1 2 9 0

UNIVERSIDADE Đ
COIMBRA

Sofia da Silva Pereira

EVOLUTION OF DATA AUGMENTATION STRATEGIES APPLIED TO MEDICAL IMAGING

Dissertation in the context of the Master in Biomedical Engineering, Specialization in Clinical Informatics and Bioinformatics, advised by Professor João Nuno Gonçalves Costa Cavaleiro Correia and Professor Fernando Jorge Penousal Martins Machado and presented to the University of Coimbra.

October, 2021

1 2 9 0 FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE D COIMBRA

Evolution of Data Augmentation Strategies Applied to Medical Imaging

Sofia da Silva Pereira

Dissertation in the context of the Master in Biomedical Engineering, Specialization in Clinical Informatics and Bioinformatics, advised by Professor João Nuno Gonçalves Costa Cavaleiro Correia and Professor Fernando Jorge Penousal Martins Machado and presented to the University of Coimbra.

October, 2021

Acknowledgments

Em primeiro lugar, quero expressar o meu agradecimento aos orientadores da dissertação, Professor João Correia e Professor Penousal Machado pelo desafio que me lançaram e pela oportunidade de integrar o projeto que me inspirou. Obrigada por me ajudarem a encontrar o caminho a seguir e por todas as sugestões que permitiram o desenvolvimento deste trabalho.

À minha família, ao Rodrigo e à Gé, obrigada por todo o apoio, paciência e confiança que depositaram em mim ao longo do meu percurso académico. Com a vossa ajuda, nunca me faltou a motivação e entusiasmo por chegar ao fim deste ciclo.

Por último, um especial agradecimento aos professores, colegas e a todos aqueles que, de uma forma ou de outra, contribuíram para o sucesso do meu percurso académico e para as memórias que levo de Coimbra.

Financing

This research was partially funded by the project grant DSAIPA/DS/0022/2018 (GADgET) and by national funds through the FCT - Foundation for Science and Technology, I.P., within the scope of the project CISUC - UID/CEC/00326/2020 and by European Social Fund, through the Regional Operational Program Centro 2020.

Abstract

Deep Learning Algorithms are widely implemented and have reached state-of-the-art results in several scientific investigations. In medical images domain and Computer-Assisted Diagnosis (CAD) systems, Convolutional Neural Network (CNN) are the preferred deep network architecture. Despite getting good results, there are still some obstacles to overcome, namely the problem of overfitting. Lately, the Data Augmentation (DA) has been integrating the training pipeline of Deep Neural Networks (DNNs) to mitigate those issues. DA is a technique that contributes to increasing the diversity and size of training data sets. This is achieved by applying some transformations on the available training samples to generate new synthetic samples.

Classical image transformations, such as geometric and colour transformations, are the most popular techniques among data augmentation. The main drawback of this technique lies in the method used to define the most adequate DA strategy for each task. Strategies are usually defined by a combination of Transformation Functions (TFs) with their respective parameters. Currently, the search for the most suitable augmentation strategy is mainly done through trial-and-error processes and depends on the experience and time of the researchers.

Taking this into account, a novel data augmentation approach is proposed. This approach is able to automatically define an optimised DA strategy for each medical image classification task. The approach combines two algorithms, a deep learning algorithm and an evolutionary algorithm. A series of experiments and preliminary tests were carried out to evaluate algorithms configurations and architectures. The results obtained demonstrate that the same level of performance is achieved whether TFs without predefined parameters or TFs with fixed parameters are provided. Therefore, an automatic DA approach that does not need the previous definition of the different transformation functions and parameters is introduced.

Keywords

Data Augmentation, Medical Images, Evolutionary Algorithm, Deep Learning, Convolutional Neural Network

Resumo

Algoritmos de aprendizagem profunda são vastamente utilizados e têm alcançado os melhores resultados em diversas investigações científicas. No domínio das imagens médicas e em sistemas de diagnóstico auxiliado por computador, as redes neurais convolucionais são a arquitetura de rede profunda preferida. Apesar de se obter bons resultados, ainda existem alguns obstáculos a ultrapassar, nomeadamente o problema de sobreajuste. Para mitigar estes problemas, o aumento de dados (AD) tem vindo a integrar o fluxo de treino de redes neuronais profundas. O AD é uma técnica que permite aumentar a diversidade e o tamanho dos conjuntos de dados. Tal é conseguido através da geração de amostras sintéticas a partir das amostras de treino disponíveis.

As transformações clássicas de imagens, como as transformações geométricas e de cores, são as técnicas mais populares de AD. A principal desvantagem desta técnica reside na forma como a estratégia de AD mais adequada para cada tarefa é delineada. Normalmente, as estratégias são combinações de funções de transformação com os seus respetivos parâmetros. Atualmente, a procura pela estratégia de AD mais adequada é conseguida através de métodos de tentativa e erro e depende da experiência e do tempo dos investigadores.

Tendo isto em consideração, é proposta uma nova abordagem de AD capaz de definir automaticamente estratégias de AD otimizadas para cada tarefa de classificação de imagens médicas. A abordagem combina dois algoritmos, um de aprendizagem profunda e um evolucionário. Um conjunto de experiências e testes preliminares foram realizados para avaliar configurações e arquiteturas dos algoritmos. Os resultados obtidos demonstram que o mesmo nível de desempenho é atingido quer sejam aplicadas funções de transformação sem parâmetros pré-definidos como funções de transformação com parâmetros fixos. Portanto, uma abordagem de AD automática que não necessita da definição prévia das diferentes funções de transformação e parâmetros a aplicar é introduzida.

Palavras-Chave

Aumento de Dados, Imagens Médicas, Algoritmo Evolucionário, Aprendizagem Profunda, Rede Neural Convolucional

Contents

Abbreviations

ANN Artificial Neural Network.

CAD Computer-Assisted Diagnosis.

CNN Convolutional Neural Network.

CT Computed Tomography.

DA Data Augmentation.

DL Deep Leaning.

DNN Deep Neural Network.

EA Evolutionary Algorithm.

ES Evolutionary Strategy.

GAN Generative Adversarial Network.

ML Machine Learning.

MRI Magnetic Resonance Imaging.

NN Neural Network.

TF Transformation Function.

List of Figures

List of Tables

1

Introduction

This introductory chapter is divided into three sections where the motivation and context, the goals and the document structure will be presented. The first section explains the underlying motivation and the context of this dissertation. The second section defines the goals related to the development of the project. Finally, the last section presents the structure of the document.

1.1 Context and Motivation

Clinical decision support is an extensive field that enclose several systems that aim to help and improve the diagnostic of physicians. These systems can reduce the impact of distractions, fatigue, limitations of memory, preconceptions that the physicians can be subject to [1]. Besides that, the power of calculation and data analysis of computer systems can allow to detect important details that humans may miss. When referring to medical images, these systems are commonly called computer-aided diagnosis or computer-assisted detection (CAD) systems [1].

In the 1960s appeared the first computer analysis system applied to medical images. The task was to automatically distinguish between normal and abnormal chest photofluorograms. In 1998 a system that identifies the regions of interest on mammograms became the first of this type of systems to be approved for commercial purposes [1]. With the continuous increase of computational power, image analyses algorithms and machine learning techniques, Computer-Assisted Diagnosis (CAD) systems are now commonly used to complement the opinion of physicians, namely radiologists, dermatologists and pathologists [2].

Nowadays, several medical tests used to evaluate the health status of the human body and diagnose pathologies are delivered in image format. Some examples of these images are X-ray images, Computed Tomography (CT) scans, Magnetic Resonance Imaging (MRI) slices, histopathology slides, among others [3]. Usually, the identification of pathologies is based on the educated opinion of specialists [4]. The role of clinical decision support systems has been increasingly discussed, specifically CAD systems.These systems are being developed to assist in fields such as diagnosis, prognostic prediction and disease detection [4].

Deep learning techniques are widespread and have been key in achieving great results in most CAD systems. Convolutional Neural Network (CNN) is the most common neural network architecture used to perform classification tasks on medical imaging data sets [5]. CNNs have been widely applied in the context of medical imaging, as they have the advantage of extracting complex relationships from data sets and, at the same time, they are able to reduce the number of trainable parameters compared to other architectures [2]. Despite getting good results using CNNs, there are still some obstacles along the way. The main problems are that they need to be fed with a large amount of data to obtain satisfactory results, the training data is of poor quality or the data set has a small size leading to overfitting, the classes are uneven and the susceptibility to suffer adversarial attacks [6]. Data Augmentation (DA) has been helping to mitigate these issues.

DA is a technique that is increasingly being used by the scientific community in image classification problems. It has become a common practice to integrate data augmentation into the training pipeline of Deep Neural Networks (DNNs) in various research projects [7]. DA allows to increase the diversity and the size of training data sets. In the image domain, this is done by applying transformations over the available images in the training data set [5]. There are several data augmentation strategies from basic image transformations until neural networks based methods. Classical image transformations, such us translations, flips, rotations, are the most popular techniques among data augmentation. These strategies are very straightforward and simple to implement, in addition to improving the performance of classification models by decreasing overfitting [8].

On the other hand, there is still no delineated procedure implemented capable of finding the DA strategy that best fits each problem. At the moment, the majority of the processes used to select the most suitable data augmentation strategy depend on the experience and time of the researchers and is done mainly by trial-and-error tasks

[5]. With this in mind, a novel approach combining algorithms of Machine Learning (ML), Evolutionary Algorithms (EAs) and classical Transformation Functions (TFs) will be introduced. This approach is delivered in the form of a prototype and optimises the strategies with respect to the current task performance. The application of the prototype to medical image classification tasks aims to yield the most suitable DA strategy found automatically.

1.2 Goals

The main goal of this work is to develop a prototype composed by an evolutionary algorithm to evolve data augmentation strategies. The prototype is designed in a way that automates the search for the most suitable data augmentation technique. The DA strategies resulting from the evolutionary process are expected to increase the performance of existing deep learning algorithms applied to medical images. This goal can be divided into 6 sub-goals:

1. Gather a set of the most commonly used transformation functions applied in the field of medical imaging and evaluate them individually.
2. Design a prototype based on an evolutionary and deep learning algorithms.
3. Perform a series of preliminary tests to adjust the prototype settings.
4. Conduct experiments with different inputs and prototype settings.
5. Conduct a comparative study between different approaches and baseline.
6. Define the most advantageous input type and configuration of the prototype.

1.3 Document Structure

This dissertation is divide in 6 chapters. The first one has already been presented and explains the context, motivation and goals behind this project. Chapter 2 provides information regarding the applications and the existent types of data augmentation in medical imaging, as well as an overview of the relation between deep learning and data augmentation. Chapter 3 is a short theoretical introduction of two of the main aspects of the prototype, data augmentation and evolutionary algorithms. A brief explanation of the prototype operation and the software and hardware used in the elaboration of the project are also stated. Chapter 4 covers the development of the prototype along with decisions made from the preliminary tests results. Chapter 5 includes the exposition of the experimental setup details regarding the experiments carried out. The results obtained and their discussion are included as well. Finally, Chapter 6 provides the conclusions and considerations regarding the developed work and suggestions of future work to improve and apply the prototype.

2

State of the Art

This chapter provides an overview of the state of deep learning algorithms and identifies key approaches to data augmentation. Section 2.1 provides a brief definition of machine learning algorithms, namely deep learning. Section 2.2 examines the main data augmentation approaches and describes particular data augmentation strategies proposed in different studies.

2.1 Machine Learning

ML is an extremely studied and growing field, with applications in the most varied domains. It is defined as the capability of a machine to learn and improve by being fed with data. The machines learn from data how to perform several tasks such identifying patterns and detecting objects. This allows the systems to make decisions with less and less human interference. Machine learning has the advantages of performing repetitive and time-consuming tasks consistently and assiduously. It is also being applied to increasingly complex problems, managing to surpass the human brain in numerous tasks. This is causing a growth in interest in the field, with an emphasis on analysis and interpretation tasks of medical images. Machine learning algorithms promise to be of great help in the design of decision support and computer-aided diagnostics systems [9].

Unsupervised, semi-supervised and supervised are the three major types of machine learning algorithms. Supervised learning is the most common of learning processes and the one within the scope of this work. The training data is composed of data samples combined to their corresponding label. The objective of the learning process is to discover a relationship between the data samples (input) and their correspondent label (output). This relation can then be used to classify new data samples with unknown label [2]. Inside of the supervised learning two learning tasks can be distinguished: classification and regression [10]. Classification will be the one employed in this work.

A classification task is made up of at least two different phases, training and testing. In the training phase, the classification model is learned by analysing the relationship between training data and training labels. The training data must be representative of the study population in order to obtain a model with good generalisation, i. e., ability to deliver satisfactory results when used on new input data. In the test phase, the obtained model is evaluated by the percentage of new data samples classified correctly. No sample from the training data set can be present in the testing data set. A third phase, called the validation phase, can also exist between the training phase and the testing phase. This phase is used to improve model performance by tuning and optimising model parameters [2].

2.1.1 Deep Learning

Deep Leaning (DL) is a subfield of machine learning that uses Artificial Neural Networks (ANNs). ANNs are capable of carrying out all the processing steps normally present in classical machine learning algorithms like feature extraction, selection and learning. That allows to process raw data directly. Typically, deep neural networks are made of several nonlinear units called artificial neurons. Their architecture is based on the human brain where the basic unit is the biological neuron. Artificial neurons have multiple inputs and produce an output by applying an activation function to the result of the weighted sum of the inputs plus a bias term [2].

