Neuroevolution of Convolutional Neural Networks for COVID-19 Classification in X-ray Images

GUSTAVO ADOLFO VARGAS HÁKIM

B.Eng

Universidad Veracruzana

Supervisor: Dr. Efrén Mezura Montes
Associate Supervisor: Dr. Héctor Gabriel Acosta Mesa

A thesis submitted in fulfilment of
the requirements for the degree of
Master in Artificial Intelligence

Artificial Intelligence Research Center
Universidad Veracruzana
Xalapa, Veracruz, Mexico

28 June 2021

CONTENTS

List of Figures

Dedications

I wholeheartedly dedicate this study to my parents and uncle Antonio, who have always supported me unconditionally to pursue my goals and dreams, for giving me the tools to thrive personally and professionally, and most importantly, for their outright love. I also dedicate my thesis to my grandparents, who continue inspiring me from heaven.

Acknowledgements

I wish to express my deepest gratitude to my thesis supervisors, Dr. Efrén Mezura Montes and Dr. Héctor Gabriel Acosta Mesa, for their invaluable support at providing me the knowledge, the direction, and most importantly, conveying me the passion for research. Their persistent guidance was a key element toward the development of my work.

I would like to also recognize the members of my thesis committee and jury, Dr. Ericka Janet Rechy Ramírez, Dr. Hector Gabriel Acosta Mesa, Dr. Marcela Quiroz Castellanos and Dr. Carlos Artemio Coello Coello, whose feedback and advice allowed me to greatly improve the quality of my thesis.

I wish to also pay my special thanks to my professors at the Artificial Intelligence Research Institute, for their training and instruction, as well as for inspiring me toward the goal of doing high-quality research.

In addition, I would like to acknowledge my friends and professors who have accompanied my throughout my academic and professional endeavors, and whose friendship has played a meaningful role in my life, professionally and personally.

This research work was also possible thanks to the support from CONACyT through a scholarship to pursue my master's studies.

Abstract

Deep Artificial Neural Networks have been at the forefront of the Artificial Intelligence revolution. These powerful computational models loosely mimic the intricate structures of the brain. They have demonstrated an impressive performance at many tasks such as computer vision, language translation, autonomous driving, among others. Particularly, Convolutional Neural Networks have been one of the most popular branches of Deep Learning. Nonetheless, the design and implementation of such algorithms is not trivial.

In the field of Neuroevolution, the design of Artificial Neural Networks is automated by the use of Evolutionary Algorithms. These techniques have demonstrated an enormous potential to deal with the complexity of applying these deep networks. Recently, major advances on neural encodings have been introduced; neural encodings are an abstract representation of a computational *genome* that can be evolved and mapped into an actual deep neural network.

In this thesis, the effects of a newly proposed neural encoding to evolve Convolutional Neural Networks are studied. It is argued that the design of a suitable computational representation can highly improve the design of these networks' architectures, by achieving high performances with low computational costs. A new Neuroevolution framework was designed in order to test that certain encodings are more desirable than others in specific scenarios, such as the classification of chest X-ray images for COVID-19 detection, which is the prospective case study. It has been discovered that a more compact representation promotes the discovery of more competitive architectures in both single- and multi-objective evolutionary optimization. Furthermore, these automatically designed networks outperformed several of the handcrafted *state-of-the-art* methods, including DarkCovidNet, Bayes SqueezeNet, Xception + ResNetV50, ResNet18, EfficientNet-B0, VGG-16, Xception, MobileNet V2, and CNN + LSTM by obtaining an equivalent accuracy (96.7% of accuracy, which is in the range

$[0.8702, 0.9934]$ found in the reviewed literature) but significantly reducing the complexity (32107 against the smallest handcrafted architectures of 1.164 million parameters).

CHAPTER 1

Introduction

Convolutional Neural Networks (CNN) are presumably the most representative architecture of Artificial Neural Networks (ANN) within the field of Deep Learning (Khan et al. 2020; LeCun, Bengio, and Hinton 2015). These networks have been at the forefront of different tasks in Machine Learning and Computer Vision, such as image classification (He et al. 2016; Huang et al. 2017; Kizhevsky, Sutskever, and Hinton 2012; Rawat and Wang 2017), image segmentation (Badrinarayanan, Kendall, and Cipolla 2017; Ronneberger, Fischer, and Brox 2015; Taghanaki et al. 2020), object detection (Girshick 2015; Redmon et al. 2017; Zhao, Zheng, et al. 2019), among others. These algorithms have also extended their application scope to areas such as audio processing (Han, Kim, and Lee 2016) and text classification (Kim 2014).

