case in the thirteenth generation; only ten of the deleted images are black and white. The remaining images are Renaissance style paintings (unfortunately, they do not include the Mona Lisa), two Van Gogh paintings, three by Kandinsky and one by Miró. Quite interestingly, four paintings of Monet’s Waterloo Bridge, a theme that is present in several of his artworks, were also removed. In this case, it was possible - and quite easy we might add - to identify the evolved images that promote the confusion between internal images and these artworks. Some of them are depicted in the bottom two rows of Figure 7.22, and we believe that the reader will also understand why, in the eyes of the classifier, they can be easily confused.

7.3 SUMMARY

In this Chapter, we presented an Artificial Artist that is characterised by its permanent quest for novelty. The system is composed of two main modules: a creator and a critic.

Table 7.6: Statistics regarding the training of the classifiers of each iteration in terms of: number of attempts, false internals and externals during training, false externals after training, and size of the internal and external and internal dataset after training.

Iteration	Attempts	During Training Cycles		After Training Attempts			
		False External	False Internal	CS False Externals	CS False Internals	Total Externals	Total Internal
initial	2	1	1	1	0	26238	26239
1	7	23	68	0	0	26170	56349
2	4	52	20	3	0	26150	56599
3	7	41	67	3	0	26083	56794
4	6	7	11	3	0	26072	56972
5	2	40	18	28	0	26054	57029
6	2	7	18	3	0	26036	57462
7	3	11	248	3	0	25788	57593
8	3	23	31	0	0	25757	57624
9	3	11	8	5	0	25749	57739
10	3	126	6	5	0	25743	57801
11	8	145	47	4	0	25696	57927
12	6	99	63	7	0	25633	58028
13	4	88	64	3	0	25589	58380

The role of the creator is played by an expression based evolutionary engine and that of the critic by an **ANN**. The **ANN** is trained to discriminate among the images produced by the evolutionary engine and a set of famous artworks. The fitness of the images being evolved depends on the output of the classifier, promoting the discovery of images that the network classifies as external. These two components make part of an **EFFECTIVE** instantiation, and on each iteration of the framework, we perform 30 evolutionary runs. When these are concluded, the Supervisor picks relevant misclassified images that are added to the set representing the production of the **AA** and the **ANN** is retrained.

The **EFFECTIVE** instantiation of this Chapter promotes and explores a competition between creator and critic. From a theoretical standpoint – assuming that the **EC** engine and the **ANN** are adequate and always able to cope – the iterative expansion of the internal set leads, necessarily, to change since the evolutionary algorithm is forced to explore new paths. Moreover, assuming that a sufficiently large number of iterations is performed and that both systems cope, the convergence to the aesthetic model (or models) implicitly defined by the set of external images, which provides an aesthetic reference to the artistic production of the **AA**, is bound to occur eventually.

To increase the diversity within evolutionary runs and prevent their early convergence and stagnation, we include mechanisms to promote the phenotype diversity of the populations. This implies taking two criteria into account when performing tournament selection: the adequacy of the image (which results from the output of the neural network, i. e., the critic) and its diversity about the images produced in the course of the same run.

The analysis of the experimental results confirms the adequacy and potential of **EFFECTIVE**, revealing that the system can consistently produce novel imagery of arguably, aesthetic merit. As such, we consider that we successfully developed a creative system that can learn, create and innovate in an entirely autonomous way.

The experimental results indicate that the images produced in the course of the first iteration of the framework are similar to those produced by expression-based interactive evolutionary art systems, where the role of the evaluator is played by a human. Analysing the results, we consider that the behaviour and production of the system during the first twelve iterations can be considered as e-creative. We find that what happens during the thirteenth generation is significantly different and goes beyond e-creativity. In this case, the system made a qualitative and substantial change both regarding production and aesthetic model, thus breaking the mould. We put forward the hypothesis that this behaviour can be seen as a limited case of h-creativity – in the sense that the system produced images that appear to be different from those previously attained by evolutionary means – and of t-creativity, in the sense that the changes seem to be related to profound changes in the aesthetic model and, therefore, to a profound transformation of the search space.

CONCLUSIONS AND FUTURE WORK

The working hypothesis for this thesis is that **EC** can be used to assess and improve the performance of classifiers by evolving new training instances. The main contribution is a framework for the assessment and improvement of classifiers, which was experimentally validated. The definition of the framework, named **EFFECTIVE** is described in Chapter 3. It combines three key modules: a **CS** module which represents the **ML** approach to be improved; an **EC** engine responsible for generating instances; and a Supervisor module responsible for managing the instances generated. The combination of these modules create an automatic, iterative process of assessment and improvement of classifiers.

One of the starting points of this thesis was the exploration of the idea by Romero et al. (2003): combining a general purpose evolutionary art system with an image classifier trained to detect faces, or other types of objects, to evolve images of a particular type. Based on the literature concerning Evolutionary Art systems, it is theoretically possible to achieve this by using proper representation and fitness assignment. We pursue this idea by using an evolutionary art tool combined with a face detector to evolve frontal faces in Chapter 4. However, we encountered some interesting and surprising results. The classifier, in some cases, detects faces on images that do not resemble, from a human perspective, a face. This result led to a research opportunity of using these cases to improve the face detector by adding them to the training dataset.

Before proposing the framework, we performed a survey on the existing techniques for dataset construction, more precisely on instance gathering, **IS** and **IG**. The survey of Chapter 2 led us to identify the most relevant concepts and aspects of the process dataset construction, and also opportunities to be explored.

In a first phase, we explore image generation. In Chapter 4, we tested **EFFECTIVE** in several image generation tasks: (i) evolution of faces; (ii) evolution of figurative images; (iii) evolution of ambiguous images; and (iv) evolution of photorealistic faces. The evolution of frontal faces confronted us with the research question of how to create a suitable fitness function. We used the internal values of the classifier to build the fitness function. In the evolution of faces experiment we used an off-the-shelf classifier combined with a **GP** based **EC** engine. The approach evolved images that were classified as faces and were evocative of human faces. However, most of the generated images did not resemble a face, although they were classified as faces by the classifier. This particular result suggested that the approach could be suitable to generate false positives. In the evolution of figurative images, we explored the application to other types of objects and created our own object detectors. The results showed that

we were still able to generate instances evocative of the target object. Once again, most of the results were classified as containing the target object but did not, from a human perspective, resemble the object. In the experiment of evolution of ambiguous images, we had the objective of creating images that induce multistable perception, by presenting an ambiguous stimulus. To attain this, we evolved images where the classifiers detected more than one type of object in the same area. In the last experiment of this Chapter, we wanted to test if we could evolve false negatives. We created an **EC** engine with a representation that allowed us to generate photorealistic faces consistently. Afterwards, we design a fitness function that allowed us to generate images that, from a human perspective, are faces and that the classifier does not classify as faces. Thus these experiments in conjunction, allowed us to show that we are able to generate false positives and false negatives with **EFFECTIVE**. Based on the aforementioned results, Chapter 4 answers the question “How to create a suitable fitness function?”.

The results obtained in Chapter 4 show that we are able to generate instances that are misclassified by the classifiers. In Chapter 5, we move towards the improvement of classifiers by performing a set of experiments where we use the misclassified instances to improve their performance. In a first test, as proof of concept, we assess the performance of classifiers trained with the contribution of instances from one **EC** run, which evolves false positives. The classifiers trained with the instances from one **EC** run revealed improvement in its performance, reducing the number of FAs when tested in a group of test datasets. Afterwards, instead of using a single **EC** run, we performed experiments where we aggregated the instances of several **EC** runs for the classifier’s dataset. We tested the usage of the Supervisor module with different strategies for filtering and selecting instances. For selection we tested the following: adding all the misclassified instances (*Aggregator*); using an external classifier to select instances that it classifies correctly (*External*); and manual selection of individuals by a user (*Manual*). The filter methods considered were: discarding images that are equal (*Unequal*); and discarding similar images using a pixel-based root mean squared error as the similarity measure (*RMSE*). The results revealed that the strategy for the Supervisor had impact on the performance of the classifiers, showing that the *External RMSE* yielded results as good as the *Manual* selection strategy. At this point we also concluded that it could be advantageous to perform several iterations of the framework. The question “How should the Supervisor select and filter individuals per **EC** run?” was partially answered here.