Artificial neurons are organised into a set of processing layers. The first layer, also called input layer, is composed of artificial neurons that receive as inputs external data. The last layer, also known as output layer, is made of artificial neurons that receive as inputs outputs of other neurons and return the output of the actual task. The middle layers are called hidden layers. The inputs of the neurons belonging to the hidden layers are, normally, outputs of the immediately preceding layer and the outputs are, usually, the inputs of the immediately following layer [11]. A scheme of a Neural Network (NN) is shown in Figure 2.1

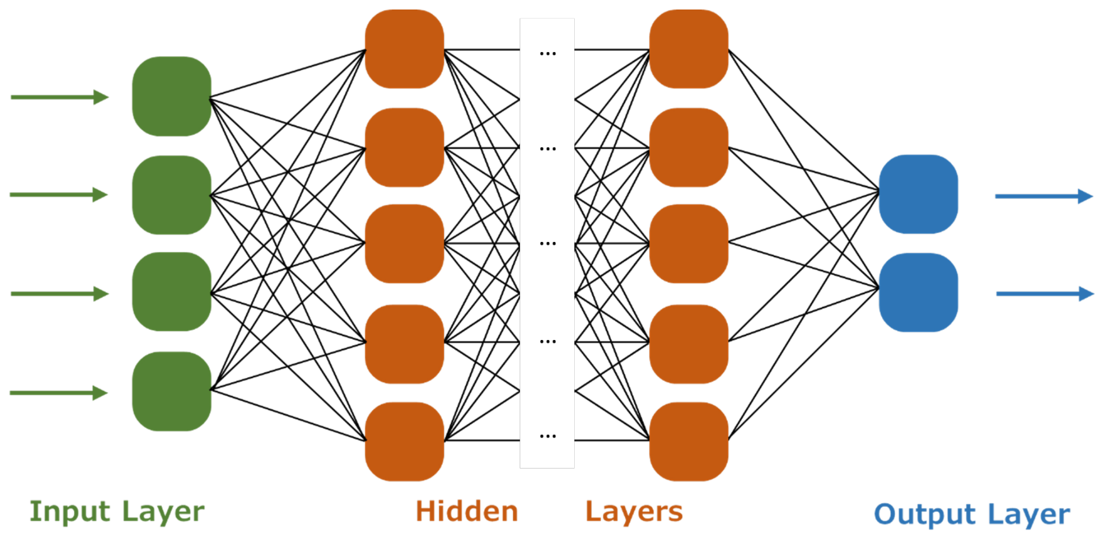


Figure 2.1: Typical scheme of a classical neural network. The coloured shapes represent neurons and the arrows illustrate the information flow.

A neural network can reach a huge number of trainable parameters based on the number of layers and neurons per layer. Consequently, dealing with relatively small data sets can be a risk due to the overfitting liability of the DL models [2].

Deep learning approaches are proving to be very effective for handling image-related tasks such as segmentation and classification and have been broadly used in the medical imaging domain. Two advantages of DL over other approaches are its representation ability and parallel calculation power that allows to model highly complex relationships [11].

The most common neural network architecture to handle image input data format is the CNN. CNNs are usually constituted by convolutional, pooling and fully connected layers, besides the input and output layers [2]. The convolution layers allow to minimise the number of trainable parameters, while retaining a sturdy representational ability. These layers apply a kernel and convolve it with the image to get network activations. The pooling layers downsample the input by computing the average or the maximum value over a window with a given size. The fully connected layers take each input and multiply it by a weights vector defined by the layer to produce an output vector [12].

Lately, the number of studies around medical imaging has shown an immense growth. The rapid development of deep learning and computer vision has allowed a great advance for CAD systems that use medical images such us pathological images, CT and MRI images, etc. The main tasks applied in the analysis of medical images are segmentation and classification [13]. Healthcare and biomedicine are considered to be one of the most valuable fields for artificial intelligence applications, being medical imaging the most promising domain [2].

2.2 Data Augmentation

Due to the development of CNNs, deep neural networks are being widely applied to computer vision to perform tasks such as image segmentation, object detection and image classification. This is a booming field thanks to the ease of access to big data, increased computational power, and continued research into deep network architectures. One of the most challenging improvements around deep learning models is the generalisation ability. Generalisation is defined as the faculty of a trained model being able to perform well when presented with unseen data. When a model has low generalisation ability, overfitting is very likely to occur [14].

It is known that, typically, larger data sets help to obtain deep learning models with better generalisation. However, gathering big data sets can be a very challenging task due to all the effort required to manually collect and label the data. Assembling big data sets of medical images is an even more difficult task due to the many constraints inherent in this sector. Some of these restrains are the necessity of medical professionals for labelling, the expenses and efforts necessary to carry out medical imaging processes, scarcity of samples of most diseases and patient privacy policies.

There are techniques developed to mitigate the overfitting problem. Some of these techniques are presented next. Dropout is a technique that prevents overfitting by ignoring a certain percentage of random neurons during the training phase. This forces the model to rely on more neurons and learn robuster features. Dropout can also be implemented by ignoring entire feature maps instead of single neurons. Another technique is batch normalisation that uses the first and the second statistical moments of the batch to normalise the set of activations in a layer. Transfer learning is a technique that uses the knowledge obtained by training a network on a large data set on a new classification task. Pretraining is a technique similar to transfer learning that allows some flexibility in the network architecture of the pretrained model. Finally, there is DA, that unlike the above techniques, approaches the problem by its root, the training data set. Augmentations can be applied to inflate the data set size but also to increase the diversity of samples by simulating more alterations that may occur in the real world [14].

In addition to being applied to mitigate the problem of overfitting, whether due to poor data quality or reduced data set size, data augmentation can also be employed to solve unbalanced class distribution issues [14] and the susceptibility of models to suffer from adversarial attacks. For example, Kwasigroch et al. (2017) [15] applies data augmentation to perform up-sampling due to a highly imbalanced data set. The strategy used was a combination of random classical transformations: rotation, translations, zoom and flips.

There are several data augmentation strategies that can be divided into 2 groups: white-box and black-box. The first group is associated with basic image transformations such as geometric and colour space transformations. Methods based on neural networks such as Generative Adversarial Network (GAN) and neural style transfer belong to the second group [6]. Basic image transformations require lower computational time and are far more simple to implement. Some of the most applied data augmentation methods are vertical and horizontal flips, vertical and horizontal translations, rotations, shearing, cropping, zooming, and scaling [16]. Simple transformations over input images, i. e., white-box strategies, have regularly demonstrated effectiveness in improving image classification performance [17, 18]. Generic data augmentation strategies have the advantage of being applied to different data sets maintaining their effectiveness [19].

Another common way to perform data augmentation is by using Generative Adversarial Networks (GANs). It is a technique that allows the unsupervised generation of new synthetic samples for training. GANs are formed by two networks, a generator and a discriminator. The role of the generator is to create new instances similar to the real ones in order to fool the discriminator. The discriminator must classify the new samples as synthetic or real, to minimise the difference. The two networks compete with each other becoming closer to their respective objective functions [20]. Having for example a data set of skin cancer, GANs have the ability to take an image of melanoma and create new instances by changing the size of the melanoma, placing it in a different position, superimposing it on healthy skin, etc [21]. Most of the time, GANs perform similar to traditional transformation methods, however, they require a lot more time and computer effort [17].

In some tasks expert knowledge is applied to suggest the most suitable data augmentations strategies [22]. Usually, a tuning of transformation functions parameters is performed to better capture the more significant features when training a deep neural network. However, in most cases DA procedures are arbitrary and the choices are justifiable taking into account the performance obtained instead of clinical considerations [23].

Zhang et al. (2018) [24] proposed a data augmentation strategy called mixup. This approach generates data by linearly interpolating images and corresponding labels from the available training data set. This produces unrealistic images, however, it has been shown to improve generalisation of state-of-the-art classification models. Chaitanya et al. (2021) [8] proposed an augmentation method based on GANs approaches that optimises the parameters of the generator by integrating unlabelled images and task loss. This process also results in augmented images somewhat unrealistic, despite improving the models performance. These examples show that reproducing realistic images is not the only way of achieving performance improvements.

3

Prototype Definition

This chapter is composed by 4 sections. The first section presents a brief explanation of data augmentation along with some difficulties in its implementation. A short theoretical introduction on evolutionary algorithms is stated in section 3.2. Section 3.3 provides an overview of the prototype concept and its operation. The last section refers the software and hardware used in the development of this dissertation.

3.1 Data Augmentation

As mentioned before, there are several data augmentation techniques. In this work, the focus will be on white-box data augmentation strategies, that is, those that do not use any type of neural networks. They are the most popular strategies because they are easily interpreted and have a lower computational cost compared to strategies belonging to the black-box category.

This type of data augmentation creates new data by applying basic transformations to exiting samples of the training data set. The most common practice is to apply a combination of transformations. Each individual transformation will be referred to as a transformation function (TF). TFs can be divided into several groups according to the type of alteration they apply to the images. A division into three groups was made:

- Arithmetic transformations;
- Geometric transformations;

- Other transformations.

Addition, multiplication, dropout, additions of Gaussian, Laplacian, salt and pepper noise, among others, are part of the arithmetic functions. Geometric functions include flips, zooms, translations, rotations and more. The last group includes transformations that do not fit into the previous categories, such as brightness, sharpness, blur, contrast, saturation, among others.

Each TF can have one or more parameters associated with it. The parameters are used to define an intensity, for example for the brightness function, a probability, for example for the dropout function and an action, for example for the translation function, to know what values to place in the pixels where there is no longer an image. Parameters can be single values, but they can also be defined as ranges.

It is easy to understand that, considering the number of existing transformation functions, the combinations they can generate between them, in addition to the number of possible parameter values, it can be said that the search space for data augmentation strategies is practically infinite. However, there is no guarantee that all possible strategies will be advantageous. To delineate the DA strategy, it is first necessary to select the functions that will be used, define the order in which they will be applied and define the parameters of each one. Currently, the search for the most adequate DA strategy for each problem is done manually. This search can be a very time-consuming and complex process that will hardly result in an optimal solution. In order to mitigate these less appealing aspects in the use of white-box DA strategies, an approach that consists of automating the search process for the most appropriate DA strategy for each problem is proposed. From the development of a prototype designed using an evolutionary approach and machine learning, it is intended to generate DA strategies that, when applied to the data set, lead to an increase in the performance of classification systems.

3.2 Evolutionary Algorithm

Evolutionary algorithms (EA) are optimisation methods inspired by the process of natural selection presented by Darwin in 1859. This type of search algorithm brings more advantages over the traditional ones, as it allows working with different types of variables simultaneously, it is more difficult to be stuck in local optimal solutions and it is suitable for problems with stochastic characteristics [25].

There are different types of evolutionary algorithms, but they all share the same underlying idea. There is a population made up of several individuals who must compete for resources resulting in natural selection, where the fittest ones survive and seed the next generation through recombination and/or mutation breeding offspring. The offspring and remain population are going to compete for a place in the next generation and the fittest will survive. This process is repeated until a computational limit is reached or an individual with enough quality is found.

The two main keys of evolutionary algorithms are the selection mechanism and the recombination and mutation mechanism, also called variation operators. The selection mechanism picks the most viable candidates considering the fitness function. This will increase the quality of the population. The function of the variation operators is to create and increase population diversity [26].

The components needed to define an evolutionary algorithm are:

- Initialisation;
- Representation;
- Evaluation function;
- Population;
- Selection;
- Variation operators;
- Termination condition.

Initialisation is the process used to create the first generation of individuals. Usually, the first generation consists of randomly generated individuals, but it is also possible to create a population with greater fitness using problem-specific heuristics and greater computational effort.

Representation is the definition of individuals through genotype and phenotype. The genotype, as the name implies, is the sequence of genes that characterises each individual in the population within the evolutionary algorithm. Each genotype matches a particular phenotype. The phenotype is the definition of individuals in the context of the original problem. Different genotype can give rise to the same phenotype. In Biology, the genotype is the set of genes of each living being and the phenotype is its physical appearance.

The evaluation function or fitness function is used to calculate the adaptability of candidates to their environment. Normally, the process used to evaluate candidates consists of the inverse representation of the individual, i.e., decoding the genotype to the phenotype, following by the application of the fitness function in the phenotype space. The individuals who demonstrate better adaptability are more likely to pass on to the next generation.

Population is the unit of evolution and is defined as the set of individuals in the genotype space. The population has a certain size that, typically, does not vary during evolution, allowing to reproduce the necessary competition for available resources. Population diversity, i.e., the number of different solutions, can vary when moving from the genotype space to the phenotype space. The presence of a phenotype implies the presence of at least one genotype, but the presence of a genotype implies the presence of at most one phenotype.

The selection mechanism is applied at the population level. Its role is to select certain individuals as parents to generate offspring for the next generation. This selection can be probabilistic, but giving advantage to individuals with better fitness, or it can be deterministic, simply selecting the fittest ones.

The variation operators, as already mentioned, are recombination/crossover and mutation. These operators have the function of creating new individuals, the children, from those that already exist, the parents. These are stochastic processes, as the result depends on various random choices. The mutation is applied to an individual to create another. The choice of the characteristic to change and its new value is done randomly. For recombination to occur, the participation of two parents is necessary to generate one or two offspring. The choice of which characteristics of each parent will be used and how they will be combined is done randomly.

The termination condition determines the end of evolution. Some termination conditions that could be used are: achieving a certain optimal fitness value, when it exists and is known; a computational limit is reached; the diversity of the population reaches a certain thresholds [26].

The evolutionary algorithm chosen to integrate the built prototype was an Evolutionary Strategy (ES), more specifically (1+λ)ES. In this case, there is only one parent who will compete with the offspring created by his mutation. Therefore, the only variation operator present is mutation. Mutation is performed by randomly changing a gene of the candidate by another randomly chosen value. Typically, in ES, the individuals are represented by real-valued vectors and the population size is usually smaller than in other evolutionary algorithms [27].