Since 2012, there have been a considerable number of advances towards the improvement of CNNs, including skip connections (He et al. 2016; Huang et al. 2017), depthwise convolutions Chollet 2017, and regularization methods such as *dropout* (Srivastava et al. 2014). Although the progress has been significant, a lack of well-established notions on how to design these networks still remains, besides the empirical expertise of practitioners. The design and hyperparameters configuration for CNNs is a time consuming, trial-and-error process that requires of expert knowledge in both Deep Learning and the application domain (Sun, Xue, et al. 2019c), for which Neuroevolution aims to assist.

Neuroevolution is a research field that for over three decades has advanced in the automatic design and training of ANNs through Evolutionary Computation (EC) techniques (Stanley, Clune, et al. 2019). Starting from simple approaches to train small neural networks in the late eighties, Neuroevolution has grown and expanded into the realm of Deep Learning,

including Convolutional Neural Networks. Since 2017, some of the main contributions in Neuroevolution of CNNs relate to the design of neural encodings.

A neural encoding is a genetic representation of an ANN. The encoding is the first and most important step to design an Evolutionary Algorithm (Eiben and Smith 2015a), as it defines the type of solutions that will be part of the search space. The hybrid encodings are one of the recent advances in the field. Nonetheless, these representations might be designed in such a way that there is an implicit bias towards larger networks. This is caused by the number of CNN components that are represented inside a single block of the encoding (i.e., modularity).

Neuroevolution aids researchers and practitioners to relief the usually long, trial-and-error process of building the highly complex CNNs. Unfortunately, most of the previous works have centered on solving problems related to benchmark datasets such as MNIST[1] or CIFAR-10[2], which are large sets of image data that often require larger networks to generalize. Large CNNs frequently entail higher training and inference times, sometimes demanding high-end computational resources such as Graphical Processing Units (GPU). This introduces the problem of applying current Neuroevolution techniques to niche problems. Niche problems, as presented in this document, are tasks related to datasets that tend to be smaller, and whose application relies on usually limited computational resources.

To address this issue, this thesis proposes that a Neuroevolution approach requires:

(1) A suitable search space led by a powerful neural encoding that encompasses neural architectures that solve the task at hand without excessive computational complexity measured by the number of trainable parameters, and compared with previously handcrafted architectures.
(2) A fitness mechanism that considers not only the networks' learning performance, but also its complexity, promoting a trade-off between both of them.

The aforementioned requirements are the target of this research project. To test the adequacy of this proposal, a niche problem is necessary. For this study, the chosen application is the

[1] MNIST is a dataset composed of binary images of hand-written digits.
[2] CIFAR-10 is a dataset of color images with ten different classes, such as bird, horse, and car.

classification of biomedical images. Biomedical images are regularly scarce (Calimeri et al. 2017), thus customized models are needed in order to better generalize and reduce the risk of overfitting. Furthermore, the application of learning algorithms to this domain *in the wild*, e.g., in hospitals, might be limited to low-end computational resources.

This biomedical imaging scenario provides for a useful environment to design customized CNNs with a moderate, almost optimal complexity. In particular, the diagnosis of Coronavirus Disease 2019 (COVID-19) in chest X-ray (CXR) images is adopted as the case study. The application of Artificial Intelligence techniques to this problem is of great interest, in view of the emergence of a global pandemic.

The standardize Reverse-Transcription Polymerase Chain Reaction (RT-PCR) testing approach has shown to be slow, expensive, low-specific and prone to failure (Ai et al. 2020; Beeching, Fletcher, and Beadsworth 2020). Due to these drawbacks, the analysis of CXR images has been explored as a potentially faster and effective alternative testing method. There is evidence of visibility of COVID-19-related symptoms in CXR images even when a negative result is obtained from the test (Kanne et al. 2020), which greatly improves classification rates.

Convolutional Neural Networks have demonstrated to be highly competent in the automatic analysis of CXR imagery (Baltruschat et al. 2019) and have obtained promising results COVID-19 diagnosis. Nonetheless, and as mentioned before, the design of these networks has been based on the designer's expertise, generally leading to large models. Recurrently, the *state-of-the-art* architectures (which were created for considerably large datasets) end up being applied, exceeding the complexity requirements and risking the performance on limited computing resources. To the best of the author's knowledge, Neuroevolution has not been applied to the COVID-19 classification on CXR images.

In this thesis, a Neuroevolution technique for the automatic design of Convolutional Neural Networks is introduced. This method is capable of building networks that classify lung conditions, including COVID-19 pneumonia, from CXR images without excessive complexity.

1.1 Problem Definition

The design of Convolutional Neural Networks for customized tasks is a time consuming, trial-and-error process that requires expertise in Deep Learning and the application domain. This often leads to the utilization of large networks, requiring high-end computational resources and being at risk of overfitting when data is scarce.

Neuroevolution deals with the automatic design of Convolutional Neural Networks architectures. However, the construction of a neural encoding plays a crucial role in improving or harming the search of architectures.