In Chapter 6 we started by performing several framework iterations evolving instances that are classified as false positives, expanding the negative dataset at each framework iteration. We also tested the impact of using several framework iterations. Several setups were performed based on the type of Supervisor in use. The results show that by expanding the negative dataset, we are able to reduce the number of FAs. Furthermore, we tested if the evolved instances had impact in the framework by comparing our approach with a commonly used method for **DA**. We concluded that

the classifiers from **EFFECTIVE** obtained the highest results for every test dataset, when compared with the classifiers trained using the **DA** approach. Afterwards, we tested evolving instances classified as false negatives, expanding the positive dataset. The corresponding results show that the system increased the %H. In a last experiment, we expanded the negative and positive datasets in the same framework iteration. The results show that with the addition of the false negative and false positive instances, we can reduce the FAs while increasing the %H, which results in an overall increase of %C. Using this method, we obtained the best performance values when compared with the baseline classifier. Based on the experiments and on the corresponding results, this Chapter answers the following questions: “How should the Supervisor select and filter individuals per **EC** run?”; “How to compensate the unbalance that will be created?”. The questions “What are the necessary conditions for the framework to succeed?” is partially answered sustained by the experimental results of this Chapter.

In Chapter 7 we tested **EFFECTIVE** in a **CC** context to create a system that promotes novelty and style change in the generation of images. The analysis of the experimental results confirms the adequacy and potential of **EFFECTIVE**, revealing that the system can consistently produce novel imagery of arguably, aesthetic merit. As such, we consider that we successfully developed a creative system that can learn, create and innovate in an entirely autonomous way.

In what concerns the scientific dissemination of the results, the work conducted in the Chapter 4 regarding generation of images of a particular type, resulted in the following publications:

- Penousal Machado, João Correia, and Juan Romero (2012a). ‘Expression-Based Evolution of Faces.’ In: *Evolutionary and Biologically Inspired Music, Sound, Art and Design - First International Conference, EvoMUSART 2012, Málaga, Spain, April 11-13, 2012. Proceedings*. Vol. 7247. Lecture Notes in Computer Science. Springer, pp. 187–198. DOI: [10.1007/978-3-642-29142-5_17](https://doi.org/10.1007/978-3-642-29142-5_17)
- João Correia, Penousal Machado, Juan Romero, and Adrián Carballal (2013a). ‘Evolving Figurative Images Using Expression-Based Evolutionary Art.’ In: *Proceedings of the fourth International Conference on Computational Creativity (ICCC)*, pp. 24–31
- Penousal Machado, Adriano Vinhas, João Correia, and Anikó Ekárt (2015b). ‘Evolving Ambiguous Images.’ In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*. Ed. by Qiang Yang and Michael Wooldridge. AAAI Press, pp. 2473–2479. URL: <http://ijcai.org/papers15/Abstracts/IJCAI15-350.html>
- Penousal Machado, Adriano Vinhas, João Correia, and Anikó Ekárt (2015c). ‘Evolving Ambiguous Images.’ In: *AI Matters 2.1*, pp. 7–8. ISSN: 2372-3483. DOI: [10.1145/2813536.2813539](https://doi.org/10.1145/2813536.2813539) URL: <http://doi.acm.org/10.1145/2813536.2813539>

- João Correia, Tiago Martins, Pedro Martins, and Penousal Machado (2016). 'X-Faces: The eXploit Is Out There.' In: *Proceedings of the Seventh International Conference on Computational Creativity (ICCC 2016)*. Ed. by François Pachet, Amílcar Cardoso, Vincent Corruble, and Fiammetta Ghedini. Sony CSL Paris, France, pp. 164–182

Next, concerning the improvement of classifiers, the following articles were published:

- Penousal Machado, João Correia, and Juan Romero (2012b). 'Improving Face Detection.' In: *Genetic Programming - 15th European Conference, EuroGP 2012, Málaga, Spain, April 11-13, 2012. Proceedings*. Ed. by Alberto Moraglio, Sara Silva, Krzysztof Krawiec, Penousal Machado, and Carlos Cotta. Vol. 7244. Lecture Notes in Computer Science. Springer, pp. 73–84. DOI: [10.1007/978-3-642-29139-5_7](https://doi.org/10.1007/978-3-642-29139-5_7)
- João Correia, Penousal Machado, and Juan Romero (2012). 'Improving haar cascade classifiers through the synthesis of new training examples.' In: *Genetic and Evolutionary Computation Conference, GECCO '12, Philadelphia, PA, USA, July 7-11, 2012, Companion Material Proceedings*. Ed. by Terence Soule and Jason H. Moore. ACM, pp. 1479–1480

The work discussed in Chapter 7 led to the following publications:

- João Correia, Penousal Machado, Juan Romero, and Adrián Carballal (2013b). 'Feature Selection and Novelty in Computational Aesthetics.' In: *Evolutionary and Biologically Inspired Music, Sound, Art and Design - Second International Conference, EvoMUSART 2013, Vienna, Austria, April 3-5, 2013. Proceedings*. Ed. by Penousal Machado, James McDermott, and Adrián Carballal. Vol. 7834. Lecture Notes in Computer Science. Springer, pp. 133–144. ISBN: 978-3-642-36954-4
- Penousal Machado, Juan Romero, Marcos Nadal, Antonino Santos, João Correia, and Adrián Carballal (2015a). 'Computerized measures of visual complexity.' In: *Acta Psychologica* 160, pp. 43–57. ISSN: 0001-6918. DOI: <http://dx.doi.org/10.1016/j.actpsy.2015.06.005>. URL: <http://www.sciencedirect.com/science/article/pii/S0001691815300160>
- Adriano Vinhas, Filipe Assunção, João Correia, Aniko Ekárt, and Penousal Machado (2016). 'Fitness and Novelty in Evolutionary Art.' In: *Evolutionary and Biologically Inspired Music, Sound, Art and Design - Fifth International Conference, EvoMUSART 2016, Porto, Portugal, March 30 – April 1, 2016, Proceedings*. Ed. by Colin Johnson, Vic Ciesielski, João Correia, and Penousal Machado. Cham: Springer International Publishing, pp. 225–240. ISBN: 978-3-319-31008-4. DOI: [10.1007/978-3-319-31008-4_16](https://doi.org/10.1007/978-3-319-31008-4_16). URL: http://dx.doi.org/10.1007/978-3-319-31008-4_{_}16
- João Correia, Penousal Machado, Juan Romero, Pedro Martins, and F Amílcar Cardoso (2017). 'Breaking the Mould: An Evolutionary Quest for Innovation

Through Style Change.’ In: *Computational Creativity: The Philosophy and Engineering of Autonomously Creative Systems*. Ed. by Tony Veale and F Amílcar Cardoso. Springer International Publishing. ISBN: 978-3-319-43608-1

Furthermore, the following publications resulted from ramifications of the research conducted in this thesis:

- Penousal Machado and João Correia (2014). ‘Semantic Aware Methods for Evolutionary Art.’ In: *Genetic and Evolutionary Computation Conference, GECCO ’14, Vancouver, BC, Canada, July 12-16, 2014. Proceedings*. ACM
- Tiago Martins, João Correia, Ernesto Costa, and Penousal Machado (2015). ‘Evo-type: Evolutionary Type Design.’ In: *Evolutionary and Biologically Inspired Music, Sound, Art and Design - Fourth International Conference, EvoMUSART 2015, Copenhagen, Denmark, April 8-10, 2015, Proceedings*. Ed. by Colin G. Johnson, Adrián Carballeda, and João Correia. Vol. 9027. Lecture Notes in Computer Science. Springer, pp. 136–147. DOI: [10.1007/978-3-319-16498-4_13](https://doi.org/10.1007/978-3-319-16498-4_13). URL: https://doi.org/10.1007/978-3-319-16498-4_13
- Tiago Martins, João Correia, Ernesto Costa, and Penousal Machado (2016). ‘Evo-type: From Shapes to Glyphs.’ In: *Proceedings of the Genetic and Evolutionary Computation Conference 2016. GECCO ’16*. New York, NY, USA: ACM, pp. 261–268. ISBN: 978-1-4503-4206-3. DOI: [10.1145/2908812.2908907](https://doi.org/10.1145/2908812.2908907). URL: <http://doi.acm.org/10.1145/2908812.2908907>
- Tiago Martins, João Correia, Ernesto Costa, and Penousal Machado (2018). ‘Evo-type: Towards the Evolution of Type Stencils.’ In: *Computational Intelligence in Music, Sound, Art and Design - Seventh International Conference, EvoMUSART 2018, Parma, Italy, April 4-6, 2018, Proceedings*. Ed. by Antonios Liapis, Juan Jesús Romero Cardalda, and Anikó Ekárt. Vol. 10783. Lecture Notes in Computer Science. Springer, pp. 299–314. DOI: [10.1007/978-3-319-77583-8_20](https://doi.org/10.1007/978-3-319-77583-8_20). URL: https://doi.org/10.1007/978-3-319-77583-8_20

The experiments described in Chapter 6 resulted in a journal paper that is currently under review. As for future work in the domain of improving the classifiers performance, the immediate step is to compare EFFECTIVE with other methods, e. g., GANs or other type of adversarial learning. Furthermore, we want to further explore the impact of the Supervisor, e.g. use the Supervisor in other types of adversarial learning.

Currently, the EFFECTIVE framework operates at the end of the training. We can introduce the evolved instances in the training phase, e. g., add instances to the training batch and use the classifier that is being trained to assign fitness. We also plan to explore the usage of DL classifiers as the CS module and as the Supervisor.

The results that we attained in Chapter 7 using a tuned EC engine with multi-objective tournament selection and an archive for the generated individuals resulted

in interesting results, which we are going to use and build upon in future experiments. So, in the domain of the generation of images of a particular type, we are interested in using the algorithms from Chapter 7 in new experiments with object detectors using different ML approaches.

BIBLIOGRAPHY

- Ahonen, Timo, Abdenour Hadid, and Matti Pietikäinen (2006). 'Face Description with Local Binary Patterns: Application to Face Recognition.' In: *IEEE Trans. Pattern Anal. Mach. Intell.* 28.12, pp. 2037–2041 (cit. on pp. 65, 132).
- Alejo, Roberto, Vicente García, and J. Horacio Pacheco-Sánchez (2015). 'An Efficient Over-sampling Approach Based on Mean Square Error Back-propagation for Dealing with the Multi-class Imbalance Problem.' In: *Neural Processing Letters* 42.3, pp. 603–617 (cit. on p. 33).
- Almogahed, Bassam A. and Ioannis A. Kakadiaris (2015). 'NEATER: filtering of over-sampled data using non-cooperative game theory.' In: *Soft Comput.* 19.11, pp. 3301–3322 (cit. on p. 33).
- Back, Thomas, David B. Fogel, and Zbigniew Michalewicz, eds. (1997). *Handbook of Evolutionary Computation*. 1st. Bristol, UK, UK: IOP Publishing Ltd. (cit. on p. 2).
- Baluja, S, D Pomerlau, and J Todd (1994). 'Towards Automated Artificial Evolution for Computer-Generated Images.' In: *Connection Science* 6.2, pp. 325–354 (cit. on pp. 49, 57, 69, 72, 131).
- Bengio, Yoshua, Pascal Lamblin, Dan Popovici, and Hugo Larochelle (2007). 'Greedy Layer-Wise Training of Deep Networks.' In: *Advances in Neural Information Processing Systems*. Vol. 19. NIPS'06 1. Cambridge, MA, USA: MIT Press, p. 153 (cit. on p. 21).
- Beymer, David and Tomaso Poggio (1995). 'Face recognition from one example view.' In: *Computer Vision, 1995. Proceedings., Fifth International Conference on*. IEEE, pp. 500–507 (cit. on p. 38).
- Bhattacharya, Binay K., Ronald S. Poulsen, and Godfried T Toussaint (1992). 'Application of Proximity Graphs to Editing Nearest Neighbor Decision Rules.' In: *International Symposium on Information Theory*, pp. 1–25 (cit. on p. 27).
- Bishop, Christopher M (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc. (cit. on pp. 1, 23).
- Blum, Avrim L. and Pat Langley (1997). 'Selection of relevant features and examples in machine learning.' In: *Artificial Intelligence* 97.1-2, pp. 245–271 (cit. on pp. 24, 26).
- Boden, Margaret (2004). *The Creative Mind: Myths and Mechanisms*. Routledge (cit. on p. 172).
- Bonferroni, C. Emilio (1935). 'Il calcolo delle assicurazioni su gruppi di teste.' In: *Studi in Onore del Professore Salvatore Ortu Carboni*. Rome, pp. 13–60 (cit. on pp. 135, 151).
- Breiman, Leo (2001). 'Random Forests.' In: *Machine Learning* 45.1, pp. 5–32 (cit. on pp. 9, 15).

- Cameron-Jones, Mike (1995). 'Instance selection by encoding length heuristic with random mutation hill climbing.' In: *Proc. 8th Australian Joint Conf. Artificial Intelligence*, pp. 99–106 (cit. on p. 34).
- Cano, José Ramón, Francisco Herrera, and Manuel Lozano (2003). 'Using Evolutionary Algorithms as Instance Selection for Data Reduction in KDD : An Experimental Study.' In: *IEEE Transactions on Evolutionary Computation* 7.6, pp. 561–575 (cit. on p. 34).
- (2005). 'Stratification for scaling up evolutionary prototype selection.' In: *Pattern Recognition Letters* 26.7, pp. 953–963 (cit. on p. 27).
- Carlson, Andrew J, Chad M Cumby, Nicholas D Rizzolo, and Jeff L Rosen (2004). *SNoW user manual* (cit. on p. 10).
- Chang, Chin-Liang (1974). 'Finding Prototypes For Nearest Neighbor Classifiers.' In: *IEEE Trans. Comput.* 23.11, pp. 1179–1184 (cit. on p. 30).
- Chawla, Nv and Kw Bowyer (2002). 'SMOTE: Synthetic Minority Over-sampling Technique Nitesh.' In: *Journal of Artificial Intelligence Research* 16.1, pp. 321–357 (cit. on p. 32).
- Chen, C.H. and Adam Jóźwik (1996). 'A sample set condensation algorithm for the class sensitive artificial neural network.' In: *Pattern Recognition Letters* 17.8, pp. 819–823 (cit. on p. 29).
- Chen, Jie, Xilin Chen, and Wen Gao (2004). 'Resampling for face detection by self-adaptive genetic algorithm.' In: *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*. Vol. 3, 822–825 Vol.3 (cit. on p. 51).
- Chen, Jie, Ruiping Wang, Shengye Yan, Shiguang Shan, Xilin Chen, and Wen Gao (2007). 'Enhancing Human Face Detection by Resampling Examples Through Manifolds.' In: *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on* 37.6, pp. 1017–1028 (cit. on pp. 40, 51).
- Chen, Jie, Xilin Chen, Jie Yang, Shiguang Shan, Ruiping Wang, and Wen Gao (2009). 'Optimization of a training set for more robust face detection.' In: *Pattern Recognition* 42.11, pp. 2828–2840 (cit. on p. 51).
- Correia, João (2009). 'Evolutionary Computation for Assessing and Improving Classifier Performance.' MA thesis. Department of Informatics Engineering of the University of Coimbra (cit. on p. 178).
- Correia, João, Penousal Machado, and Juan Romero (2012). 'Improving haar cascade classifiers through the synthesis of new training examples.' In: *Genetic and Evolutionary Computation Conference, GECCO '12, Philadelphia, PA, USA, July 7-11, 2012, Companion Material Proceedings*. Ed. by Terence Soule and Jason H. Moore. ACM, pp. 1479–1480 (cit. on pp. 58, 218).
- Correia, João, Penousal Machado, Juan Romero, and Adrián Carballal (2013a). 'Evolving Figurative Images Using Expression-Based Evolutionary Art.' In: *Proceedings of the fourth International Conference on Computational Creativity (ICCC)*, pp. 24–31 (cit. on pp. 50, 61, 69, 82, 90, 91, 93, 217).