3.3 The prototype

As the main objective of this work is the automation of the search process for the most adequate DA strategy for the classification task, a prototype was built for this purpose. The prototype is based on two different mechanisms, an evolutionary algorithm and a machine learning algorithm. The evolutionary algorithm has the function of automatically generate data augmentation strategies. The evolutionary process is guided by an evaluation function. The role of the machine learning algorithm is to evaluate each of the proposed strategies. The evolutionary algorithm chosen was the evolutionary strategy and the recommended machine learning algorithm is deep learning, more specifically a CNN.

The functioning of the prototype is described by the following steps:

1. The evolutionary algorithm is fed by a set of transformation functions;
2. Through the evolutionary process, solutions formed by the supplied TF are generated;
3. Each solution is used to augment the training data set that will serve as input to the deep learning algorithm;
4. The DL algorithm will return the metrics (fitness function) obtained after training the classifier with the data set augmented by each of the solutions provided;
5. The evolutionary algorithm, based on the returned metrics, will follow up the evolution of the solutions;
6. New solutions are created based on the previous ones and their respective fitness function;
7. The process is repeated from step 3 to 7 until the number of generations reaches the defined value.

It should be noted that this entire procedure has a stochastic nature. In the calculation of the fitness function for each individual, is inherent all the randomness associated with the training of a deep learning classification model, as well as the randomness of the augmentation process. Consequently, the same individual can be evaluated multiple times and, each time, can be associated with different scores. This has implications for the evolutionary algorithm optimisation curve of the best solution, as may not be monotonically increasing as expected. As an example, the same individual can be selected two consecutive generations as the fittest. This does not mean that its fitness value will be the same in both generations, nor that there will be two adjacent points on the evolutionary optimisation curve with the same score. The second time it is selected, the fitness value can be lower or higher than the previous one. As a result, there may be a rise or fall in the evolutionary optimisation curve.

3.4 Software and Hardware

All the code used in the realisation of this dissertation was written in python programming language, version 3.7.9. Several open-source python libraries were used with highlighting for TensorFlow 2 [1] through Keras API [2], imblearn [3] and imgaug [4]. TensorFlow 2 is a platform used to implement machine learning algorithms. Keras is a tensorflow API that specialises in deep learning problems. Imblearn or imbalanced learn is an open-source python library that allows dealing with imbalanced classes for classification tasks. Imgaug is also an open-source python library used to perform data augmentation tasks for machine learning projects. It allows easy manipulation and combination of various data augmentation techniques.

The studies conducted were performed on three computers equipped with GeForce GTX 1080 Ti GPU, Intel Core i7-9700K processor and 32 GB RAM memory and one computer equipped with Titan X GPU, Intel Core i7-9700K processor and 32 GB RAM memory.

[1]https://www.tensorflow.org/
[2]https://keras.io/
[3]https://imbalanced-learn.org/stable/
[4]https://imgaug.readthedocs.io/en/latest/

4

Prototype Development

This chapter showcases all aspects regarding the development of the prototype. It is divided into 6 sections. The first describes the data set that will be used during the experiments. Section 4.2 surveys the selected functions and explains how they will be used. Section 4.3 presents the machine learning algorithm that will be used during the experiments. Section 4.4 describes the evolutionary strategy used and and its settings. Section 4.5 reports preliminary tests carried out in order to adjust additional prototype settings. Finally, section 4.6 resumes all the configurations applied to the prototype that will be used in the experiments referred to in the next chapter.

4.1 Data set

The data set chosen to test the prototype designed was Histopathology Images of the Breast consisting of 277 524 small patches of size 50x50 pixels that were extracted from digital images of breast tissue. Each sample is classified as positive (label 1) if it contains characteristics of invasive ductal carcinoma cells, or negative (label 0) otherwise. There are 78 786 samples with the presence of Invasive Ductal Carcinoma (IDC) and in the remaining 198 738 it is not. Invasive Ductal Carcinoma is the most common type of breast cancer, encompassing 80% of the cases. This carcinoma is detected by an examination called screening mammogram [28]. The data set is available on the Kaggle platform, which is a repository of community published code and data. Figure 4.1 displays examples of samples with the presence of IDC (red) and without the presence of IDC (green).

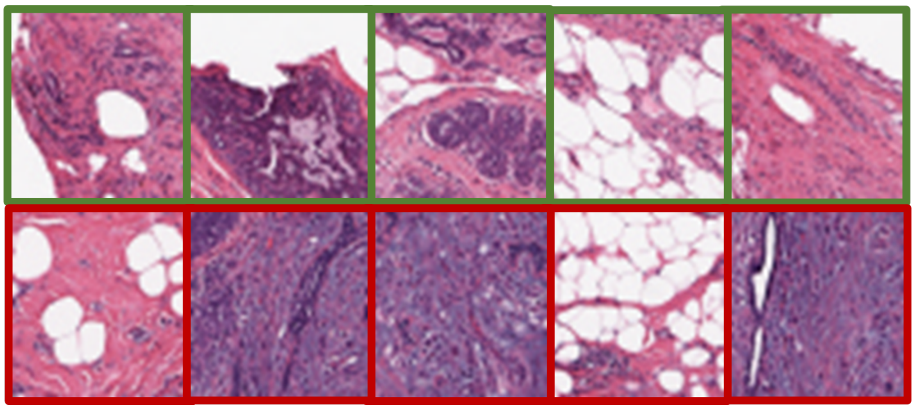


Figure 4.1: Examples of samples without the presence of IDC (green) and with IDC (red).

The data set processing was adapted from a script published on the same page of the data set on the Kaggle platform. This processing consisted of selecting the first 90 000 images from the original data set followed by a division into testing and training. For testing, 20% of the data (18 000 images) were randomly selected and for training the remaining 80% of the samples (72000 images). Then, with the support of the open-source library imblearn, the training data set was balanced. The resulting data set is described in Table 4.1:

	No. of train samples	**No. of test samples**	
No. of IDC(−)	20 371	12 954	33 325
No. of IDC(+)	20 371	5 046	25 417
	40 742	18 000	58 742

Table 4.1: Number of train, test, IDC(-) and IDC(+) samples.

4.2 Transformation Functions

A set of transformation functions was selected from the python library imgaug. The functions were chosen based on their mention in medical imaging literature. Other functions considered pertinent in the field of medical imaging were also picked. The functions were divided into three groups according to the type of associated transformation. The number of selected functions, total and per group can be consulted in Table 4.2. Generally, the functions belonging to the geometric group are the ones that result in the best performance [29].

Group	**No. of different functions (excluding parameters)**	**No. of functions (including parameters)**
Arithmetic	17	72
Geometric	9	31
Others	13	33
Total	**39**	**136**

Table 4.2: Number of functions per group with and without parameters variation.

The 39 different transformation functions selected are shown in Table 4.3. The variables p1, p2 and p3 are to be replaced by the parameters values. Since p1, p2 and p3 can vary between 0 and 1, some conditions were added around these variables, for each TF, to ensure that the recommended boundaries described in the imgaug documentation are met. To deal with descriptive parameters, the variables are used to define an index of a vector with all the possibilities. For example, the translate functions have a field called "mode" that stipulates how the newly created pixels should be filled. The options available for this field are "constant", "edge", "symmetric", "reflect" and "wrap". So, the vector with all the possibilities is: mode_op = ['constant','edge','symmetric','reflect','wrap']. It is possible to obtain an index from the p variables by multiplying their value by 5 (possibilities vector length) and selecting the whole part of the result. This way, all possibilities are equally likely to be chosen.

Imagining that p1 and p2 values are given. It is possible that the p1 value is greater than the p2 value. However, when dealing with intervals, if the variable p1 sets the left bound and p2 the right bound, this will cause a nuisance as it results in an invalid interval. So, when variables are used to define intervals, the left limit will take the value of the smallest variable value, either p1 or p2, and the right limit will be the greatest value between p1 and p2.

Nevertheless, it may happen that the given parameters do not produce feasible functions. For these scenarios, a validation method was created to catch the malfunctioning functions. When a not feasible function is detected the algorithm reruns the function creation method until a feasible function is generated.

Index (1st part (of the gene)	**Transformation Function**
0	"Add((-round(100*p1),round(100*p2)))"

Index (1st part (of the gene)	Transformation Function
1	"AddElementwise((-round(100*p1),round(100*p2)))"
2	"AdditiveGaussianNoise(scale=(0,round(p1*255)),per_channel=p2)"
3	"AdditiveLaplaceNoise(scale=(0,round(p1*255)),per_channel=p2)"
4	"AdditivePoissonNoise((0,p1*10),per_channel=p2)"
5	"Multiply((2*min([p1,p2]),2*max([p1,p2])))"
6	"MultiplyElementwise((2*min([p1,p2]),2*max([p1,p2])))"
7	"Cutout(nb_iterations=round(10*p1),size=p2, fill_mode=’gaussian’ if p3<0.5 else ’constant’)"
8	"Dropout(p=(0,0.5*p1),per_channel=p2)"
9	"CoarseDropout(p1,size_percent=p2)"
10	"ReplaceElementwise(0.5*p1,[0,255],per_channel=p2)"
11	"SaltAndPepper(0.5*p1,per_channel=p2)"
12	"CoarseSaltAndPepper(0.5*p1,size_percent=p2,per_channel=p3)"
13	"Salt(0.5*p1,per_channel=p2)"
14	"CoarseSalt(0.5*p1,size_percent=p2,per_channel=p3)"
15	"Pepper(0.5*p1,per_channel=p2)"
16	"CoarsePepper(0.5*p1,size_percent=p2)"
17	"Fliplr(p1)"
18	"Flipud(p1)"
19	"ScaleX((2*min([p1,p2]),2*max([p1,p2])),mode=mode_op[int(5*p3)])"
20	"ScaleY((2*min([p1,p2]),2*max([p1,p2])),mode=mode_op[int(5*p3)])"
21	"TranslateX(percent=(-p1,p2),mode=mode_op[int(5*p3)])"
22	"TranslateY(percent=(-p1,p2),mode=mode_op[int(5*p3)])"
23	"Rotate((-p1,p2),mode=mode_op[int(5*p3)])"
24	"ShearX((-p1,p2),mode=mode_op[int(5*p3)])"
25	"ShearY((-p1,p2),mode=mode_op[int(5*p3)])"
26	"MultiplyBrightness((2*min([p1,p2]),2*max([p1,p2])))"
27	"AddToBrightness((-round(100*p1),round(100*p2)))"
28	"MultiplySaturation((2*min([p1,p2]),2*max([p1,p2])))"
29	"AddToSaturation((-round(100*p1),round(100*p2)))"
30	"GammaContrast((2*min([p1,p2]),2*max([p1,p2])))"
31	"LinearContrast((2*min([p1,p2]),2*max([p1,p2])))"
32	"HistogramEqualization()"
33	"GaussianBlur(sigma=3*p1)"
34	"AverageBlur(k=int(7*p1))"
35	"pillike.EnhanceContrast(factor=(0.5+p1,1.5-p2))"
36	"pillike.EnhanceBrightness(factor=(0.5+p1,1.5-p2))"

Index (1st part (of the gene)	**Transformation Function**
37	"pillike.EnhanceSharpness(factor=(0.5+p1,1.5-p2))"
38	"pillike.FilterSharpen()"

Table 4.3: All 39 selected functions with their respective first part of the gene. The values p1, p2 and p3 will vary during the evolution process. Vector [p1 p2 p3] is the second part of the genes.

From the 39 different functions selected, 136 functions were established (Table 4.4), by defining parameters. These parameters have been set within small ranges in order to result in smooth changes to the image. For the same function, several parameters may have been defined. For example, the TF "Fliplr" in Table 4.3 was defined with six different parameters, resulting in six different TFs present in Table 4.4.

Index (Gene)	**Transformation Function**
0	"TranslateY(percent=(-0.2,0.2),mode='edge')"
1	"Flipud(0.3)"
2	"TranslateX(percent=(-0.1,0.1),mode='edge')"
3	"Fliplr(0.4)"
4	"Rotate((-40,40),mode='edge')"
5	"Flipud(0.4)"
6	"Fliplr(0.1)"
7	"Fliplr(0.3)"
8	"Rotate((-10,10),mode='edge')"
9	"ShearY((-20,20),mode='edge')"
10	"ShearX((-30,30),mode='edge')"
11	"Fliplr(0.5)"
12	"ScaleY((0.9,1.1),mode='edge')"
13	"Rotate((-30,30),mode='edge')"
14	"CoarseDropout(0.02,size_percent=0.7)"
15	"ShearY((-10,10),mode='edge')"
16	"ScaleX((0.8,1.2),mode='edge')"
17	"ShearX((-20,20),mode='edge')"
18	"Fliplr(0.2)"
19	"TranslateY(percent=(-0.1,0.1),mode='edge')"
20	"Rotate((-50,50),mode='edge')"
21	"Fliplr(0.6)"