1.2 Research Proposal

The proposal of this work is to design a Neuroevolution algorithm to automatically design Convolutional Neural Networks for COVID-19 classification in CXR images. The system should comprise the following:

- **Encoding:** to design a flexible encoding leading to a search space of compact networks.
- **Search Algorithm:** to employ a Genetic Algorithm due to its known exploration-exploitation balance.
- **Fitness:** to use a fitness mechanism that considers both, the learning performance as well as the complexity of the networks.

1.3 Justification

Most of Neuroevolution methods concentrate only on classification performance, disregarding the networks' complexity. Multi-Objective Neuroevolution has been explored, considering the size of the networks as a second objective. Nonetheless, these methods rely on genetic encodings which can be biased towards larger models.

Automatic COVID-19 pneumonia classification in CXR images has the potential to compensate the lack of testing kits, and to assist for better diagnosis accuracy. Although Deep Learning has shown good results in this task, the Convolutional Neural Networks architectures remain being human-designed, leading to highly complex models.

1.4 Hypothesis

- **Hypothesis 1:** A Genetic Algorithm with a new neural encoding and a bi-objective fitness approach can find Convolutional Neural Networks able to classify lung conditions (including COVID-19) in CXR images with (a) an accuracy of 90% or more, (b) specificity and sensitivity values within the range of the *state-of-the-art*, and (c) a lower number of parameters with respect to the previously used architectures.
- **Hypothesis 2:** A hybrid encoding based on simple convolutional blocks and binary connectivity patterns helps to find less complex networks with respect to a hybrid encoding based on DenseBlocks (Wang, Sun, et al. 2019b)

1.5 General Objective

To use a Genetic Algorithm with a hybrid encoding to find an architecture of Convolutional Neural Networks capable of correctly classifying COVID-19 pneumonia in multi-class classification of CXR images with high performance and a smaller number of parameters than those in the *state-of-the-art*.

1.5.1 Specific Objectives

(1) To collect and annotate a CXR image dataset of COVID-19, bacterial/viral pneumonia, and healthy patients, and to apply a preprocessing procedure to augment the quality of images.

(2) To design a hybrid encoding for CNNs along with variation operators able to modify it.

(3) To design a fitness mechanism that considers the learning performance of a CNN as well as its complexity measured by its number of parameters.

(4) To execute a given number of experiments using the Genetic Algorithm, the proposed encoding and the fitness mechanism. Executions using a *rival* encoding from the *state-of-the-art* are also going to be carried out. A high-performance computing facility is to be used.

(5) To statistically analyze the experimental data to evaluate the hypotheses.

1.6 Contributions

The contributions of this thesis are presented below:

- A literature review and a proposed taxonomy of the existing encodings for Neuroevolution of CNNs.
- A novel flexible hybrid encoding able to generate compact Convolutional Neural Networks to classify COVID-19 in CXR images.
- A series of variation operators able to handle this new encoding. These operators form part of the Genetic Algorithm framework, called *DeepGA*.
- A new fitness mechanism based on a linear aggregate function, considering classification accuracy and the networks' complexity.
- Experimental evidence on the impact of two different hybrid encodings on the complexity of the networks during the search.

1.7 Chapter Summary

In this chapter, a general introduction to the thesis content has been provided. Convolutional Neural Networks have been presented as widely utilized and effective learning algorithms. However, the design of their complex architectures is a time consuming process that requires expert knowledge in Deep Learning and the application domain.

A Neuroevolution method is proposed, in order to automatically design Convolutional Neural Networks using a Genetic Algorithm. The chosen application domain is COVID-19 classification in Chest X-ray images. The contributions of this thesis are (1) a literature review on neural encoding for Neuroevolution of Convolutional Neural Networks, (2) the design of a genetic hybrid encoding capable of representing compact networks, (3) a set of variation operators to handle the new encoding, (4) a fitness mechanism that takes into account the networks' accuracy and their number of parameters, and (5) empirical evidence on the impact of two different hybrid encodings on the complexity of the networks during the search.

CHAPTER 2

Theoretical Framework and Background

In this chapter, a theoretical framework is presented in order to familiarize the reader with the key concepts on which this thesis is based. In essence, three main fields of research are broadly introduced: Evolutionary Algorithms, Convolutional Neural Networks, and Neuroevolution. A review on the *state-of-the-art* both on Neuroevolution of CNNs and pulmonary disease classification using CXR images will be also presented.

2.1 Evolutionary Algorithms

Evolutionary Algorithms (EA) are computational metaphors that mimic the process of natural evolution (Eiben and Smith 2015a). EAs are search and optimization algorithms that have demonstrated highly competitive performance in a wide variety of problems.

EAs are able to perform well in complex optimization problems with multiple variables, discontinuities, multiple objective functions, multiple global and local optima, among other challenges (Galván and Mooney 2020). Evolutionary Computation (EC) is the sub-field of Artificial Intelligence that groups this set of bio-inspired algorithms with a series of common characteristics as defined in (Baldominos, Saez, and Isasi 2019b), which are introduced next.