- (2013b). ‘Feature Selection and Novelty in Computational Aesthetics.’ In: *Evolutionary and Biologically Inspired Music, Sound, Art and Design - Second International Conference, EvoMUSART 2013, Vienna, Austria, April 3-5, 2013. Proceedings*. Ed. by Penousal Machado, James McDermott, and Adrián Carballal. Vol. 7834. Lecture Notes in Computer Science. Springer, pp. 133–144 (cit. on pp. 50, 171, 172, 176–178, 218).
- Correia, João, Tiago Martins, Pedro Martins, and Penousal Machado (2016). ‘X-Faces: The eXploit Is Out There.’ In: *Proceedings of the Seventh International Conference on Computational Creativity (ICCC 2016)*. Ed. by François Pachet, Amílcar Cardoso, Vincent Corruble, and Fiammetta Ghedini. Sony CSL Paris, France, pp. 164–182 (cit. on pp. 61, 99, 100, 218).
- Correia, João, Penousal Machado, Juan Romero, Pedro Martins, and F Amílcar Cardoso (2017). ‘Breaking the Mould: An Evolutionary Quest for Innovation Through Style Change.’ In: *Computational Creativity: The Philosophy and Engineering of Autonomously Creative Systems*. Ed. by Tony Veale and F Amílcar Cardoso. Springer International Publishing (cit. on pp. 50, 171, 218).
- Cortes, Corinna and Vladimir Vapnik (1995). ‘Support-Vector Networks.’ In: *Machine Learning* 20.3, pp. 273–297 (cit. on pp. 12, 13, 32).
- Cruz, António, Penousal Machado, Filipe Assunção, and António Leitão (2015). ‘ELICIT: Evolutionary Computation Visualization.’ In: *Genetic and Evolutionary Computation Conference, GECCO 2015, Madrid, Spain, July 11-15, 2015, Companion Material Proceedings*. Ed. by Juan Luis Jiménez Laredo, Sara Silva, and Anna Isabel Esparcia-Alcázar. ACM, pp. 949–956 (cit. on p. 72).
- Darwin, Charles (1859). *On the origin of species*. London: John Murray (cit. on p. 1).
- Datta, Ritendra, Dhiraj Joshi, Jia Li, and James Ze Wang (2006). ‘Studying Aesthetics in Photographic Images Using a Computational Approach.’ In: *Computer Vision – ECCV 2006, 9th European Conference on Computer Vision, Part III*. LNCS. Graz, Austria: Springer, pp. 288–301 (cit. on pp. 176, 177).
- Datta, Ritendra, Dhiraj Joshi, Jia Li, and James Z Wang (2008). ‘Image retrieval: Ideas, influences, and trends of the new age.’ In: *ACM Comput. Surv.* 40.2, 5:1–5:60 (cit. on p. 176).
- David E. Rumelhart, James L. McClelland (1986). ‘Parallel distributed processing : explorations in the microstructure of cognitio.’ In: ed. by David E Rumelhart, James L McClelland, and CORPORATE PDP Research Group. Cambridge, MA, USA: MIT Press. Chap. Informatio, pp. 194–281 (cit. on p. 21).
- Davison, A C and D V Hinkley (1997). *Bootstrap Methods and their Application (Cambridge Series in Statistical and Probabilistic Mathematics)*. 1st ed. Cambridge University Press (cit. on p. 36).
- Dempster, A.P., N.M. Laird, and Donald B Rubin (1977). ‘Maximum likelihood from incomplete data via the EM algorithm.’ In: *Journal of the Royal Statistical Society Series B Methodological* 39.1, pp. 1–38 (cit. on p. 36).

- Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei (2009). 'ImageNet: A Large-Scale Hierarchical Image Database.' In: *CVPR09* (cit. on p. 23).
- Derrac, Joaquín, Isaac Triguero, Salvador García, and Francisco Herrera (2012). 'Integrating instance selection, instance weighting, and feature weighting for nearest neighbor classifiers by coevolutionary algorithms.' In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* 42.5, pp. 1383–1397 (cit. on p. 34).
- Drucker, Harris, Corinna Cortes, L.D. Jackel, Yann LeCun, and Vladimir Vapnik (1994). 'Boosting and Other Machine Learning Algorithms.' In: *Machine Learning Proceedings 1994*, pp. 53–61 (cit. on pp. 24, 33).
- Drummond, Chris and R C Holte (2003). 'C4.5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling.' In: *Workshop on Learning from Imbalanced Datasets II*, pp. 1–8 (cit. on p. 32).
- Dyk, David a van and Xiao-Li Meng (2001). 'The Art of Data Augmentation.' In: *Journal of Computational and Graphical Statistics* 10.1, pp. 1–50 (cit. on p. 36).
- Eiben, A. E. and J. E. Smith (2003). *Introduction to Evolutionary Computing*. Natural Computing Series. Berlin, Heidelberg: Springer Berlin Heidelberg (cit. on pp. 2, 56).
- Faria, João, Stanislav Bagley, Stefan Ruger, and Toby Breckon (2013). 'Challenges of finding aesthetically pleasing images.' In: *Image Analysis for Multimedia Interactive Services (WIAMIS), 2013 14th International Workshop on*. IEEE, pp. 1–4 (cit. on pp. 176, 177).
- Fei-Fei, Li, Rob Fergus, and Pietro Perona (2004). 'Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories.' In: *2004 Conference on Computer Vision and Pattern Recognition Workshop*, p. 178 (cit. on pp. 82, 84).
- Frank, A. and A. Asuncion (2010). *UCI Machine Learning Repository* (cit. on p. 39).
- Freund, Yoav and Robert E Schapire (1997). 'A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting.' In: *Journal of Computer and System Sciences* 55.1, pp. 119–139 (cit. on pp. 13, 24, 33, 62).
- Fu, Yifan, Xingquan Zhu, and Bin Li (2013). 'A survey on instance selection for active learning.' In: *Knowledge and Information Systems* 35.2, pp. 249–283 (cit. on p. 25).
- Fukushima, Kunihiko (1980). 'Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position.' In: *Biological cybernetics* 36.4, pp. 193–202 (cit. on p. 15).
- García-Pedrajas, Nicolás and Aida De Haro-García (2014). 'Boosting instance selection algorithms.' In: *Knowledge-Based Systems* 67, pp. 342–360 (cit. on p. 25).
- García-Pedrajas, Nicolás, Javier Pérez-Rodríguez, and Aida De Haro-García (2013). 'OligoIS: Scalable instance selection for class-imbalanced data sets.' In: *IEEE Transactions on Cybernetics* 43.1, pp. 332–346 (cit. on p. 27).