Index (Gene)	**Transformation Function**
22	"Rotate((-20,20),mode='edge')"
23	"ShearY((-30,30),mode='edge')"
24	"ReplaceElementwise(0.05,[0,255])"
25	"Flipud(0.6)"
26	"ScaleX((0.9,1.1),mode='edge')"
27	"AverageBlur(k=2)"
28	"AdditivePoissonNoise(30)"
29	"Dropout(p=(0,0.05))"
30	"CoarseDropout(0.02,size_percent=0.9)"
31	"CoarseSaltAndPepper(0.05,size_percent=0.9)"
32	"Flipud(0.5)"
33	"TranslateX(percent=(-0.2,0.2),mode='edge')"
34	"CoarseDropout(0.03,size_percent=0.9)"
35	"Flipud(0.1)"
36	"MultiplyElementwise((0.8,1.2))"
37	"CoarseDropout(0.03,size_percent=0.5)"
38	"AdditiveGaussianNoise(scale=(0,0.3*255))"
39	"CoarseSaltAndPepper(0.05,size_percent=0.5)"
40	"SaltAndPepper(0.05)"
41	"CoarsePepper(0.05,size_percent=0.7)"
42	"ReplaceElementwise(0.1,[0,255])"
43	"ScaleY((0.8,1.2),mode='edge')"
44	"CoarseSaltAndPepper(0.1,size_percent=0.7)"
45	"Pepper(0.1)"
46	"Flipud(0.2)"
47	"Cutout(nb_iterations=(1,3),size=0.2,fill_mode='gaussian')"
48	"AdditiveGaussianNoise(scale=(0,0.1*255))"
49	"ShearX((-10,10),mode='edge')"
50	"AdditiveLaplaceNoise(scale=(0,0.1*255))"
51	"Salt(0.1)"
52	"MultiplyElementwise((0.7,1.3))"
53	"Cutout(nb_iterations=(1,3),size=0.05,fill_mode='gaussian')"
54	"CoarsePepper(0.1,size_percent=0.5)"
55	"CoarseDropout(0.03,size_percent=0.7)"
56	"AdditiveGaussianNoise(scale=(0,0.2*255))"
57	"Cutout(nb_iterations=(1,3),size=0.1,fill_mode='gaussian')"
58	"CoarsePepper(0.1,size_percent=0.7)"

Index (Gene)	**Transformation Function**
59	"AddElementwise((-20,20))"
60	"AdditiveLaplaceNoise(scale=(0,0.2*255))"
61	"ReplaceElementwise(0.15,[0,255])"
62	"AdditivePoissonNoise(20)"
63	"SaltAndPepper(0.1)"
64	"CoarseSalt(0.05,size_percent=0.9)"
65	"CoarsePepper(0.05,size_percent=0.5)"
66	"Dropout(p=(0,0.1))"
67	"CoarseSaltAndPepper(0.1,size_percent=0.5)"
68	"CoarseDropout(0.02,size_percent=0.5)"
69	"Pepper(0.05)"
70	"Salt(0.05)"
71	"ReplaceElementwise(0.05,iap.Normal(128,0.4*128))"
72	"ReplaceElementwise(0.1,iap.Normal(128,0.4*128))"
73	"Salt(0.15)"
74	"CoarseSalt(0.1,size_percent=0.9)"
75	"CoarseSalt(0.05,size_percent=0.5)"
76	"AddElementwise((-60,60))"
77	"CoarsePepper(0.05,size_percent=0.9)"
78	"CoarseSaltAndPepper(0.1,size_percent=0.9)"
79	"ReplaceElementwise(0.2,[0,255])"
80	"CoarseSalt(0.1,size_percent=0.7)"
81	"ReplaceElementwise(0.15,iap.Normal(128,0.4*128))"
82	"AddElementwise((-40,40))"
83	"SaltAndPepper(0.2)"
84	"GammaContrast((0.9,1.1))"
85	"pillike.EnhanceSharpness(factor=(0.9,1.1))"
86	"pillike.EnhanceSharpness(factor=(0.5,1.5))"
87	"CoarseSalt(0.1,size_percent=0.5)"
88	"AddElementwise((-10,10))"
89	"pillike.FilterSharpen()"
90	"Dropout(p=(0,0.2))"
91	"CoarseSaltAndPepper(0.05,size_percent=0.7)"
92	"AdditivePoissonNoise(10)"
93	"MultiplyElementwise((0.6,1.4))"
94	"CoarsePepper(0.1,size_percent=0.9)"
95	"Add((-10,10))"

Index (Gene)	**Transformation Function**
96	"ReplaceElementwise(0.2,iap.Normal(128,0.4*128))"
97	"GaussianBlur(sigma=0.5)"
98	"GaussianBlur(sigma=0.1)"
99	"Add((-20,20))"
100	"CoarseSalt(0.05,size_percent=0.7)"
101	"GammaContrast((0.5,1.5))"
102	"pillike.EnhanceContrast(factor=(0.9,1.1))"
103	"pillike.EnhanceSharpness(factor=(0.7,1.3))"
104	"AddToBrightness((-10,10))"
105	"AddToSaturation((-10,10))"
106	"Pepper(0.15)"
107	"AdditiveLaplaceNoise(scale=(0,0.3*255))"
108	"SaltAndPepper(0.15)"
109	"AddToSaturation((-20,20))"
110	"GaussianBlur(sigma=0.3)"
111	"GammaContrast((0.7,1.3))"
112	"MultiplyBrightness((0.9,1.1))"
113	"MultiplySaturation((0.9,1.1))"
114	"Add((-40,40))"
115	"LinearContrast((0.9,1.1))"
116	"pillike.EnhanceBrightness(factor=(0.9,1.1))"
117	"AddToBrightness((-30,30))"
118	"AddToSaturation((-30,30))"
119	"Add((-60,60))"
120	"MultiplySaturation((0.7,1.3))"
121	"Multiply((0.8,1.2))"
122	"pillike.EnhanceContrast(factor=(0.7,1.3))"
123	"Add((-80,80))"
124	"AddToBrightness((-50,50))"
125	"Multiply((0.7,1.3))"
126	"MultiplyBrightness((0.7,1.3))"
127	"LinearContrast((0.7,1.3))"
128	"HistogramEqualization()"
129	"pillike.EnhanceContrast(factor=(0.5,1.5))"
130	"pillike.EnhanceBrightness(factor=(0.7,1.3))"
131	"AddToBrightness((-70,70))"
132	"MultiplySaturation((0.5,1.5))"

Index (Gene)	Transformation Function
133	"LinearContrast((0.5,1.5))"
134	"Multiply((0.6,1.4))"
135	"MultiplyBrightness((0.5,1.5))"

Table 4.4: All 136 selected functions with their respective genes and predefined parameters.

Examples of the resulting images after the application of some TFs are depicted in Figure 4.2. Whenever it is mentioned that a TF is applied to the training data set as a form of DA, the probability of it being applied on each image is 50%. The only exception resides in flips transformations, where the parameter itself defines the probability of being applied to an image.

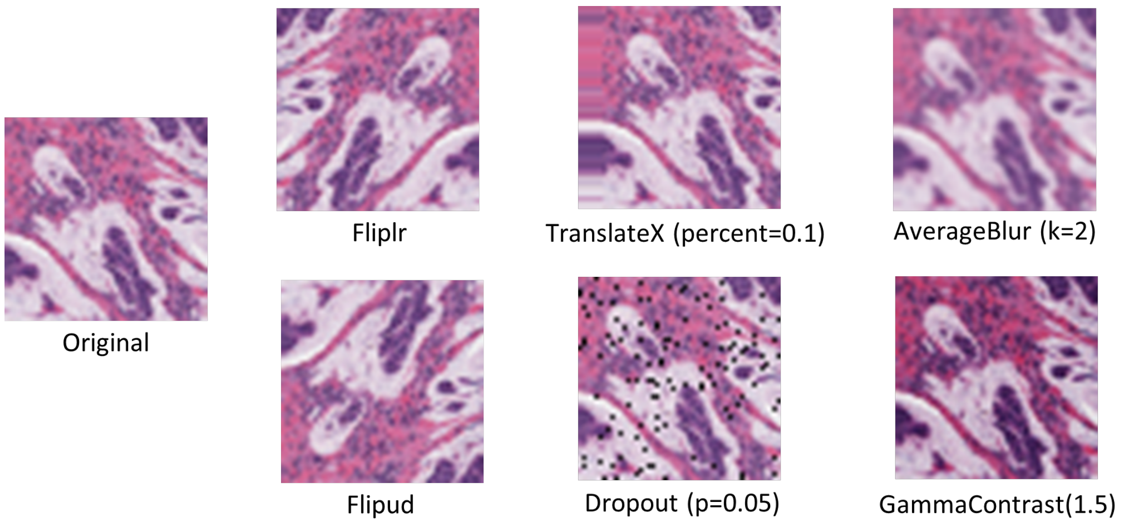


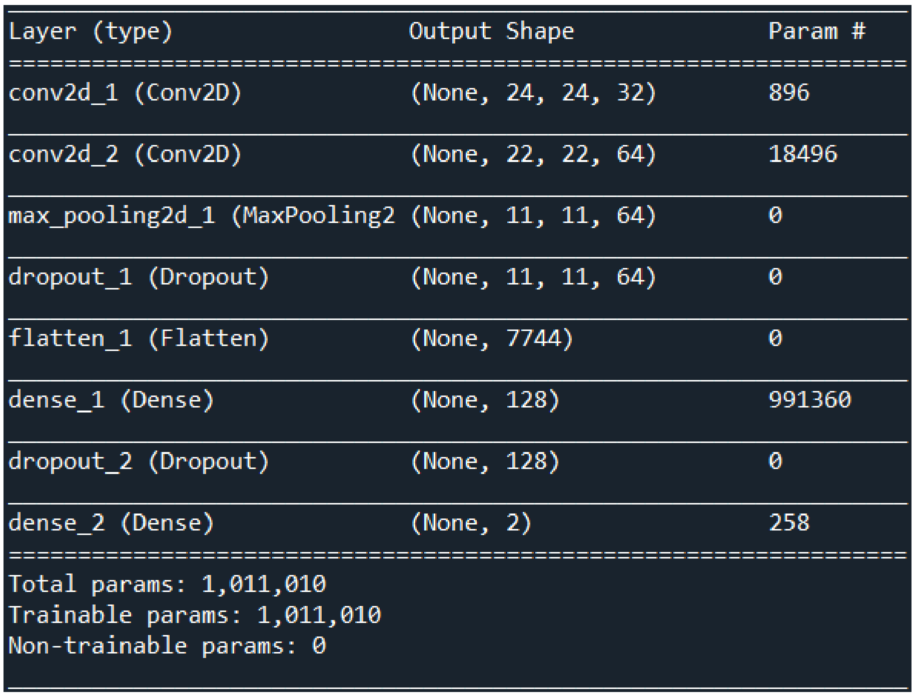
Figure 4.2: Examples of the resulting images after applying some transformation functions.

4.3 Deep Learning Algorithm

The classification model used in the prototype was adapted from the script published on the same page of the data set on the Kaggle platform. A convolutional neural network was used. The batch size of the model is 32 and the number of epochs is 20. The model summary is presented in Figure 4.3.

Each train of the CNN consisted in running 20 epochs using 5-fold cross validation.

One of the folds was used as validation set. Therefore, the size of the validation set is 20% of the training set. Ten repetitions of the training were performed, in each procedure, in order to record several classification evaluation metrics with their respective means and standard deviations.


```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 24, 24, 32)        896

conv2d_2 (Conv2D)            (None, 22, 22, 64)        18496

max_pooling2d_1 (MaxPooling2 (None, 11, 11, 64)        0

dropout_1 (Dropout)          (None, 11, 11, 64)        0

flatten_1 (Flatten)          (None, 7744)              0

dense_1 (Dense)              (None, 128)               991360

dropout_2 (Dropout)          (None, 128)               0

dense_2 (Dense)              (None, 2)                 258
=================================================================
Total params: 1,011,010
Trainable params: 1,011,010
Non-trainable params: 0
```

Figure 4.3: Classification model summary.

4.4 Evolutionary Strategy

As mentioned before, an evolutionary strategy was chosen to integrate the prototype. The components defined for the ES are described below:

- Initialisation: Both the number of genes and the value of the genes are randomly generated for the first individual of the first generation.

- Population: The population size does not vary throughout evolution, keeping its value constant of 5 individuals.

- Representation: The representation of individuals is made by using real numbers. Each individual can be between one and five genes length. A gene, in the genotype space, matches a transformation function, in the phenotype space. Therefore, each candidate is a sequence of one or more transformation functions, i. e., a data augmentation strategy.

- Variation operators: Mutation is the only way to create offspring. It can occur through three different operations: addition, removal and alteration. Addition consists of inserting a new gene with a random value in a random position of the individual. Removal consists of deleting a gene at a random position in the individual. Alteration consists in exchanging the value of a random gene of the individual for a random value. Note that removal is not possible if the individual has only one gene, nor addition if the individual already has 5 genes.
- Fitness function: The fitness function used in the prototype is the ROC AUC score. The justification for this choice is presented in the next subsection.
- Selection: During the selection process only one individual is selected as a parent. The selection occurs in a deterministic and elitist way, i. e., the fittest one is chosen. The parent, in addition to breeding all the offspring, also passes on to the next generation.
- Termination condition: The evolution ends after a certain number of generations. In this work, experiments of 50 and 100 generations will be carried out.

Figure 4.4 shows a diagram of the working flow of the projected evolutionary strategy.

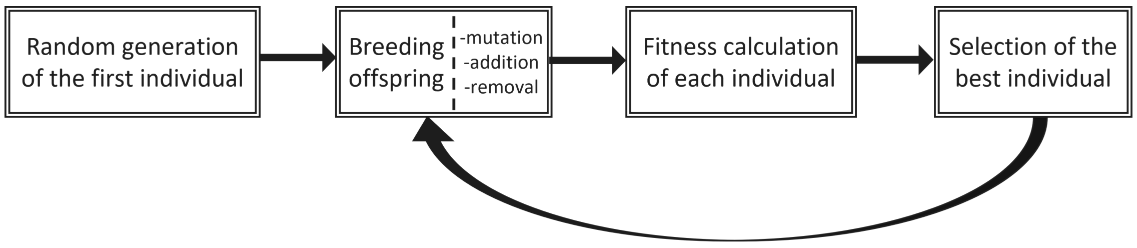

Random generation of the first individual
Breeding offspring
-mutation
-addition
-removal
Fitness calculation of each individual
Selection of the best individual

Figure 4.4: Schematic representation of the working flow of the designed evolutionary strategy.

For each procedure there will be a table, called the input table, which will reflect the possible gene values and their corresponding FT. This means that individuals will only be composed by values that are in the input table.