Population-based algorithms. An EA consists of a series of potential solutions, usually identified as *individuals*. This approach is useful to cover different regions of the search space at the same time. Here, the term *search space* is used as the set of all possible solutions to a given optimization problem. The operators in the EAs grant the individuals to move, explore and exploit the search space looking for an optimal solution.

Stochastic algorithms. The stochastic nature of an EA refers to its non-determinism. The effect of many evolutionary mechanisms inside the EAs possess a random component, making the output of the algorithm to be different in each execution. However, the randomness of an EA does not prevent it to have a robust behavior over many executions.

Metaheuristic algorithms. EAs do not require any application-specific knowledge, unlike heuristic methods that are highly related to the search problem at hand. Furthermore, an EA can be utilized in many different problems without changing its functioning. However, and mainly due to their stochastic nature, EAs cannot guarantee finding the optimal solution to a problem in most cases (ibid.).

There are two major families of bio-inspired algorithms: evolutionary algorithms, and swarm Intelligence (SI) algorithms. EAs take inspiration from the evolution of species, whilst SI algorithms are based on complex behaviors that emerge from the communication between simple individuals. There are some distinctive exemplars from the former, such as the Genetic Algorithm (GA), Evolution Strategies (ES), Evolutionary Programming (EP), Genetic Programming (GP), and Differential Evolution (DE). The SI group is composed mainly by Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), and Artificial Bee Colony (ABC) algorithm, among others. This thesis focuses on the Genetic Algorithm. An overview and analysis on the field of EC can be found in (Eiben and Smith 2015b).

2.1.1 The Genetic Algorithm

The Genetic Algorithm was originally proposed by John Holland in 1975 (Baldominos, Saez, and Isasi 2019b; Holland 1975) becoming on of the most popular representatives of EAs. This algorithm loosely mimics the basic principles of the Darwinian Evolution Theory of the survival of the fittest (Mirjalili et al. 2019).

The GA is an optimization and search algorithm, that has evolved and improved through the years to adapt to very diverse problems and representation schemes. Its general operation is based on a population of solutions that during a number of generations are selected to

recombine in order to travel the search space of a problem. Fig. 2.1 illustrates the general framework of a GA.

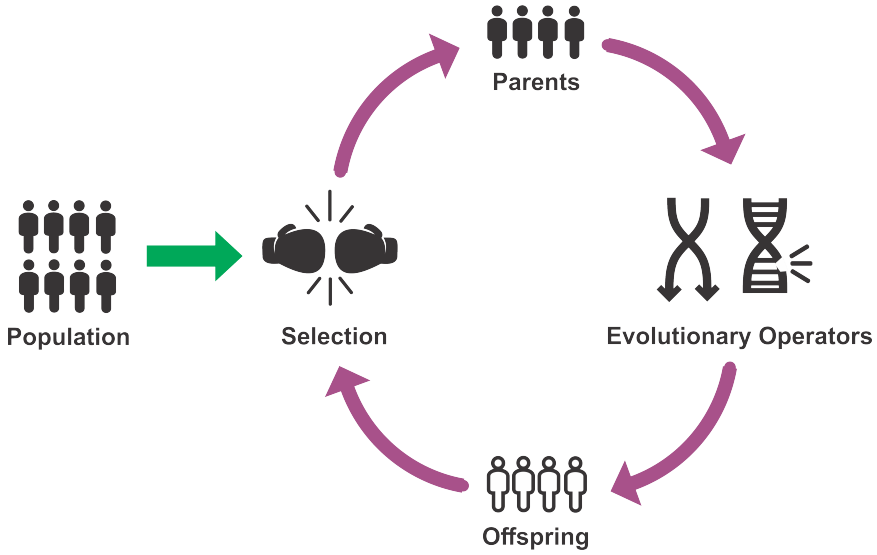

FIGURE 2.1: The general template of a Genetic Algorithm. A subset of parents is selected from an initial population to recombine and mutate in order to create the offspring, which are later used as the new population.

The GA is presented here as a symbiosis of three important factors: (1) genetic representation, (2) environment, and (3) evolutionary operators.

The **genetic representation** or *encoding* is a computational metaphor of the biological genome in living beings. The encoding is a mechanism that describes any potential solution in a search and optimization problem. As discussed by Eiben & Smith (2015a), the representation is the first and most important part in the design of any EA. One of the reasons for this is that the set of all possible instances of an encoding corresponds to the size and complexity of the search space **S**. For example, a numerical optimization problem can be solved by using a binary encoding of four bits to represent integer numbers in the range $[0, 15]$. On the other hand, the utilization of floating point numbers with four decimal positions as the encoding would offer more precision, at the cost of considerably increasing the size of the search space. Secondly, the evolutionary operators are chosen or designed based on the genetic encoding.