- García, Salvador and Francisco Herrera (2009). 'Evolutionary undersampling for classification with imbalanced datasets: proposals and taxonomy.' In: *Evolutionary computation* 17.3, pp. 275–306 (cit. on p. 27).
- Gathercole, Chris and Peter Ross (1994). 'Dynamic training subset selection for supervised learning in Genetic Programming.' In: *Parallel Problem Solving from Nature (PPSN) III*. Ed. by Y. Davidor, HP. Schwefel, and R. Männer. Vol. 866. Springer, Berlin, Heidelberg, pp. 312–321 (cit. on p. 34).
- Georghiades, A S, P N Belhumeur, and D J Kriegman (2001). 'From few to many: illumination cone models for face recognition under variable lighting and pose.' In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 23.6, pp. 643–660 (cit. on p. 117).
- Goodfellow, Ian (2016). 'NIPS 2016 Tutorial: Generative Adversarial Networks.' In: (cit. on pp. 42 57).
- Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio (2014). 'Generative Adversarial Nets.' In: *Advances in Neural Information Processing Systems* 27, pp. 2672–2680 (cit. on pp. 20, 43, 53, 57).
- Goshtasby, A Ardeshtir (2012). 'Image Registration: Principles, Tools and Methods.' In: London: Springer London. Chap. Similarity, pp. 7–66 (cit. on pp. 132 182).
- Gregor, Karol, Ivo Danihelka, Alex Graves, Daan Wierstra, Danilo Rezende, and Daan Wierstra (2015). 'DRAW: A Recurrent Neural Network For Image Generation.' In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, pp. 1462–1471 (cit. on pp. 44, 46, 47).
- Hall, Mark, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten (2009). 'The WEKA Data Mining Software: An Update.' In: *SIGKDD Explor. Newsl.* 11.1, pp. 10–18 (cit. on p. 66).
- Han, Hui, Wen-Yuan Wang, and Bing-Huan Mao (2005). 'Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning.' In: *Advances in intelligent computing* 17.12, pp. 878–887 (cit. on p. 32).
- Hara, K and K Nakayama (2000). 'A training method with small computation for classification.' In: *EEE-INNS-ENNS International Joint Conference on Neural Networks* 00.C, pp. 543–548 (cit. on p. 28).
- Hart, Philip (1968). 'The condensed nearest neighbor rule (Corresp.)' In: *IEEE Transactions on Information Theory* 14.3, pp. 515–516 (cit. on pp. 31, 34).
- He, Haibo, Yang Bai, Eduardo A. Garcia, and Shutao Li (2008). 'ADASYN: Adaptive synthetic sampling approach for imbalanced learning.' In: *Proceedings of the International Joint Conference on Neural Networks, IJCNN 2008, part of the IEEE World Congress on Computational Intelligence, WCCI 2008, Hong Kong, China, June 1-6, 2008*. IEEE, pp. 1322–1328 (cit. on p. 32).

- Heijer, Eelco den (2012). 'Evolving Art using Measures for Symmetry, Compositional Balance and Liveliness.' In: *IJCCI*, pp. 52–61 (cit. on pp. 176, 177).
- Hinton, Geoffrey E. (2006). 'Reducing the Dimensionality of Data with Neural Networks.' In: *Science* 313.5786, pp. 504–507 (cit. on pp. 15, 21).
- Hinton, Geoffrey E. and Richard S. Zemel (1994). 'Autoencoders, Minimum Description Length and Helmholtz Free Energy.' In: *Advances in Neural Information Processing Systems* 6, pp. 3–10 (cit. on p. 18).
- Hochreiter, Sepp and Jürgen Schmidhuber (1997). 'Long Short-Term Memory.' In: *Neural Computation* 9.8, pp. 1735–1780 (cit. on p. 16).
- Jaderberg, Max, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman (2014). 'Synthetic Data and Artificial Neural Networks for Natural Scene Text Recognition.' In: *CoRR* abs/1406.2227 (cit. on pp. 40, 42).
- Jankowski, Norbert and Marek Grochowski (2004). 'Comparison of Instances Seletion Algorithms I.' In: *Artificial intelligence and soft computing - ICAISC 2004. 7th international conference, Zakopane, Poland, June 7-11, 2004. Proceedings*. Pp. 598–603 (cit. on pp. 26, 31).
- Jesorsky, Oliver, Klaus J Kirchberg, and Robert W Frischholz (2001). 'Robust Face Detection Using the Hausdorff Distance.' In: *Computer. Lecture Notes in Computer Science* 2091. June, pp. 90–95 (cit. on p. 117).
- Kienzle, Wolf, Gökhan H. Bakır, Matthias O. Franz, and Bernhard Schölkopf (2005). 'Face Detection — Efficient and Rank Deficient.' In: *Advances in Neural Information Processing Systems* 17. Ed. by Lawrence K. Saul, Yair Weiss, and Léon Bottou. Cambridge, MA: MIT Press, pp. 673–680 (cit. on pp. 121, 127).
- Kingma, Diederik P and Max Welling (2013). 'Auto-Encoding Variational Bayes.' In: *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*. 2014 (cit. on pp. 20, 57).
- Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2012). 'ImageNet Classification with Deep Convolutional Neural Networks.' In: *Advances in Neural Information Processing Systems* 25. Ed. by F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger. Curran Associates, Inc., pp. 1097–1105 (cit. on p. 16).
- Larsen, Anders Boesen Lindbo, Søren Kaae Sønderby, and Ole Winther (2015). 'Autoencoding beyond pixels using a learned similarity metric.' In: *CoRR* abs/1512.09300 (cit. on pp. 41, 44).
- LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner (1998). 'Gradient-based learning applied to document recognition.' In: *Proceedings of the IEEE* 86.11, pp. 2278–2324 (cit. on p. 15).
- Lehman, Joel and Kenneth O Stanley (2008). 'Exploiting Open-Endedness to Solve Problems Through the Search for Novelty.' In: *Proc. of the Eleventh Intl. Conf. on Artificial Life (ALIFE XI)*. Cambridge, MA: MIT Press (cit. on pp. 183, 187).

- Lewis, David D. and Jason Catlett (1994). 'Heterogeneous uncertainty sampling for supervised learning.' In: *Proceedings of the 11th international conference on machine learning (ICML'94)*, pp. 148–156 (cit. on p. 28).
- Lewis, Matthew (2007). 'Evolutionary Visual Art and Design.' In: *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music*. Ed. by Juan Romero and Penousal Machado. Springer Berlin Heidelberg, pp. 3–37 (cit. on p. 61).
- Li, Congcong and Tsuhan Chen (2009). 'Aesthetic Visual Quality Assessment of Paintings.' In: *IEEE Journal of Selected Topics in Signal Processing* 3.2, pp. 236–252 (cit. on pp. 176, 177).
- Li, Yuangui, Zhonghui Hu, Yunze Cai, and Weidong Zhang (2005). 'Support Vector Based Prototype Selection Method for Nearest Neighbor Rules.' In: pp. 528–535 (cit. on p. 32).
- Lienhart, R, A Kuranov, and V Pisarevsky (2003). 'Empirical analysis of detection cascades of boosted classifiers for rapid object.' In: *Lecture Notes in Computer*, pp. 297–304 (cit. on p. 116).
- Lienhart, Rainer and Jochen Maydt (2002). 'An Extended Set of Haar-Like Features for Rapid Object Detection.' In: *IEEE ICIP 2002*, pp. 900–903 (cit. on pp. 69, 82).
- Lienhart, Rainer, Alexander Kuranov, and Vadim Pisarevsky (2002). 'Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection.' In: *Computing. Lecture Notes in Computer Science* 2781.MRL, pp. 297–304 (cit. on pp. 82, 104, 116, 118, 121).
- Lilliefors, Hubert W (1967). 'On the Kolmogorov-Smirnov test for normality with mean and variance unknown.' In: *Journal of the American Statistical Association* 62.318, pp. 399–402 (cit. on pp. 135, 151).
- Liu, Xu-Ying, Jianxin Wu, and Zhi-Hua Zhou (2009). 'Exploratory Undersampling for Class Imbalance Learning.' In: *IEEE Transactions on Systems, Man and Cybernetics* 39.2, pp. 539–550 (cit. on p. 24).
- MacDorman, Karl F, Robert D Green, Chin-Chang Ho, and Clinton T Koch (2009). 'Too real for comfort? Uncanny responses to computer generated faces.' In: *Computers in Human Behavior* 25.3, pp. 695–710 (cit. on p. 112).
- Machado, Penousal and Amílcar Cardoso (1997). 'Model Proposal for a Constructed Artist.' In: *First World Multiconference on Systemics, Cybernetics and Informatics, SCI'97/ISAS'97*. Ed. by N Callaos, C Khoong, and E Cohen. Vol. 2. Caracas, Venezuela, pp. 521–528 (cit. on p. 171).
- Machado, Penousal and Amílcar Cardoso (2002). 'All the truth about NEvAr.' In: *Applied Intelligence, Special Issue on Creative Systems* 16.2. Ed. by P Bentley and D Corne, pp. 101–119 (cit. on pp. 2, 50, 67).
- Machado, Penousal and João Correia (2014). 'Semantic Aware Methods for Evolutionary Art.' In: *Genetic and Evolutionary Computation Conference, GECCO '14, Vancouver, BC, Canada, July 12–16, 2014. Proceedings*. ACM (cit. on p. 219).