4.4.1 Fitness Function

The goal of the evolutionary process is to optimise the search process for the data augmentation strategy that maximises the performance of the classification task. This performance must be measured over the test data set. However, during evolution there is no access to the test data set, only to the validation and training sets. Hence, comes the need to extrapolate the performance under test from the performance under validation.

In order to evaluate the process of extrapolating the test performance based on the validation performance, a series of steps were performed. Firstly, the 136 transformation functions mentioned in Tables 4.2 and 4.4 were used as data augmentation strategies, individually, i. e., 136 different data augmentation strategies composed only by one transformation. Each data augmentation strategy was applied to the training data set without changing its size. Then, the augmented data sets were used to train the classification model (the CNN described early in this chapter) during 20 epochs using 5 folds. Ten repetitions of the training were performed for each of the 136 TFs and several classification evaluation metrics were recorded.

With the collected metrics, a study of the correlation between the test metrics and the validation metrics was made.

	Spearman Correlation	**Pearson Correlation**
Accuracy	0,714	0,823
Loss	0,887	0,934
Sensitivity	0,996	0,996
Specificity	0,997	0,998
ROC AUC Score	0,991	0,996
Precision	0,995	0,997
F1 Score	0,747	0,853
Mean Average Precision	0,981	0,990

Table 4.5: Spearman and Pearson correlations between validation and testing metrics for several classification metrics.

As can be seen in Table 4.5, all metrics have a high Spearman coefficient (>0.7) showing a strong correlation between test and validation performance. This correlation is also illustrated in Figure 4.5. To perform the plot, the transformations functions were sorted in ascending order by their test ROC AUC score. On the horizontal axis are represented the ordered TF and on the vertical axis their respective ROC AUC scores, test in blue and validation in green. The similarity between the behaviour of the validation and test metrics for each TF is very evident. Therefore, the validation metrics can be used as good indicators of the state of the evolutionary process, since the test performance can be inferred from the validation performance.

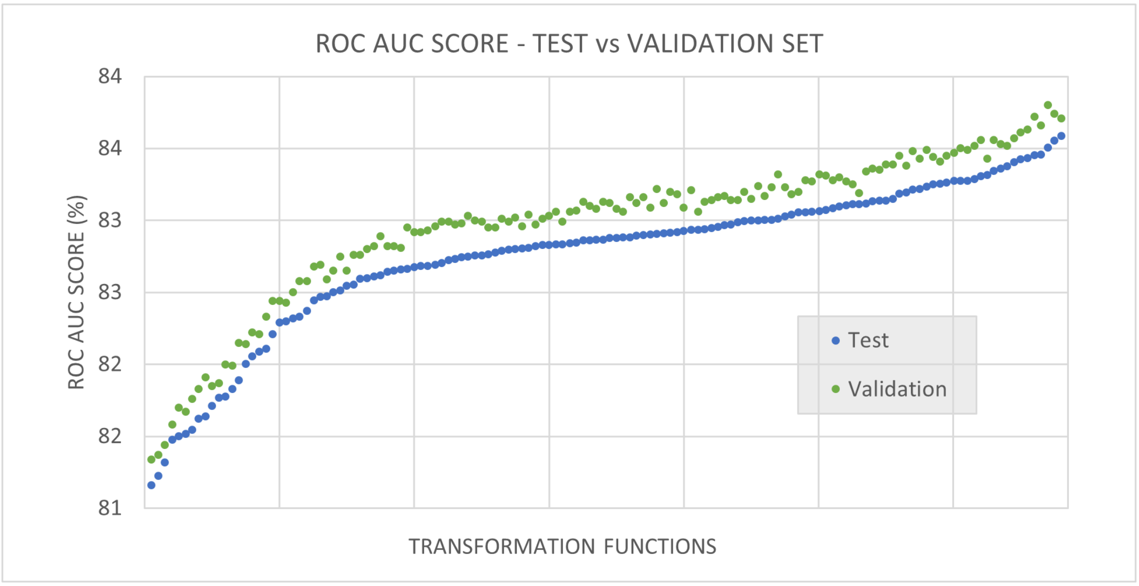


Figure 4.5: Graphic illustration of the correlation between test and validation performance.

The ROC AUC score metric will be used as the fitness function of the prototype's evolutionary strategy. This metric was chosen as an indicator of test performance from the validation as it presents a very strong correlation between the results of the two sets (Spearman's coefficient = 0.991). Besides, it is considered a comprehensive and widely used metric, well accepted for analysing medical imaging tests, which expresses a trade-off between specificity and sensitivity [30].

4.5 Preliminary Tests

To find the best configuration for the prototype and to evaluate its operation, some preliminary tests were carried out. These tests will allow to answer the following questions: Is the evolutionary process working? What is the most gainful training data set size during evolution? How many generations should the evolutionary algorithm last?

As mentioned before, several classification metrics were recorded by training the classification model with the 136 different transformation functions. Of these metrics, the validation ROC AUC score was chosen to be the fitness function. Figure 4.6 depicts the distribution of ROC AUC validation scores of the TFs when applied to the training data set individually. Analysing the graphic is visible that the TFs belonging to the geometric group can reach higher scores, as reported in the literature [31]. To perform the preliminary tests, the TFs were sorted by their ROC AUC validation score.

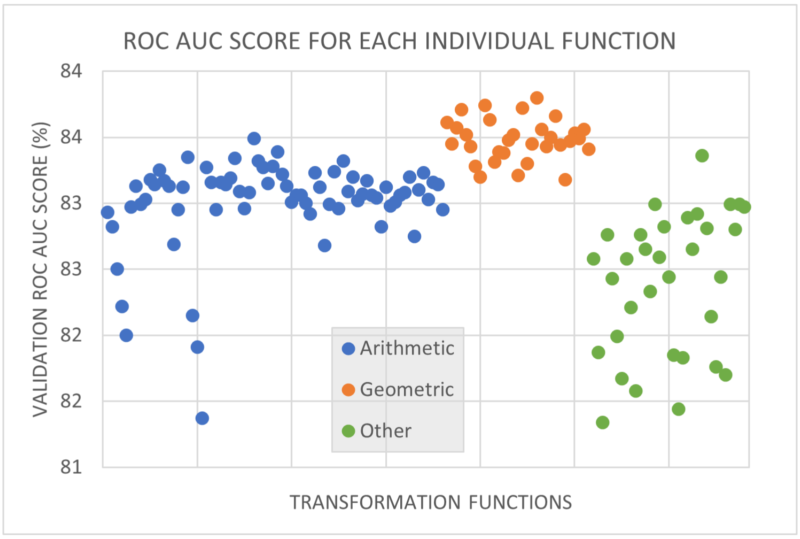


Figure 4.6: Graphic illustration of the distribution of the 136 TFs by their ROC AUC validation score, when employed individually to augment the training data set.

▪ Is the evolutionary process working?

To find this answer, from the sorted 136 TFs, the best one and the worst nine were selected to make the input table. The input table for this test is Table 4.6

In the context of the evolutionary algorithm, the genotype space is formed by integers from 0 to 9 (column "Gene" of Table 4.6) that map to the ten TFs in the phenotype space (column "Transformation Function" of Table 4.6). The gene 0 maps the function with the highest validation ROC AUC score and the genes 1 through 9 map the nine functions with the lowest scores. The goal of this experiment is to test out that the algorithm is evolving. Therefore, it is expected that the process evolves towards isolating the highest score function (gene 0) or towards a combination of TFs with a score higher than the TF score represented by the gene 0.

Table 4.7 identifies the best individuals of each generation, i. e., those with the highest fitness function value. Looking at the table, it is possible to verify that,

Gene	Transformation Function
0	"TranslateY(percent=(-0.2,0.2),mode='edge')"
1	"MultiplyBrightness((0.5,1.5))"
2	"Multiply((0.6,1.4))"
3	"LinearContrast((0.5,1.5))"
4	"MultiplySaturation((0.5,1.5))"
5	"AddToBrightness((-70,70))"
6	"pillike.EnhanceBrightness(factor=(0.7,1.3))"
7	"pillike.EnhanceContrast(factor=(0.5,1.5))"
8	"HistogramEqualization()"
9	"LinearContrast((0.7,1.3))"

Table 4.6: Input table consisting of the function with the highest ROC AUC validation score and the 9 functions with the lowest scores.

during the 50 generations, the best individual is the gene 0 isolated in 58% of them and in 100% of the cases, the gene 0 is part of the individual, as expected. This results confirm that evolution occurs.

Generation	0	1	2	3	4	5	6	7	8	9
Best Individual	[5 9 5 0]	[5 5 0]	[5 0]	[5 0]	[0]	[0]	[0]	[0]	[0]	[0]
Generation	10	11	12	13	14	15	16	17	18	19
Best Individual	[0]	[0]	[0]	[0]	[5 0]	[5 0]	[0]	[0]	[0 0]	[0 0]
Generation	20	21	22	23	24	25	26	27	28	29
Best Individual	[0 0]	[0]	[0]	[0]	[0]	[4 0]	[0]	[0]	[0]	[0]
Generation	30	31	32	33	34	35	36	37	38	39
Best Individual	[0]	[6 0]	[0]	[0]	[0]	[0]	[0 0]	[0]	[0]	[0 0]
Generation	40	41	42	43	44	45	46	47	48	49
Best Individual	[0 0]	[0 0]	[0 9]	[0 9]	[0 0 9]	[0 0]	[0 0]	[0]	[0]	[0 0]

Table 4.7: Individual with the highest value of fitness function per generation.

■ What is the most gainful training data set size during evolution?

To find the answer for this question, from the sorted 136 TFs, the best ten were selected to make the input table. The input table for this situation is Table 4.8.

Gene	Transformation Function
0	"TranslateY(percent=(-0.2,0.2),mode='edge')"
1	"Flipud(0.3)"
2	"TranslateX(percent=(-0.1,0.1),mode='edge')"
3	"Fliplr(0.4)"
4	"Rotate((-40,40),mode='edge')"
5	"Flipud(0.4)"
6	"Fliplr(0.1)"
7	"Fliplr(0.3)"
8	"Rotate((-10,10),mode='edge')"
9	"ShearY((-20,20),mode='edge')"

Table 4.8: Input table consisting of the 10 functions with the highest ROC AUC validation score.

To evaluate this question, four different sizes of the training data set were tested during the evolution process. The training data set sizes used during evolution were the original size and 1/2, 1/4 and 1/8 of the original size. For each data set size, three runs of the evolutionary process during 50 generations were made using different seeds. The best solutions (individuals) found for each seed were data augmentation strategies that revealed superior validation score.

Note, that until now, there has been no interaction with the test data set, only the training and validation data sets take part in the evolution process. The DA strategies resulting from the evolutionary process must be evaluated to find their ROC AUC test score. This process is done using the original data set size and augmenting the training data set with the DA strategy in evaluation. Ten repetitions of the classification model training were performed during 20 epochs using 5 folds. Test and validation metrics were recorded.

A summary of the results for this preliminary test is shown in Table 4.9. Due to time constraints, for the original data set size, only 2 folds were used during the evolutionary process, instead of 5 as in the other cases. The average ROC AUC test score was achieved through the use of three different seeds. In bold is highlighted the result considered to be the best trade-off between performance and evolution time. Therefore, in the last preliminary test and for future use of the prototype, the size of the data set utilised during evolution will be fixed at 1/4 of the original size.

Evolution Data set Size	**Average ROC AUC Test Score for the Best Solution(%)**	**Evolution Time (h)**
Original	83,52 ± 0,98	∼ 40
Original/2	83,49 ± 0,97	∼ 50
Original/4	**83,60 ± 0,95**	∼ 25
Original/8	83,34 ± 1,41	∼ 12

Table 4.9: Summary table of the results obtained by using 3 different seeds for each data set size during evolution.

■ How many generations should the evolutionary algorithm last?

To answer this question, all the 136 TFs are going to be part of the input table. The input table for this situation is Table 4.4.

The evolution process was tested using two different numbers of generations, 50 and 100. As referred to, the training data set size used during evolution was 1/4 of the original size. For each number of generations, three runs of the evolutionary process were made using different seeds. The best solutions (individuals) found for each seed are data augmentation strategies that revealed a superior validation score.

Analogously to the last test,the DA strategies resulting from the evolutionary process must be evaluated to find their ROC AUC test score. To record the test and validation metrics, the classification model was trained ten times during 20 epochs with 5 folds. This process is done using the original data set size and augmenting the training data set with the DA strategy in evaluation.

A summary of the results for this test is shown in Table 4.10. Similarly to the question above, the average ROC AUC test score was achieved through the use of three different seeds. In bold is highlighted the result considered to be the best trade-off between performance and evolution time. Therefore, for future use of the prototype, the number of generations will be fixed at 50, when the input table is defined by TFs with predefined parameters.

Number of Generations	**Average ROC AUC Test Score for the Best Solution(%)**	**Evolution Time (h)**
50	**83,53 ± 0,80**	$\sim$ 35
100	83,18 ± 1,44	$\sim$ 65

Table 4.10: Summary table of the results obtained by using 3 different seeds for each data set size during evolution.

4.6 Summary

In this chapter, the configurations used in the prototype development were discussed and defined. The prototype can receive as input two different types of transformation functions, ones with fixed parameters and the others without predefined parameters. In the evolutionary strategy, populations are made up of 5 individuals and each individual can be between 1 and 5 genes length. New individuals are created solely through the mutation mechanism. The fitness function that will be used in evaluating solutions is the ROC AUC validation score. During the evolution process the training data set is reduced to 1/4 the size of the original data set. Depending on the experiment to be carried out, the total number of generations will be 50 or 100. The configurations presented will be used in the next chapter to execute the experiments.

5

Evolution of Data Augmentation Strategies

This chapter will describe the experimental setup of the experiments performed, as well as their results and discussion. Section 5.1 explains the entire structure and procedure followed in the delineation of the experiments. In section 5.2 the results obtained are presented and discussed.