The genetic encoding can be compared to the *genotype* and to the *phenotype*. A genotype level encoding requires a decoding step to be evaluated. In numerical optimization, for example, a binary string needs to be transformed into a decimal number so as to be used as input for a numerical function. A phenotype level encoding requires no further processing in order to be evaluated.

The **environment** is the optimization problem to which the individuals need to adapt in order to survive. The environment, thus, is defined by the *fitness function* (also named objective function) to be optimized. A standard optimization problem, in the context of minimization, is defined as in Eq. (2.1b) (Boyd and Vandenberghe 2004):

$$\min_{\mathbf{x}} \quad f(\mathbf{x}) \tag{2.1a}$$

$$\text{subject to} \quad g_i(\mathbf{x}) \leq b_i, \quad i = 1, ..., m \tag{2.1b}$$

where the vector $\mathbf{x} = (x_1, ..., x_n)$ groups the n *decision variables* that are used to minimize the fitness function f whose mapping is $f : \mathbb{R}^n \rightarrow \mathbb{R}$. There are m constraint functions, which are limited by the constants b_i. The optimal vector $\mathbf{x}^*$ solves the problem of minimizing the function f given that there is no other vector $\mathbf{x}$ such that $f(\mathbf{x}) \leq f(\mathbf{x}^*)$, and by satisfying the constraints g_i. The variable vector $\mathbf{x}$ can be generalized to the previously mentioned genetic encoding. A fitness value is assigned to an individual by evaluating its encoding in the fitness function. The environment plays a crucial role in guiding the search process, as it determines which individuals are better adapted to solve a problem, against those who are not.

The **evolutionary operators** are in charge of performing the search. The GA is characterized by the presence of *crossover* as its flagship operator, and the *mutation* as a secondary operator. These two operators provide for mechanisms to expand the current population towards new solutions that are expected to be better as the generations go by.

Additionally, there is a *parents selection* procedure that allows the algorithm to choose some of the best individuals based on fitness. The parents selection process seeks to select

some of the individuals in the population for recombination (Drezner and Drezner 2020). Parents selection is in most cases a biased procedure towards the solutions with higher fitness. A stochastic component can or cannot be added. Four important selection families exist (Goldberg and Deb 1991): ranking methods (Baker 1985), proportional methods (Jong 1975), tournament methods (Brindle 1981), and genitor methods (Whitley 1989). For the proposed Neuroevolution technique in this thesis, the tournament selection framework is chosen due to its low computational complexity and ease of implementation (Goldberg and Deb 1991).

The deterministic tournament selection bases simply on randomly choosing a subset of s individuals (tournament), and selecting the one with the higher fitness (see Algorithm 1). This procedure is repeated until the parents pool P is full (usually $P = \frac{N}{2}$, with N being the population size).

Algorithm 1 Deterministic Tournament Selection

Require: A population *Pop*
Output: A parent solution $parent$.
Randomly Select s individuals for tournament from *Pop*.
Choose the individual with the highest fitness as $parent$.

A stochastic version of tournament selection can be devised by using a probability p that determines whether the best solution is chosen or not. If the best individual is not selected, the second best individual could be chosen with a probability $p(1-p)$; otherwise, the third best individual with a probability $p(1-p)^2$, and so on. The Algorithm 2 summarizes this procedure.

Algorithm 2 Stochastic Tournament Selection

Require: A population *Pop*
Output: A parent solution $parent$.
Select s individuals for tournament from *Pop*.
Sort the tournament by fitness from 0 to $s-1$ (k is the position of each individual).
if $U(0,1) \leq p(1-p)^k$ **then**
Choose the k-th best individual from the tournament as $parent$.
end if

The symbol $U(0,1)$ represents a random uniform distribution. Once the parents are selected, the recombination is performed. The crossover operator combines two *chromosomes* in order

to produce the offspring. In the GA, a series of offspring solutions are generated, being the number of them equal to the number of parents $|P|$ or to the population size N.

In crossover, two parent solutions are combined at the genotype level, unless they are encoded at the phenotype level, for which crossover is also possible. An important factor in crossover, is that the offspring should possess information inherited from the parents only; no new information should be added. The aforementioned is important as this ensures the exploitation of the search space. Exploitation consists on taking advantage of the region of the search space that is nearby the parents, hoping to find solutions with higher fitness. This could also lead to find a local optimum. The notion of exploitation is shown in Fig. 2.2-(a).