- Machado, Penousal, Juan Romero, and Bill Manaris (2007a). 'Experiments in Computational Aesthetics: An Iterative Approach to Stylistic Change in Evolutionary Art.' In: *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music*. Ed. by Juan Romero and Penousal Machado. Springer Berlin Heidelberg, pp. 381–415 (cit. on pp. 49, 50, 53, 57, 61, 67, 69, 70, 72, 131, 171, 172, 174, 176–178).
- Machado, Penousal, Juan Romero, Antonino Santos, Amílcar Cardoso, and Alejandro Pazos (2007b). 'On the development of evolutionary artificial artists.' In: *Computers {&} Graphics* 31.6, pp. 818–826 (cit. on pp. 49, 171, 172, 176, 177).
- Machado, Penousal, João Correia, and Juan Romero (2012a). 'Expression-Based Evolution of Faces.' In: *Evolutionary and Biologically Inspired Music, Sound, Art and Design - First International Conference, EvoMUSART 2012, Málaga, Spain, April 11–13, 2012. Proceedings*. Vol. 7247. Lecture Notes in Computer Science. Springer, pp. 187–198 (cit. on pp. 50, 53, 58, 61, 69, 85, 90, 91, 93, 117, 217).
- (2012b). 'Improving Face Detection.' In: *Genetic Programming - 15th European Conference, EuroGP 2012, Málaga, Spain, April 11–13, 2012. Proceedings*. Ed. by Alberto Moraglio, Sara Silva, Krzysztof Krawiec, Penousal Machado, and Carlos Cotta. Vol. 7244. Lecture Notes in Computer Science. Springer, pp. 73–84 (cit. on pp. 2, 50, 53, 58, 99, 174, 218).
- Machado, Penousal, Juan Romero, Marcos Nadal, Antonino Santos, João Correia, and Adrián Carballal (2015a). 'Computerized measures of visual complexity.' In: *Acta Psychologica* 160, pp. 43–57 (cit. on p. 218).
- Machado, Penousal, Adriano Vinhas, João Correia, and Anikó Ekárt (2015b). 'Evolving Ambiguous Images.' In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25–31, 2015*. Ed. by Qiang Yang and Michael Wooldridge. AAAI Press, pp. 2473–2479 (cit. on pp. 61, 90, 171, 217).
- Machado, Penousal, Adriano Vinhas, João Correia, and Anikó Ekárt (2015c). 'Evolving Ambiguous Images.' In: *AI Matters* 2.1, pp. 7–8 (cit. on pp. 90, 217).
- Machado, Penousal, João Correia, and Filipe Assunção (2015d). 'Graph-Based Evolutionary Art.' In: *Handbook of Genetic Programming Applications*. Ed. by Amir Gandomi, Amir Hossein Alavi, and Conor Ryan. Berlin: Springer (cit. on p. 176).
- Martins, Tiago, João Correia, Ernesto Costa, and Penousal Machado (2015). 'Evotype: Evolutionary Type Design.' In: *Evolutionary and Biologically Inspired Music, Sound, Art and Design - Fourth International Conference, EvoMUSART 2015, Copenhagen, Denmark, April 8–10, 2015, Proceedings*. Ed. by Colin G. Johnson, Adrián Carballal, and João Correia. Vol. 9027. Lecture Notes in Computer Science. Springer, pp. 136–147 (cit. on p. 219).
- (2016). 'Evotype: From Shapes to Glyphs.' In: *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. GECCO '16. New York, NY, USA: ACM, pp. 261–268 (cit. on p. 219).

- (2018). 'Evotype: Towards the Evolution of Type Stencils.' In: *Computational Intelligence in Music, Sound, Art and Design - Seventh International Conference, EvoMUSART 2018, Parma, Italy, April 4-6, 2018, Proceedings*. Ed. by Antonios Liapis, Juan Jesús Romero Cardalda, and Anikó Ekárt. Vol. 10783. Lecture Notes in Computer Science. Springer, pp. 299–314 (cit. on p. 219).
- McCormack, Jon (2007). 'Facing the Future: Evolutionary Possibilities for Human-Machine Creativity.' In: *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music*. Ed. by Juan Romero and Penousal Machado. Springer Berlin Heidelberg, pp. 417–451 (cit. on pp. 61, 171).
- Melville, Prem and Raymond J Mooney (2004). 'Creating diversity in ensembles using artificial data.' In: *Information Fusion* 6, pp. 99–111 (cit. on pp. 38, 39).
- Meyliana and Eko K. Budiardjo (2014). *An Application of Oversampling, Undersampling, Bagging and Boosting in Handling Imbalanced Datasets*. Ed. by Tutut Herawan, Mustafa Mat Deris, and Jemal Abawajy. Vol. 285. Lecture Notes in Electrical Engineering. Singapore: Springer Singapore, pp. 479–487 (cit. on p. 24).
- Miloud-Aouidate, Amal and Ahmed Riadh Baba-Ali (2013). 'An Efficient Ant Colony Instance Selection Algorithm for KNN Classification.' In: *International Journal of Applied Metaheuristic Computing* 4.3, pp. 47–64 (cit. on p. 34).
- Mitchell, Thomas M (1997). *Machine Learning*. McGraw-Hill Higher Education (cit. on pp. 1, 7, 9, 10).
- Mollineda, R.A., F.J. Ferri, and E. Vidal (2002). 'An efficient prototype merging strategy for the condensed 1-NN rule through class-conditional hierarchical clustering.' In: *Pattern Recognition* 35.12, pp. 2771–2782 (cit. on p. 29).
- Nakhmani, Arie and Allen Tannenbaum (2013). 'A new distance measure based on generalized image normalized cross-correlation for robust video tracking and image recognition.' In: *Pattern recognition letters* 34.3, pp. 315–321 (cit. on p. 182).
- Nguyen, Anh, Jason Yosinski, and Jeff Clune (2015). 'Deep Neural Networks are Easily Fooled.' In: *Computer Vision and Pattern Recognition, 2015 IEEE Conference on*, pp. 427–436 (cit. on pp. 50, 53).
- Niyogi, Partha, Federico Girosi, and Tomaso Poggio (1998). 'Incorporating prior information in machine learning by creating virtual examples.' In: *Proceedings of the IEEE* 86.11, pp. 2196–2209 (cit. on p. 37).
- Ojala, Timo, Matti Pietikäinen, and David Harwood (1996). 'A comparative study of texture measures with classification based on feature distributions.' In: *Pattern Recognition* 29.1, pp. 51–59 (cit. on p. 66).
- Olvera-López, J Arturo, J Ariel Carrasco-Ochoa, and J Fco. Martínez-Trinidad (2008). 'Prototype Selection Via Prototype Relevance.' In: *Progress in Pattern Recognition, Image Analysis and Applications: 13th Iberoamerican Congress on Pattern Recognition, CIARP 2008, Havana, Cuba, September 9-12, 2008. Proceedings*. Ed. by José Ruiz-Shulcloper and Walter G Kropatsch. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 153–160 (cit. on p. 30).