5.1 Experiments

With the prototype delineated and the settings adjusted it is time to start the experiments. First, a baseline had to be defined. The baseline was established by training ten times the classification model, without any kind of data augmentation, during 20 epochs using 5 folds.

Thereafter, two experiments were devised and put into practice. The two experiments use different approaches to apply data augmentation strategies. In the first, the DA strategies are built with transformation functions that have predefined parameters (136 different TF – Table 4.4) and in the second, the transformation functions parameters are free (39 different TF – Table 4.3), being subject to changes during the evolutionary process. As already mentioned, the data augmentation strategies are used without increasing the data set size and the probability of each TF being applied on an image is 50%. The exception resides in flips transformations, where the parameter itself defines the probability of being applied to an image.

5.1.1 Fixed Parameters

In this experiment, the TFs with predefined parameters (Table 4.4) were used. In this procedure, the genotype of each individual consists of integers ranging from 0 to 135. Basically, the genotype works as an index that matches a given TF. Following this logic, the phenotype is the TFs themselves. During the evolutionary process, TF sequences are generated from the 136 available ones, which define DA strategies. Each population has 5 individuals. The size of an individual can range from 1 to 5 genes. Figure 5.1 exemplifies an individual.

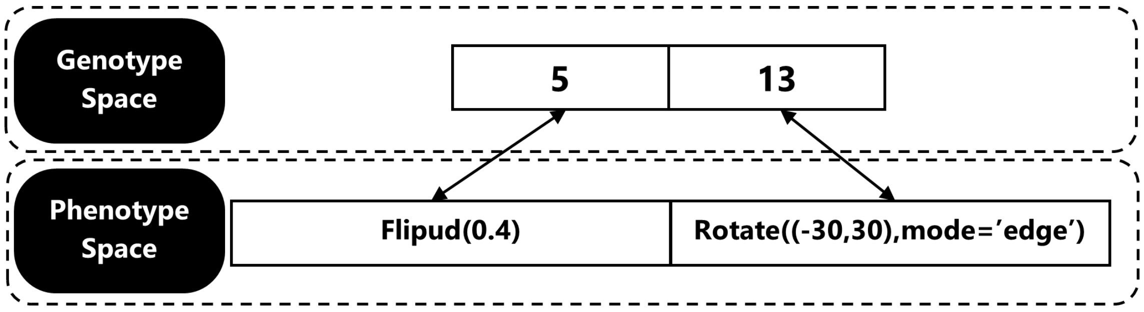


Figure 5.1: Illustration of the genotype and the phenotype of an individual with 2 genes in the fixed parameters experiment.

In each generation, only one individual is selected as a parent, the fittest one. The parent, in addition to breeding all the offspring, also passes on to the next generation. In the designed prototype, mutation is the only way to create offspring. Mutation can occur through 3 different operations: addition, removal, alteration. Addition consists of inserting a gene with a random value in a random position of the individual. Removal consists of deleting a gene at a random position in the individual. The alteration consists in exchanging the value of a random gene of the individual for a random value. All operations have the same probability of occurring, however, removal is not possible if the individual has only one gene, nor addition if the individual already has 5 genes. The mutation diagram is depicted in Figure 5.2. The offspring creation process ends when 4 different and viable offspring are created.

Initialisation is done randomly by defining an individual with random gene size and values. The evolutionary process is guided by an evaluation function that analyses each DA strategy to select the most suitable one. The evaluation function used by the evolutionary algorithm is the validation ROC AUC metric. This value is obtained by training the machine learning algorithm after applying the DA strategy under analysis on the image training data set. The best solutions or the best

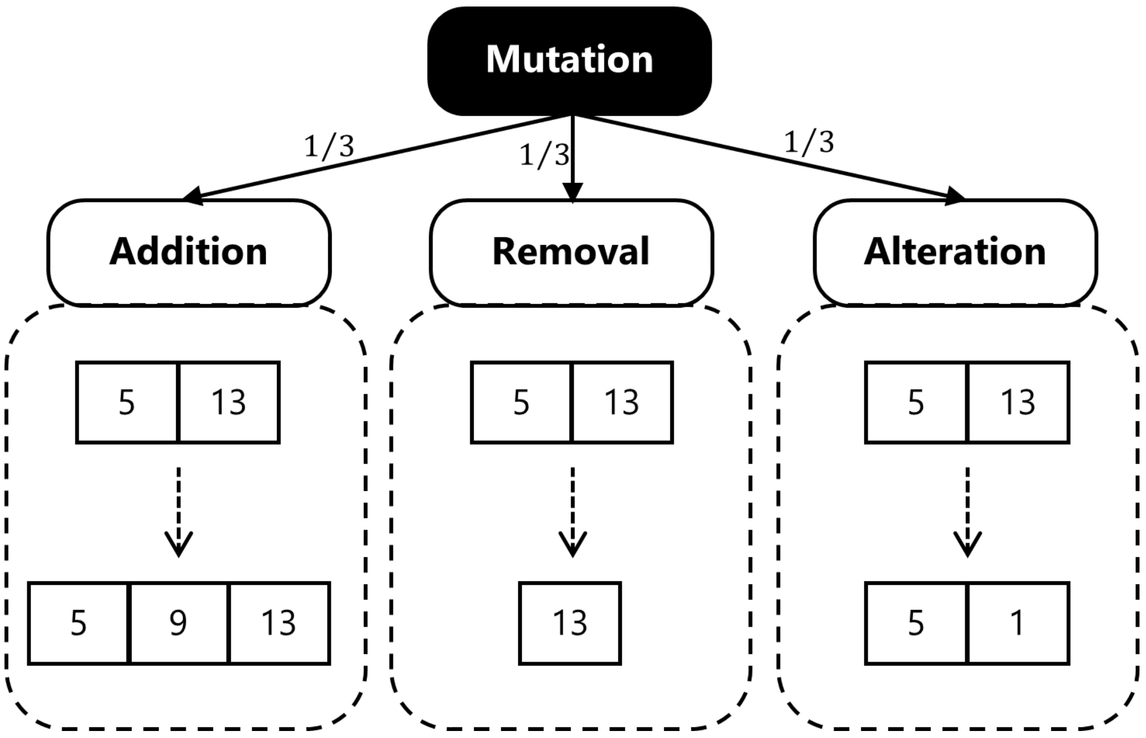


Figure 5.2: Schematic representation of the mutation mechanism in the fixed parameters experiment.

strategies are considered to be those with the highest evaluation function value.

As a result of the trade-off between time and performance, evolution is finished after reaching a certain number of generations, in this case 50. The size of the data set used during evolution has also been reduced to $\frac{1}{4}$ of the size of the original data set. The prototype was tested using 30 different seeds. Several characteristics of the population over the generations were recorded, as well as the differences in performance detected between the solutions obtained from the use of the prototype and the baseline (without DA).

5.1.2 Free Parameters

From the previous experience to this one, it was necessary to perform some changes, namely at the level of representation in the evolutionary algorithm. Below it is pointed out what these differences are. The rest of the specifications apply analogously to this experiment.

In this case, TFs with free parameters were used, i.e., the function parameters are also subject to modifications during the evolution process. The genotype of each individual is constituted by a tuple of two arrays. The first array is made up of positive integers ranging from 0 to 38 (number of TFs minus one). Without predefined parameters 39 different TF are available (Table 4.3). Each integer is seen as an index that matches a given TF with no associated parameters. The second array is composed by arrays of floats between 0 and 1. These values represent the parameters used by the functions represented in the first array. In other words, a gene is composed by two parts, an integer that encodes a TF plus an array of floats corresponding to the parameters used by the TF. The phenotype of each individual is represented by the TF with the corresponding parameters. Figure 5.3 exemplifies an individual in these conditions.

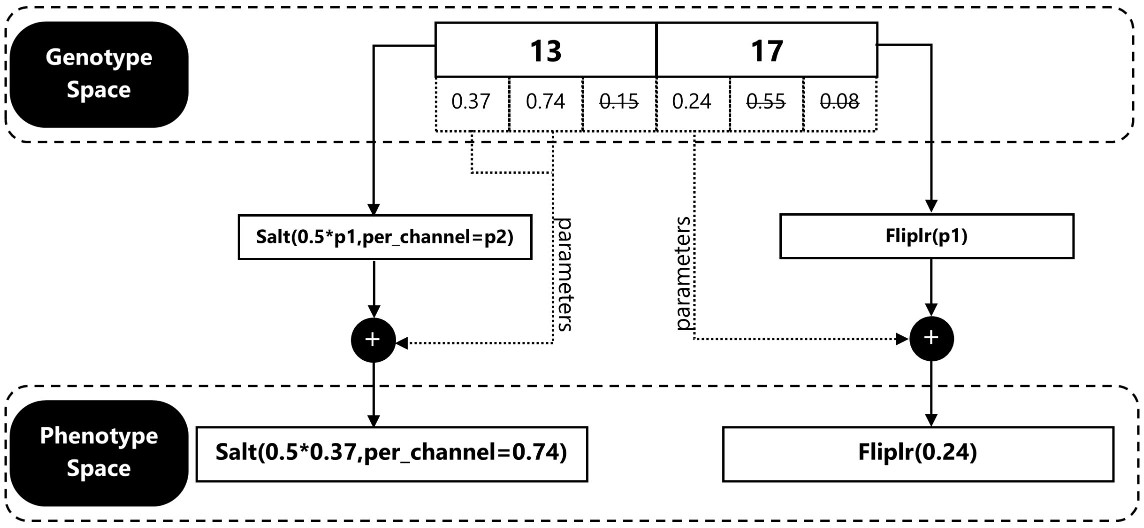


Figure 5.3: Illustration of the genotype and the phenotype of an individual with 2 genes in the free parameters experiment.

When the number of parameters used by a TF is inferior to the size of the parameters vector, the extra parameters are ignored. Making an analogy with biology, these extra parameters represent the non-coding DNA. This feature allows different genotypes to encode individuals with the same phenotype.

The mutation can occur at two distinct levels: at the level of the array that encodes the TFs (first part) or at the level of the parameters array (second part). There is a 50% chance of the mutation occurring on the first part and 50% on the second part. In the first part of the gene, the mutation proceeds in an analogous manner to the fixed parameters experiment. Three operations are possible: addition, removal, alteration with equal probability. When addition occurs, along with the insertion of a new integer in the first array, a new array of parameters corresponding to the new value is also inserted. When removal occurs, the array of parameters related to the deleted integer is also discarded. In case of alteration, there are no changes in the parameters array, only the integer is modified. If the mutation occurs at the parameter level, the value of a random parameter is exchanged for a random number, keeping the first array of the tuple intact. This mechanism is illustrated in Figure 5.4.

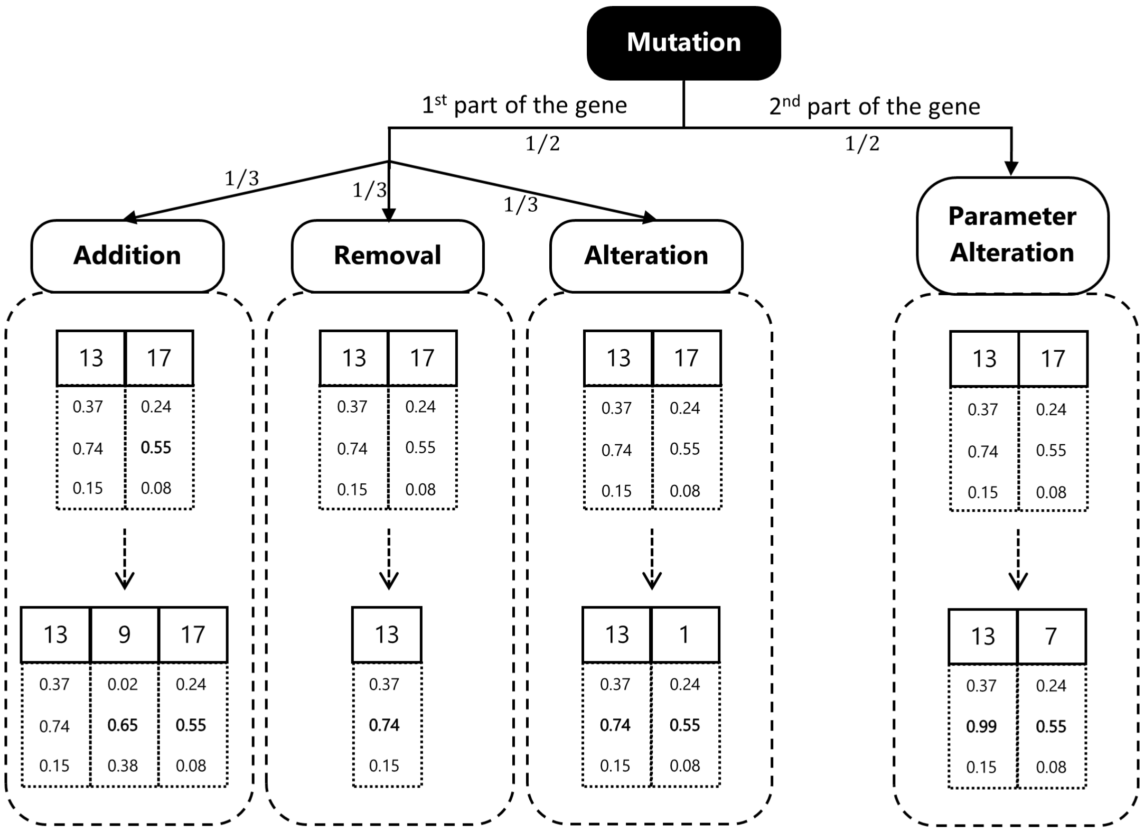


Figure 5.4: Schematic representation of the mutation mechanism in the free parameters experiment.