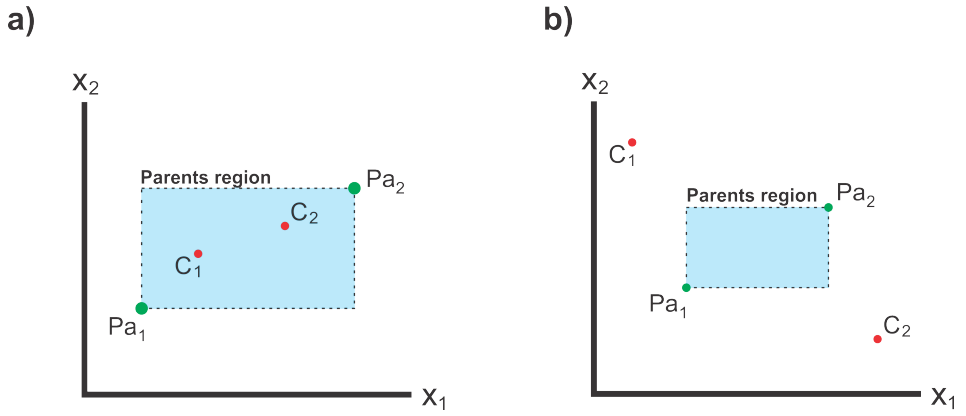

FIGURE 2.2: A scheme of the search process during recombination. A sample optimization problem of two variables x_1 and x_2 is presented. Two parents Pa_1 and Pa_2 form the *parents region*, colored in blue. **a)** Exploitation occurs when the offspring C_1 and C_2 are in the region enclosed by the parents. **b)** Exploration occurs when the offspring C_1 and C_2 are outside the parents' area.

Given two parents Pa_1 and Pa_2 that are randomly selected from the pool P, crossover is applied with a probability *CXPB* called *crossover rate*. Crossover is performed differently based on the encoding. In this thesis, one of the utilized representations is the binary string. One simple way to combine two of these encodings is using *single-point crossover*. Given two equally-sized binary strings of n bits, a random position i is selected. All the bits in the positions $i+1, ..., n-1$ are exchanged between both strings, as can be seen in Fig. 2.3.

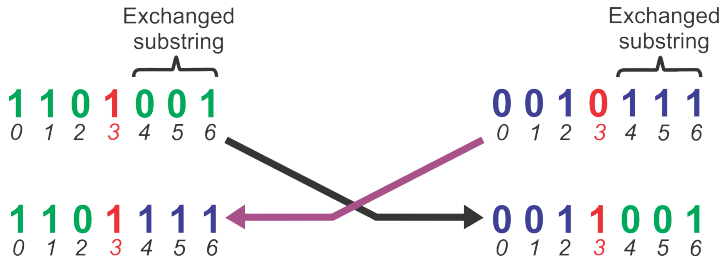

FIGURE 2.3: Single-point crossover between two binary strings. In this example, the cutoff point is in the position $i = 3$ (red colored). All the subsequent bits are exchanged between strings

In several cases, the crossover results in two new offspring solutions. As mentioned before, the new solutions are composed of a mixture of the information of the two parents, without inserting any new *knowledge*.

When the offspring solutions are produced, the mutation is applied with a probability *MUPB*, called *mutation rate*. The objective of mutation is to encourage exploration in the search space. Unlike crossover, the mutation seeks to position the offspring away from the parents' area to discover other potentially competitive regions to search (see Fig. 2.2-(b)). It also adds or changes the information in the offspring's encoding on a random, unbiased basis. In the binary string scenario, a simple mutation can be done by the *bit flip mutation*, in which a random bit i is selected: if its value is 0, it is flipped to 1 and vice versa.

After crossover and mutation take place, a *survival selection* or *replacement* is employed. This process consists in renewing the population with new solutions. In *generational* genetic algorithms, the entire population is replaced by the offspring at each generation. If the best solution is always preserved, it is said to be an *elitist* replacement approach. An elitist GA with a sufficiently large number of generations always converges to the optimal solution (Bhandari, Murthy, and Pal 1996). In this thesis, the chosen survival and replacement approach is the $\mu + \lambda$ replacement, which was introduced in Evolution Strategies (Mezura-Montes and Coello 2008). In the adaptation of this approach for a GA, the population set μ is merged with the offspring set λ. The entire set is sorted based on fitness, and the best $|\mu|$ individuals are chosen to be part of the next generation.

When the number of generations is used as a stop criterion, a GA is described in Algorithm 3. The crossover and mutation rates are user-defined parameters, as well as the number of generations T and the population size P. If tournament selection is used, the tournament size s is also a user-defined parameter.