- Olvera-López, J. Arturo, J. Ariel Carrasco-Ochoa, J. Francisco Martínez-Trinidad, and Josef Kittler (2010). 'A review of instance selection methods.' In: *Artificial Intelligence Review* 34.2, pp. 133–143 (cit. on pp. [24](#), [26](#), [27](#), [32](#)).
- Olvera-López, José Arturo, Jesús Ariel Carrasco-Ochoa, and José Francisco Martínez Trinidad (2008). 'Object Selection Based on Clustering and Border Objects.' In: *Computer Recognition Systems 2*. Ed. by Marek Kurzynski, Edward Puchala, Michal Wozniak, and Andrzej Zolnerek. Vol. 45. Advances in Soft Computing. Springer, pp. 27–34 (cit. on p. [29](#)).
- Oord, Aaron van den, Nal Kalchbrenner, and Koray Kavukcuoglu (2016). 'Pixel Recurrent Neural Networks.' In: *International Conference on Machine Learning (ICML)* 48 (cit. on pp. [47](#), [49](#)).
- Osuna, E, R Freund, F Girosi, E. Osuna, R. Freund, and F. Girosi (1997). 'Training Support Vector Machines: An Application to Face Detection.' In: *Proceedings of Computer Vision and Pattern Recognition '97*, pp. 130–136 (cit. on p. [37](#)).
- Otsu, Nobuyuki (1979). 'A threshold selection method from gray-level histograms.' In: *IEEE transactions on systems, man, and cybernetics* 9.1, pp. 62–66 (cit. on p. [92](#)).
- Ougiaroglou, Stefanos and Georgios Evangelidis (2012). 'Efficient dataset size reduction by finding homogeneous clusters.' In: *Proceedings of the Fifth Balkan Conference in Informatics* October, pp. 168–173 (cit. on p. [30](#)).
- (2016). 'Efficient editing and data abstraction by finding homogeneous clusters.' In: *Annals of Mathematics and Artificial Intelligence* 76.3-4, pp. 327–349 (cit. on p. [30](#)).
- Paredes, Roberto and Enrique Vidal (2000). 'Weighting prototypes - a new editing approach.' In: *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*. Vol. 2. IEEE Comput. Soc, pp. 25–28 (cit. on p. [30](#)).
- Pérez, Patrick, Michel Gangnet, and Andrew Blake (2003). 'Poisson image editing.' In: *ACM Transactions on Graphics (TOG)*. Vol. 22. ACM, pp. 313–318 (cit. on p. [68](#)).
- Provost, Foster and Ron Kohavi (1998). 'Introduction: On Applied Research in Machine Learning.' In: *Machine Learning* 30.2-3, pp. 127–132 (cit. on p. [7](#)).
- Radford, Alec, Luke Metz, and Soumith Chintala (2015). 'Unsupervised representation learning with deep convolutional generative adversarial networks.' In: *arXiv preprint arXiv:1511.06434* (cit. on pp. [44](#), [45](#), [53](#)).
- Raicharoen, Thanapant and Chidchanok Lursinsap (2005). 'A divide-and-conquer approach to the pairwise opposite class-nearest neighbor (POC-NN) algorithm.' In: *Pattern Recognition Letters* 26.10, pp. 1554–1567 (cit. on p. [29](#)).
- Riquelme, José C., Jesús S. Aguilar-Ruiz, and Miguel Toro (2003). 'Finding representative patterns with ordered projections.' In: *Pattern Recognition* 36.4, pp. 1009–1018 (cit. on p. [28](#)).
- Romero, Juan, Penousal Machado, Antonino Santos, and Amílcar Cardoso (2003). 'On the Development of Critics in Evolutionary Computation Artists.' In: *Applications of Evolutionary Computing, EvoWorkshops 2003: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP*,

- EvoMUSART, EvoSTOC*. Ed. by R. Günther et al. Vol. 2611. LNCS. Essex, UK: Springer (cit. on pp. 53, 61, 69, 112, 171, 172, 176, 177, 215).
- Romero, Juan, Penousal Machado, Adrian Carballal, and João Correia (2012). 'Computing Aesthetics with Image Judgement Systems.' In: *Computers and Creativity*. Ed. by Jon McCormack and Mark D'Inverno. Springer Berlin Heidelberg, pp. 295–322 (cit. on pp. 171, 172).
- Rowley, Henry A, Shumeet Baluja, and Takeo Kanade (1998a). 'Neural Network-Based Face Detection.' In: *IEEE Transactions On Pattern Analysis and Machine intelligence* 20, pp. 23–38 (cit. on pp. 37, 40).
- Rowley, Henry, Shumeet Baluja, and Takeo Kanade (1998b). 'Rotation Invariant Neural Network-Based Face Detection.' In: *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* (cit. on p. 120).
- Rubner, Yossi, Carlo Tomasi, and Leonidas J Guibas (2000). 'The earth mover's distance as a metric for image retrieval.' In: *International journal of computer vision* 40.2, pp. 99–121 (cit. on p. 177).
- Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1986). 'Learning representations by back-propagating errors.' In: *Nature* 323, p. 533 (cit. on p. 10).
- Salzberg, S, A L Delcher, D Heath, and S Kasif (1995). 'Best-case results for nearest-neighbor learning.' In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17.6, pp. 599–608 (cit. on p. 30).
- Sammur, Claude and Geoffrey I Webb, eds. (2010). *Encyclopedia of Machine Learning*. Springer (cit. on p. 36).
- Sánchez, J. S. (2004). 'High training set size reduction by space partitioning and prototype abstraction.' In: *Pattern Recognition* 37.7, pp. 1561–1564 (cit. on p. 29).
- Santana, Modesto Castrillón, Oscar Déniz-Suárez, Luis Antón-Canal, and Javier Lorenzo-Navarro (2008). 'Face and Facial Feature Detection Evaluation - Performance Evaluation of Public Domain Haar Detectors for Face and Facial Feature Detection.' In: *VISAPP (2)*. Ed. by Alpesh Ranchordas and Helder Araújo. INSTICC - Institute for Systems, Technologies of Information, Control, and Communication, pp. 167–172 (cit. on p. 82).
- Sapp, Benjamin, Ashutosh Saxena, and Andrew Y Ng (2008). 'A Fast Data Collection and Augmentation Procedure for Object Recognition.' In: *AAAI '08: Proceedings of the 23rd national conference on Artificial Intelligence*, pp. 1402–1408 (cit. on p. 40).
- Saunders, Rob and John Gero (2001). 'The digital clockwork muse: A computational model of aesthetic evolution.' In: *AISB'01 Symposium on Artificial Intelligence and Creativity in Arts and Science*. Ed. by G Wiggins. York, UK, pp. 12–21 (cit. on p. 69).
- Secretan, Jimmy, Nicholas Beato, David B D Ambrosio, Adelein Rodriguez, Adam Campbell, and Kenneth O Stanley (2008). 'Picbreeder: evolving pictures collaboratively online.' In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, pp. 1759–1768 (cit. on p. 50).