As in the previous experiment, the breeding of offspring terminates when 4 viable and different individuals are created. Initialisation is done randomly by defining an individual with random size, gene value and parameter values. Evolution is finished after 100 generations. After the analysis of the preliminary tests in section 4.5 it was verified that the best trade-off between performance and evolution time was to use 50 generations. However, the preliminary tests were carried out with the TFs belonging to the Table 4.4, the same used for the fixed parameters experiment. For this experiment the search space is substantially wider, therefore it was decided to divide this experiment in two. In one, the evolution would end after 100 generations and in the other it would end after 50 generations. As in the previous procedure, the experiments were repeated using 30 different seeds, the training data set was 1/4 of the size of the original data set and the characteristics of the populations over the generations were recorded.

5.2 Results and Discussion

As already described in section 5.1, a baseline was established. The training of the classification model was executed 10 times to obtain an average value with a standard deviation. Each execution was composed by 5 folds and the model was trained during 20 epochs. No augmentation was applied. The baseline is shown in Table 5.1.

	ROC AUC Test Score (%)
Baseline	82,63 ± 1,09

Table 5.1: Baseline value for ROC AUC test score.

All experiments executed the prototype 30 times, using different seeds. As already said, the free parameters experiment will be divided in two, one called Free Parameters - 100, where the information of the 100 generations will be all used, and the other one called Free Parameters - 50, where all the information and results will be extracted only from the first 50 generations.

Figures 5.5, 5.6 and 5.7 show the average evolutionary optimisation curve for the experiments. All curves stabilise at practically the same performance level. This may be an indicator that the results of the three experiments are very similar.

A small increase on the performance across the 50 generations in Figure 5.5 is visible, but the increase in Figures 5.6 and 5.7 is more notorious. The interpretation of this result lies on the assumption that smooth image transformations (Table 4.4) can be more beneficial to increase the performance of classification tasks. As in Figure 5.5 the applied transformations are smooth, the improvement margin becomes smaller, and the evolution is less accentuated. On the other hand, as in free parameter experiments the parameters are free to take larger ranges of values, the initial solutions are more likely not to be the most suitable. Thus, evolution starts with lower performances, which leads to a sharper rise in fitness. Tapering to the most suitable parameter values is a little slower.

Another issue that implies the small delay in stabilising the evolutionary optimisa-

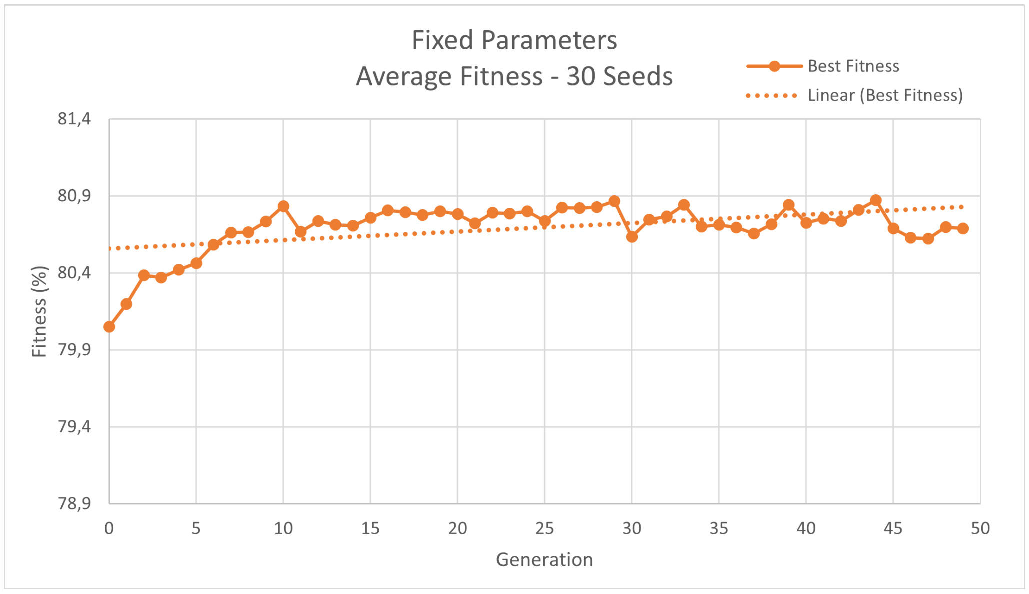


Figure 5.5: Evolutionary optimisation curve obtained during 50 generations for the fixed parameters experiment. The plot was obtained by averaging the fitness function of the best individual for each generation from 30 different seeds.

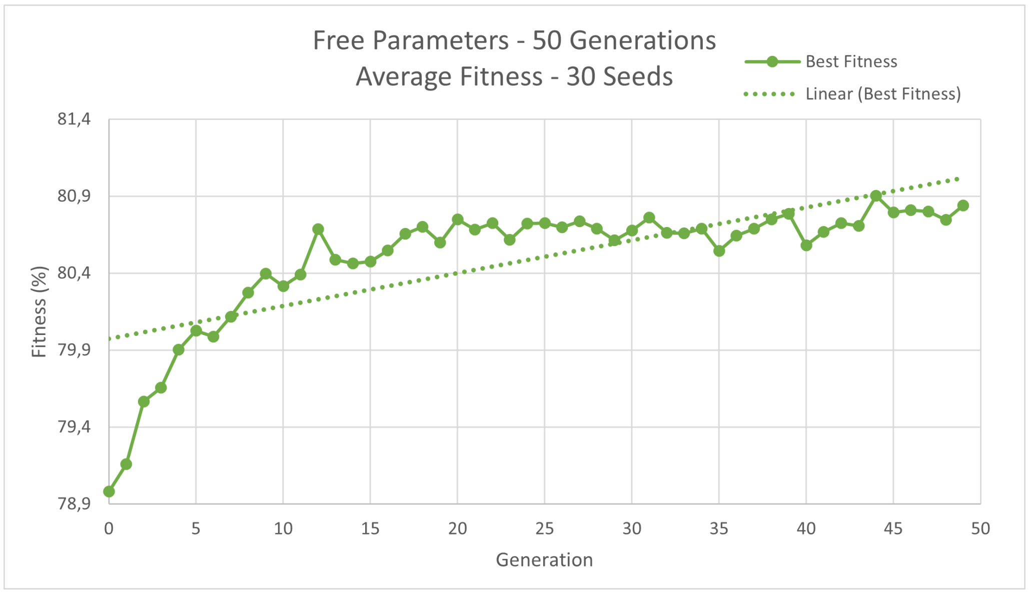


Figure 5.6: Evolutionary optimisation curve obtained during 50 generations for the free parameters experiment. The plot was obtained by averaging the fitness function of the best individual for each generation from 30 different seeds.

tion curve is the huge search space of the last two experiments. As the search space is so much higher than the fixed experiment, it is normal for the curve to take a little longer to stabilise.

For each seed in each experiment the best solution was found, i. e., the one with the

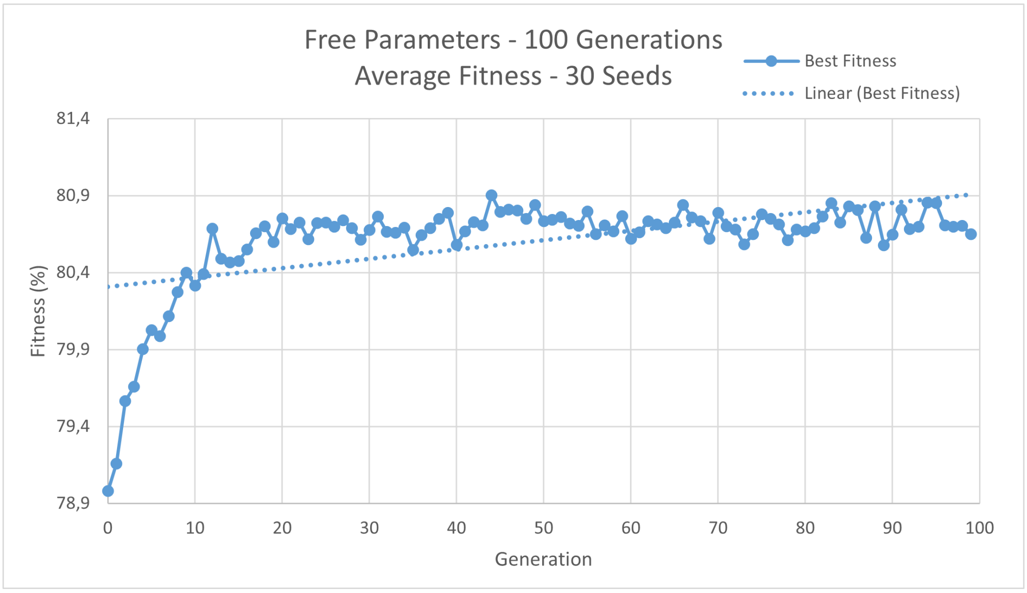


Figure 5.7: Evolutionary optimisation curve obtained during 100 generations for the free parameters experiment. The plot was obtained by averaging the fitness function of the best individual for each generation from 30 different seeds.

highest value. To obtain the ROC AUC test score of these solutions and to be able to compare them with the baseline, they were subjected to the same procedure applied to obtain the baseline. The training of the classification model was executed 10 times to obtain a mean value with a respective standard deviation. Ten executions of the training of the classification model were executed. Each training consisted in 5-fold cross validation with 20 epochs. The training data set was augmented with the DA strategy defined by the solution being processed.

Table 5.2 shows the results for the three experiments. All experiments exceeded the baseline. The Average ROC AUC Test Score and the Best ROC AUC Test Score are very similar between the three experiments. Further on, a statistical test will be carried out to assess or not the presence of significant differences between the results of the three experiments. The best ROC AUC test score for the three experiments outscored the baseline by 1,3%.

The most suitable DA strategies found for each experiment are shown below. Some examples of images resulting from the application of these strategies on the original image of Figure 4.2 are also provided in Figures 5.8, 5.9 and 5.10:

	Fixed Parameters	**Free Parameters-50**	**Free Parameters-100**
Generations	50	50	100
Average Evolution Time (h)	35,4	$\sim 29,8$	59,6
Average ROC AUC Test Score(%)	83,34 ± 1,03	83,33 ± 1,05	83,31 ± 1,21
Best solution	[35 33 20 19]	[18 24 22] [0,61 0,46 0,59] [0,68 0,49 0,26] [0,83 0,81 0,57]	[17 19 25] [0,38 1,00 0,47] [0,47 0,48 1,00] [0,84 0,54 0,03]
Best ROC AUC Test Score (%)	83,69 ± 0,60	83,70 ± 0,64	83,68 ± 0,68

Table 5.2: Summary table of the results obtained for each experiment. The mean values are obtained by averaging the results obtained by each seed.

■ Fixed Parameters Experiment

Genotype: [35 33 20 19]

Phenotype: ["Flipud(0.1)"
"TranslateX(percent=(-0.2,0.2),mode='edge')"
"Rotate((-50,50),mode='edge')"
"TranslateY(percent=(-0.1,0.1),mode='edge')"]

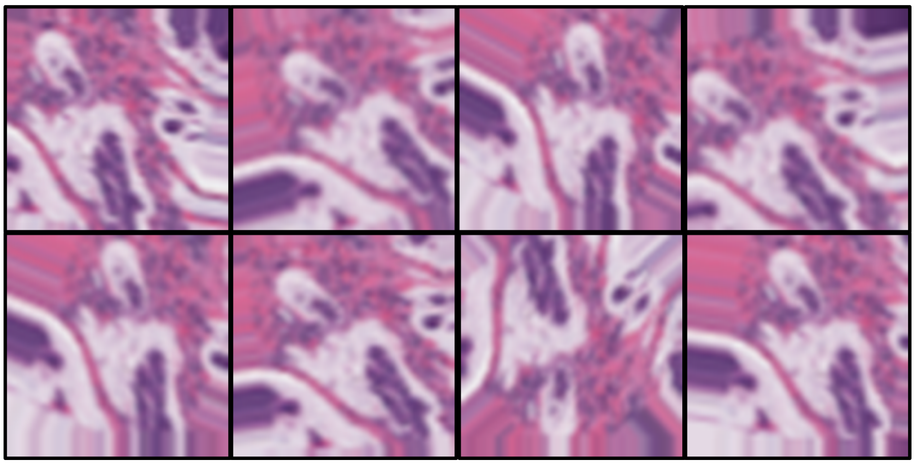


Figure 5.8: Examples of the resulting images after applying the best DA strategy found with the Fixed Parameters Experiment. The original image is shown in Figure 4.2.

■ Free Parameters - 50 Experiment

Genotype: [18 24 22]
[0,61 0,46 0,59]
[0,68 0,49 0,26]
[0,83 0,81 0,57]

Phenotype: ["Flipud(0.61)"
"ShearX((-0.46,0.49),mode='wrap')"
"TranslateY(percent=(-0.59,0.26),mode='symmetric')"]

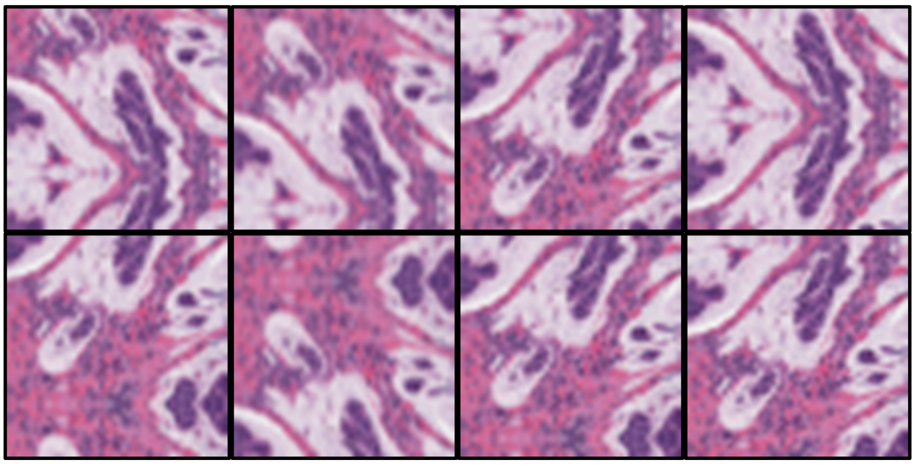


Figure 5.9: Examples of the resulting images after applying the best DA strategy found with the Free Parameters - 50 Experiment. The original image is shown in Figure 4.2.