Algorithm 3 Genetic Algorithm

Require: Population size N, number of generations T, tournament size s, crossover rate CXPB, mutation rate MUPB.
Output: A set of near-optimal solutions *Pop*.

```
Initialize population Pop with N random solutions.
t ← 0
while t ≤ T do
    Fill parents pool P with |P| parents with tournament selection.   //Parents Selection
    while Offspring set Off is not full do
        Randomly choose two parents Pa_1 and Pa_1
        if uniform(0,1) ≤ CXPB then
            Apply crossover between parents to obtain offspring C_1 and C_2.   //Crossover
        end if
        if uniform(0,1) ≤ MUPB then
            Mutate offspring C_1   //Mutation
        end if
        if uniform(0,1) ≤ MUPB then
            Mutate offspring C_2   //Mutation
        end if
    end while
    Sort Pop ∪ Off by fitness.
    Select the N individuals with higher fitness.   //μ + λ Replacement
    t ← t + 1
end while
```

A similar template is used for the project presented in this thesis. Being a metaheuristic algorithm, the GA can be applied to a variety of problems, including Neuroevolution.

2.1.2 Multi-Objective Genetic Algorithm

There is a family of GAs that deal with a special type of optimization problems. In different search and optimization scenarios, it is not possible to choose one solution based solely on one objective function. Multi-Objective Evolutionary Algorithms (MOEAs) (Deb et al. 2002) have gained considerable momentum by being highly competitive in Multi-Objective

Optimization problems (MOPs). The automatic design of Convolutional Neural Networks architectures can be posed as a Multi-Objective Optimization problem therefore it is relevant to discuss MOPs.

A multi-objective optimization problem can be defined as to find an n-dimensional solution vector $\mathbf{x} = [x_1, x_2, ..., x_n]^T$ that minimizes a set of m functions $\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_m(\mathbf{x})]^T$, where each variable $x_j \in [lo_j, up_j]$ (Vargas-Hákim, Mezura-Montes, and Galván 2020). The terms *lo* and *up* represent the lower and upper bounds of a variable, respectively. In multi-objective optimization, the objective functions are in conflict with each other; a solution that improves one of the functions, may worsen one or several of the others.

In these problems, it is not possible to sort the population based on one of the objectives and to expect that the top of these solutions is the best overall solution. Conversely, the population might contain a set of solutions that represent trade-offs among the objective functions. For this reason, a different sorting criterion is needed to highlight the best individuals from the others. Unlike single-objective optimization, where a single solution is found, in multi-objective optimization the best possible trade-offs among the objectives gives rise to a collection of solutions which are called the *Pareto Set*. To evaluate the fitness of the individuals inside this set, *Pareto dominance* is introduced. Mathematically, a solution vector $\mathbf{x} = [x_1, x_2, ..., x_n]^T$ dominates $\mathbf{y} = [y_1, y_2, ..., y_n]^T$, denoted as $\mathbf{x} \prec \mathbf{y}$, if and only if, $f_i(\mathbf{x}) \leq f_i(\mathbf{y})$ for all $i \in [1, ..., m]$ and $f_i(\mathbf{x}) < f_i(\mathbf{y})$ for at least one $i \in [1, ..., m]$. The above is in the context of minimization. The Pareto dominance between two points is presented in Fig. 2.4.

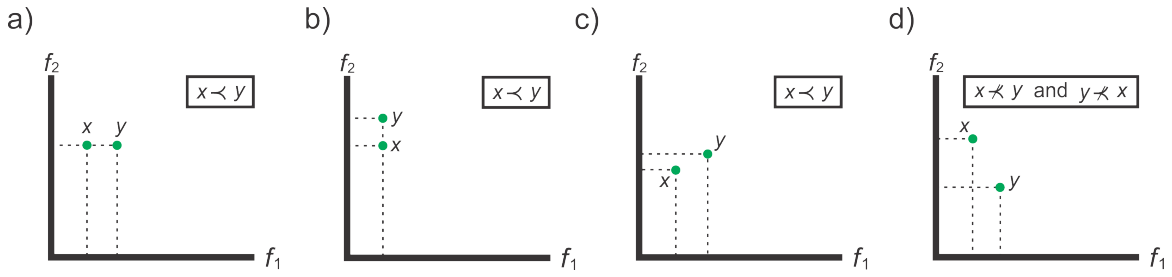

FIGURE 2.4: Pareto dominance between solutions x and y in a function space with two objective functions f_1 and f_2. **a)** x and y are equal on f_2 ($f_2(x) = f_2(y)$) but x improves f_1 in comparison to y ($f_1(x) < f_1(y)$). **b)** x and y are equal on f_1 ($f_1(x) = f_1(y)$) but x improves f_2 in comparison to y ($f_2(x) < f_2(y)$). **c)** x improves both f_1 and f_2 with respect to y ($f_1(x) < f_1(y)$ and $f_2(x) < f_2(y)$). **d)** neither x nor y dominate each other, as $f_1(x) < f_1(y)$ but $f_2(y) < f_2(x)$.

A solution $\mathbf{x}^*$ belongs to the Pareto Optimal Set P^* if there is not any other solution $\mathbf{x}$ such that $\mathbf{x} \prec \mathbf{x}^*$. The Pareto Optimal Front is therefore defined as $PF^* = \{\mathbf{f}(\mathbf{x}) | \mathbf{x} \in P^*\}$. It is important to further clarify that Pareto sets are composed of solutions, meanwhile Pareto fronts are composed of function values evaluated using the solutions in the Pareto sets.