- Settles, Burr (2010). 'Active Learning Literature Survey.' In: *Machine Learning* 15.2, pp. 201–221 (cit. on pp. [24](#), [25](#)).
- Simard, Patrice, Dave Steinkraus, and John Platt (2003). 'Best practices for convolutional neural networks applied to visual document analysis.' In: *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings*. Vol. 1. Icdar. IEEE Comput. Soc, pp. 958–963 (cit. on pp. [38](#), [39](#)).
- Sims, Karl (1991). 'Artificial Evolution for Computer Graphics.' In: *ACM Computer Graphics* 25, pp. 319–328 (cit. on p. [67](#)).
- Spector, Lee and Adam Alpern (1994). 'Criticism, culture, and the automatic generation of artworks.' In: *Proceedings of Twelfth National Conference on Artificial Intelligence*. Seattle, Washington, USA: AAAI Press/MIT Press, pp. 3–8 (cit. on pp. [49](#), [57](#), [72](#), [131](#)).
- Srisawat, Anantaporn, Tanasanee Phienthrakul, and Boonserm Kijsirikul (2006). 'SV-kNNC: An Algorithm for Improving the Efficiency of K-nearest Neighbor.' In: *Proceedings of the 9th Pacific Rim International Conference on Artificial Intelligence*, pp. 975–979 (cit. on p. [32](#)).
- Stahl, Volker, Alexander Fischer, Rolf Bippus, and Er Fischer (2001). 'Acoustic Synthesis of Training Data for Speech Recognition in Living Room Environments.' In: *in ICASSP*, pp. 21–24 (cit. on p. [38](#)).
- Sung, K.-K. and Tomaso Poggio (1995). 'Example-based learning for view-based human face detection.' In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20.1, pp. 39–51 (cit. on pp. [1](#), [23](#), [36](#), [37](#), [40](#), [51](#), [63](#), [146](#), [148](#)).
- Sung, Kah-kay (1996). 'Learning and Example Selection for Object and Pattern Detection.' PhD thesis (cit. on p. [33](#)).
- Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich (2015). 'Going deeper with convolutions.' In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9 (cit. on p. [16](#)).
- Tamura, Hideyuki, Shunji Mori, and Takashi Yamawaki (1978). 'Textural Features Corresponding to Visual Perception.' In: *Systems, Man and Cybernetics, IEEE Transactions on* 8.6, pp. 460–473 (cit. on p. [177](#)).
- Tanner, Martin; and Wing Hung Wong (2009). 'The Calculation of Posterior Distributions by Data Augmentation Author (s): Martin A . Tanner and Wing Hung Wong Source : Journal of the American Statistical Association , Vol . 82 , No . 398 (Jun . , 1987) , pp . 528- Published by : American Statistica.' In: *Journal of the American Statistical Association* 82.398, pp. 528–540 (cit. on p. [36](#)).
- Teller, Amit and M Veloso (1995). 'Algorithm evolution for face recognition: what makes a picture difficult.' In: *Evolutionary Computation, 1995., IEEE International Conference on* (cit. on pp. [49](#), [57](#), [72](#), [131](#)).

- Tenenbaum, Joshua B, Vin de Silva, and John C Langford (2000). 'A Global Geometric Framework for Nonlinear Dimensionality Reduction.' In: *Science* 290.5500, pp. 2319–2323 (cit. on p. 39).
- Tomek, Ivan (1976). 'An Experiment with the Edited Nearest-Neighbor Rule.' In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-6.6, pp. 448–452 (cit. on p. 31).
- Toussaint, Godfried (2002). 'Proximity graphs for nearest neighbor decision rules: recent progress.' In: *Proceedings of the 34th Symposium on Computing Science and Statistics* (cit. on pp. 31, 33).
- Triguero, I, M Galar, S Vluymans, C Cornelis, H Bustince, F Herrera, and Y Saeys (2015). 'Evolutionary undersampling for imbalanced big data classification.' In: *2015 IEEE Congress on Evolutionary Computation (CEC)*, pp. 715–722 (cit. on p. 35).
- Tsai, Chih-Fong, William Eberle, and Chi-Yuan Chu (2013). 'Genetic algorithms in feature and instance selection.' In: *Knowledge-Based Systems* 39, pp. 240–247 (cit. on p. 34).
- Van Der Malsburg, C (1986). 'Frank Rosenblatt: Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.' In: *Brain Theory*. Ed. by Günther Palm and Ad Aertsen. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 245–248 (cit. on p. 9).
- Vázquez, Fernando, J. Salvador Sánchez, and Filiberto Pla (2005). 'A Stochastic Approach to Wilson's Editing Algorithm.' In: ed. by Jorge S. Marques, Nicolás Pérez de la Blanca, and Pedro Pina. Vol. 3523. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 35–42 (cit. on p. 31).
- Vinhas, Adriano (2015). 'Novelty and Figurative Expression-Based Evolutionary Art.' MA thesis. Department of Informatic Engineering, Faculty of Sciences and Technology, University of Coimbra (cit. on pp. 67, 179, 180).
- Vinhas, Adriano, Filipe Assunção, João Correia, Aniko Ekárt, and Penousal Machado (2016). 'Fitness and Novelty in Evolutionary Art.' In: *Evolutionary and Biologically Inspired Music, Sound, Art and Design - Fifth International Conference, EvoMUSART 2016, Porto, Portugal, March 30 – April 1, 2016, Proceedings*. Ed. by Colin Johnson, Vic Ciesielski, João Correia, and Penousal Machado. Cham: Springer International Publishing, pp. 225–240 (cit. on pp. 171, 218).
- Viola, Paul and Michael Jones (2001). 'Rapid Object Detection using a Boosted Cascade of Simple Features.' In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. Hawaii, I–511–I–518 vol.1 (cit. on pp. 2, 15, 37, 62, 64–66, 72, 104, 116, 121).
- Vishwanathan, S. V. N. and M Narasimha Murty (2002). 'Use of Multi-category Proximal SVM for Data Set Reduction.' In: *Hybrid Information Systems*. Heidelberg: Physica-Verlag HD, pp. 19–24 (cit. on p. 29).
- Wang, Liwei, Yan Zhang, and Jufu Feng (2005a). 'On the Euclidean distance of images.' In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 27.8, pp. 1334–1339 (cit. on p. 182).

- Wang, Ruiping, Jie Chen, Shengye Yan, and Wen Gao (2005b). 'Face detection based on the manifold.' In: *Proceedings of the 5th international conference on Audio and Video-Based Biometric Person Authentication*. AVBPA'05. Hilton Rye Town, NY: Springer-Verlag, pp. 208–218 (cit. on pp. [1](#), [39](#)–[41](#)).
- Wilcoxon, Frank (1945). 'Individual comparisons by ranking methods.' In: *Biometrics bulletin*, pp. 80–83 (cit. on pp. [135](#), [151](#)).
- Wilson, D. Randall and Tony R. Martinez (2000). 'Reduction Techniques for Instance-Based Learning Algorithms.' In: *Machine Learning* 38.3, pp. 257–286 (cit. on p. [31](#)).
- Wilson, Dennis L. (1972). 'Asymptotic Properties of Nearest Neighbor Rules Using Edited Data.' In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-2.3, pp. 408–421 (cit. on p. [31](#)).
- World, Linda (1996). 'Aesthetic Selection: The Evolutionary Art of Steven Rooke.' In: *IEEE Computer Graphics and Applications* 16.1 (cit. on p. [61](#)).
- Y. Ng, Andrew and Michael I. Jordan (2002). 'On Discriminative vs. Generative Classifiers: A comparison of logistic regression and naive Bayes.' In: *Advances in Neural Information Processing Systems* 14. Ed. by T. G. Dietterich, S. Becker, and Z. Ghahramani, pp. 841–848 (cit. on pp. [8](#), [56](#)).
- Yang, Ming-Hsuan, Dan Roth, and Narendra Ahuja (2000). 'A SNoW-Based Face Detector.' In: *Advances in Neural Information Processing Systems* 12. MIT Press, pp. 855–861 (cit. on pp. [37](#), [51](#)).
- (2002). 'A Tale of Two Classifiers: SNoW vs. SVM in Visual Recognition.' In: *Proceedings of the 7th European Conference on Computer Vision-Part IV*. ECCV '02. London, UK, UK: Springer-Verlag, pp. 685–699 (cit. on p. [37](#)).