■ Free Parameters - 100 Experiment

Genotype: [17 19 25]
[0,38 1,00 0,47]
[0,47 0,48 1,00]
[0,84 0,54 0,03]

Phenotype: ["Fliplr(0.38)"
"ScaleX((0.96,2.00),mode='symmetric')"
"ShearY((-0.47,1.00),mode='constant')"]

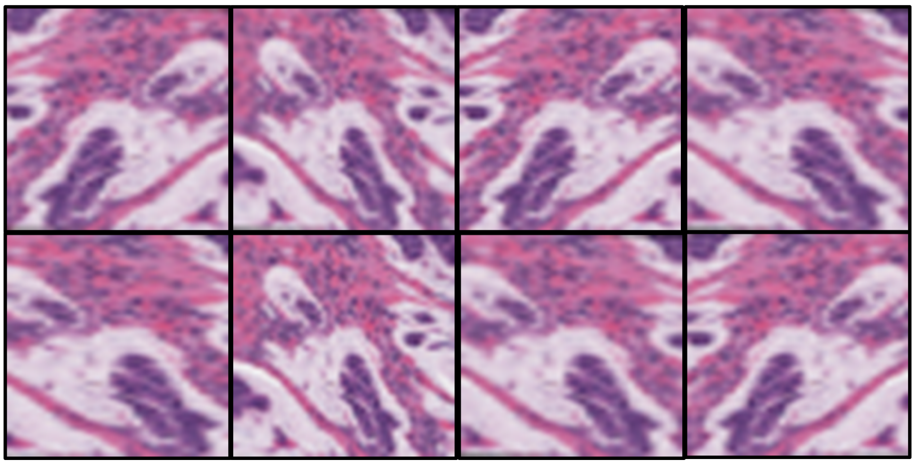


Figure 5.10: Examples of the resulting images after applying the best DA strategy found with the Free Parameters - 100 Experiment. The original image is shown in Figure 4.2.

Note that the data augmentation strategies found with the highest performances for each experiment are all made up only of transformation functions that belong to the geometric transformations group. As already mentioned, according to the literature, this is the type of transformation that usually results in a greater increase in classification performance. The results presented corroborate this statement.

Looking at the results of the three experiments, it can be seen that the values obtained are very similar to each other. To assess this similarity the Mann-Whitney U test was applied. The Mann-Whitney U test is a non-parametric test employed when comparing two independent groups. This test is commonly applied to assess if there is a statistically significant difference between two small-sized and non-normally distributed groups [32].

The Mann-Whitney U test hypothesis are:

H_0: the two groups have the same distribution.
H_1: the distributions of the two groups are different.[33]

Table 5.3 shows the p-values of the Mann-Whitney U test results between the fixed parameters experiment and the free parameters experiments. As can be seen from the table, the p-values are much higher than the stipulated confidence level of 0.05. The null hypothesis cannot be rejected and, therefore, it can be said that the samples have the same distribution. This means that the results from the different experiments do not show statistically significant differences.

	p-value
	Mann–Whitney U test
Fixed 50 vs Free 50	0,976
Fixed 50 vs Free 100	0,918

Table 5.3: P-values obtained by performing the Mann–Whitney U test between the experiments.

Of the three experiments, Free Parameters - 50 is the one with the most practical and automated architecture. It is more practical because compared to the other two, it has the lowest average evolution time. It is the most automated because no manual action is required to set parameters. Despite presenting lower scores at the beginning of the evolution compared to the fixed experiment, the evolution quickly led the algorithm to reach and stabilise at the same level as the fixed experiment. It presents an average score slightly lower than the fixed parameters, but higher than the free parameters - 100. The solution with the highest performance was obtained by this experiment. As there are no significant differences in the results of the three experiments, it can be concluded that the architecture of this experiment is the most advantageous.

6

Conclusions

Lately, a series of medical exams are delivered in image format. CAD systems are clinical decision support systems focused on image analyses. The use of these systems is being increasingly discussed to assist in areas such as diagnosis, prognostic prediction and disease detection. Deep learning techniques are a fundamental tool of the CAD systems. CNNs still have some challenges to overcome, especially regarding overfitting issues. DA is one of the techniques used to mitigate this issue. The effectiveness of classical image transformations in increasing the performance of classification tasks in medical imaging domain has been reported. However, a manual search is performed, in order to find the suitable augmentation strategy. This approach, mainly made of trial-and-error tasks can be very time-consuming and complex.

In this work, the development of a prototype capable of automatically augmenting data sets is proposed. The prototype consists of two algorithms, a deep learning algorithm and an evolutionary algorithm. The evolutionary algorithm has the function of automatically generating data augmentation strategies. The generation of strategies is achieved through an evolutionary process guided by an evaluation function. The deep learning algorithm has the role of evaluating each of the proposed strategies. To make some initial decisions regarding the construction of the prototype, a series of preliminary tests were carried out. The decisions were made in order to express the best trade-off between performance and consumed time and effort. Three experiments with different architectures were developed to evaluate which produced the best results using the built prototype.

The results demonstrate that there are no statistically significant differences between the performances obtained by the different experiments. In one of the experiments the functions used had predefined parameters, so there had to be a manual action to define them. So, in the other two experiments, that do not have fixed parameters, there was no manual action. This way, the last experiments are more automated and represent a preferred choice. The difference between these two was the number of generations used in evolution. In one 100 was used and in the other 50. As the results suggest, the performance of one experiment is practically the same as that of the other. Therefore, it can be concluded that the the Free Parameters - 50 experiment presents the best conjugation between automation, computational time, effort and performance.

In sum, the final prototype proposed has the same settings of the Free Parameters - 50 experiment. The data set size during evolution is 1/4 of the original size, the number of generations performed is 50, the fitness function is the ROC AUC score and the input table is made of free parameters functions. This is a new data augmentation approach, with an automatic search, that does not require the transformation functions to be defined in advance.

It should be noted that the prototype was designed with the aim of being flexible and being able to be applied to different situations. To do so, it is possible to change the classification model used by another more suited to the problem at hand. The number of generations, the individuals and population size are also easily modifiable.

The main contribution of this dissertation was the delivery of a generalised approach capable of automatically defining the most adequate data augmentation strategy for each specific problem.

For future work, a next step may go through testing the alteration of other parameters of the prototype besides those that were referenced to in this dissertation. Due to time constraints only a few parameters were modified and optimised. The population and individual's size is an aspect that can be studied in future work to seek relationships with performance. The probabilities of the various possible operations during the mutation mechanism are also relevant parameters to be examined. Finally, as the developed prototype has the potential of being applied in several problems, a logical next step would be to test it on multiple data sets and analyse its performance.

References

[1] Leila H. Eadie, Paul Taylor, and Adam P. Gibson. A systematic review of computer-assisted diagnosis in diagnostic cancer imaging. volume 81, pages e70–e76, 2012.

[2] Isabella Castiglioni, Leonardo Rundo, Marina Codari, Giovanni Di Leo, Christian Salvatore, Matteo Interlenghi, Francesca Gallivanone, Andrea Cozzi, Natascha Claudia D'Amico, and Francesco Sardanelli. Ai applications to medical images: From machine learning to deep learning. volume 83, pages 9–24, 2021.

[3] Youssef Skandarani, Pierre-Marc Jodoin, and Alain Lalande. GANs for medical image synthesis: An empirical study. volume abs/2105.05318, May 2021.

[4] Metin N Gurcan, Laura Boucheron, Ali Can, Anant Madabhushi, Nasir Rajpoot, and Bulent Yener. Histopathological image analysis: A review. volume 2, pages 147–171, 2009.

[5] Luisa F. Sánchez-Peralta, Artzai Picón, Francisco Miguel Sánchez-Margallo, and José B. Pagador. Unravelling the effect of data augmentation transformations in polyp segmentation. volume 15, pages 1975 – 1988, 2020.

[6] Agnieszka Mikołajczyk and Michał Grochowski. Data augmentation for improving deep learning in image classification problem. In *2018 International Interdisciplinary PhD Workshop (IIPhDW)*, pages 117–122, 2018.

[7] João Correia, Tiago Martins, and Penousal Machado. Evolutionary data augmentation in deep face detection. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '19, page 163–164, New York, NY, USA, 2019. Association for Computing Machinery.

[8] Krishna Chaitanya, Neerav Karani, Christian F. Baumgartner, Ertunc Erdil,

Anton Becker, Olivio Donati, and Ender Konukoglu. Semi-supervised task-driven data augmentation for medical image segmentation. volume 68, page 101934, 2021.

[9] Bradley J. Erickson, Panagiotis Korfiatis, Zeynettin Akkus, and Timothy L. Kline. Machine learning for medical imaging. volume 37, pages 505–515, 2017. PMID: 28212054.

[10] Diksha Sharma and Neeraj Kumar. A review on machine learning algorithms, tasks and applications. volume 6, pages 2278–1323, October 2017.

[11] Y. Li, B. Sixou, and F. Peyrin. A review of the deep learning methods for medical images super resolution problems. volume 42, pages 120–133, 2021.

[12] Andrew. Janowczyk and Anant. Madabhushi. Deep learning for digital pathology image analysis: A comprehensive tutorial with selected use cases. volume 7, page 29, 2016.

[13] Lei Cai, Jingyang Gao, and Di Zhao. A review of the application of deep learning in medical image classification and segmentation. volume 8, 2020.

[14] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. volume 6, pages 1–48, 2019.

[15] Arkadiusz Kwasigroch, Agnieszka Mikołajczyk, and Michał Grochowski. Deep neural networks approach to skin lesions classification — a comparative analysis. In *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*, pages 1069–1074, 2017.

[16] Eun Kyeong Kim, Hansoo Lee, Jin Yong Kim, and Sungshin Kim. Data augmentation method by applying color perturbation of inverse psnr and geometric transformations for object recognition based on deep learning. volume 10, 2020.

[17] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. 2017.

[18] Sebastien C. Wong, Adam Gatt, Victor Stamatescu, and Mark D. McDonnell. Understanding data augmentation for classification: When to warp? In *2016 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, pages 1–6, 2016.

[19] Yan Xu, Ran Jia, Lili Mou, Ge Li, Yunchuan Chen, Yangyang Lu, and Zhi Jin. Improved relation classification by deep recurrent neural networks with data augmentation. 2016.

[20] Talha Iqbal and Hazrat Ali. Generative adversarial network for medical images (mi-gan). volume 42. Springer Science and Business Media LLC, October 2018.

[21] Hoo-Chang Shin, Neil A Tenenholtz, Jameson K Rogers, Christopher G Schwarz, Matthew L Senjem, Jeffrey L Gunter, Katherine Andriole, and Mark Michalski. Medical image synthesis for data augmentation and anonymization using generative adversarial networks. 2018.

[22] Cristina Vasconcelos and Bárbara Vasconcelos. Increasing deep learning melanoma classification by classical and expert knowledge based image transforms. February 2017.

[23] Mohamed Elgendi, Muhammad Umer Nasir, Qunfeng Tang, David Smith, John-Paul Grenier, Catherine Batte, Bradley Spieler, William Donald Leslie, Carlo Menon, Richard Ribbon Fletcher, Newton Howard, Rabab Ward, William Parker, and Savvas Nicolaou. The effectiveness of image augmentation in deep learning networks for detecting covid-19: A geometric transformation perspective. volume 8, page 153, 2021.

[24] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. 2018.

[25] Blas Galván, David Greiner, Jacques Periaux, Mourad Sefrioui, and Gabriel Winter. Parallel evolutionary computation for solving complex CFD optimization problems : A review and some nozzle applications. December 2003.

[26] A. E. Eiben and James E. Smith. *Introduction to Evolutionary Computing*. Springer Publishing Company, Incorporated, 2nd edition, 2015.

[27] Conor Ryan. Evolutionary algorithms and metaheuristics. In Robert A. Meyers, editor, *Encyclopedia of Physical Science and Technology (Third Edition)*, pages 673–685. Academic Press, New York, third edition edition, 2003.

[28] Angel Cruz-Roa, Ajay Basavanhally, Fabio A. González, Hannah Gilmore, Michael D. Feldman, Shridar Ganesan, Natalie Shih, John E. Tomaszeweski, and Anant Madabhushi. Automatic detection of invasive ductal carcinoma in

whole slide images with convolutional neural networks. In *Medical Imaging*, 2014.

[29] Zeshan Hussain, Francisco Gimenez, Darvin Yi, and Daniel Rubin. Differential data augmentation techniques for medical imaging classification tasks. volume 2017, pages 979–984, 04 2018.

[30] Charles E. Metz. Roc analysis in medical imaging: a tutorial review of the literature. In *Radiological physics and technology vol. 1*, January 2008.

[31] Luke Taylor and Geoff Nitschke. Improving deep learning using generic data augmentation. volume abs/1708.06020, 2017.

[32] Jeffrey E. Harris, Carol Boushey, Barbara Bruemmer, and Sujata L. Archer. Publishing nutrition research: A review of nonparametric methods, part 3. volume 108, pages 1488–1496, 2008.

[33] Nadim Nachar. The mann-whitney u: A test for assessing whether two independent samples come from the same distribution. volume 4, March 2008.