In MOEAs, the population is to be sorted with respect to dominance (*non-dominated sorting*) (Deb 2007); the individuals are grouped in different sub-optimal Pareto fronts. These fronts are also ranked, being the first front the best, and the last front the worst. The first PF is composed of all the solutions $\mathbf{x}$ that do not dominate each other, but that dominate all the excluded solutions. The second front is composed of those excluded solutions that do not dominate each other but dominate a new group of excluded solutions. The process repeats until all the individuals are inside their corresponding fronts. Fig. 2.5 exemplifies how the individuals inside a population can be classified into different fronts.

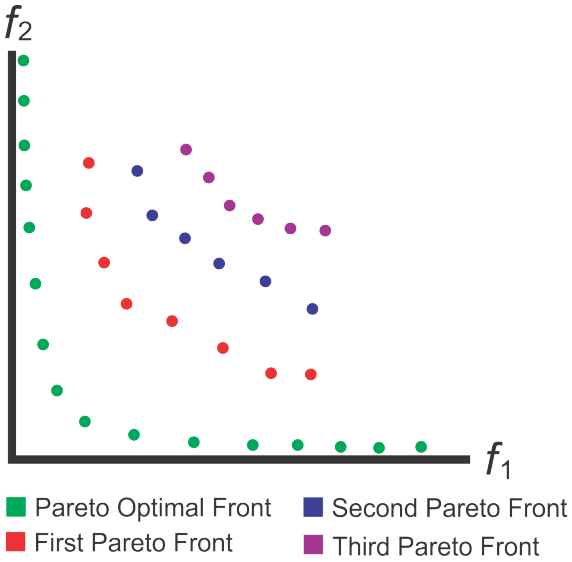

FIGURE 2.5: A graphical example of the different Pareto fronts in a population, in contrast with the Pareto Optimal Front.

The exact non-dominance sorting procedure is performed in Algorithm 4. In a Multi-Objective Genetic Algorithm (MOGA), this sorting process is used to rank the population for replacement, instead of using a traditional sorting with respect to a single objective function.

As in the ranking of individuals, measuring the quality of a MOEA is more complex in comparison to its single-objective counterparts. A number of performance indicators have been proposed (Yen and He 2014), from which Hypervolume is one of the most popular. The Hypervolume metric has become a very powerful indicator due to three important characteristics (Shang et al. 2020): (1) it is highly related to the Pareto front; all the Pareto optimal sets maximize the Hypervolume. (2) It evaluates the optimization quality at the same time as the spread of solutions. (3) Hypervolume requires one parameter only. Based on Shang et al. (2020), the mathematical definition of Hypervolume is presented next.

Consider a set of m-dimensional points Z and let $\mathbf{1}_Z$ be a function that returns a 1 when the argument is a point $\mathbf{z} \in Z$ and 0 otherwise. The Lebesgue measure of the set Z is computed by integrating $\int_{\mathbf{z}\in Z} \mathbf{1}_Z(\mathbf{z})d(\mathbf{z})$. Now, let $PF \in \mathbb{R}^m$ be the final Pareto front, with m-dimensional points. A reference point $\mathbf{r} \in \mathbb{R}^m$ is defined such that no point in PF is dominated by $\mathbf{r}$.The

Algorithm 4 Non-Dominance Sorting Algorithm

Require: Population *Pop* of N individuals, m objective functions.
Output: Sorted population F based on non-dominance criterion.

```
while |Pop| is not 0 do
  ind ← Pop[0]
  F_i ← [ind]
  for q in Pop but not in F_i do
    if |F_i| = 1 then
      p ← F_i[0]
      if q ≺ p then
        p ← q
        F_i ← [p]
      else if p ⊀ q and q ⊀ p then
        F_i ← F_i ∪ [q]
      end if
    else
      n ← 0
      for f in F_i do
        if q ≺ f then
          if q not in F_i then
            Replace f by q in F_i
          else
            Remove f from F_i
          end if
        else if f ⊀ q and q ⊀ f then
          Increase n by 1
        end if
      end for
      if n = |F_i| then
        F_i ← F_i ∪ [q]
      end if
    end if
  end for
  F ← F ∪ F_i
  Pop ← Pop − F_i
end while
```

Hypervolume is then the Lebesgue measure of the union of all the points b that dominate r and at the same time are dominated by the points in PF, as shown in Eq. (2.2):

$$HV(PF, \mathbf{r}) = \mathcal{L}\left(\bigcup_{a \in \text{PF}} \{\mathbf{b} | \mathbf{a} \prec \mathbf{b} \prec \mathbf{r}\} \right) \tag{2.2}$$

The Hypervolume represents a length, an area and a volume, when m is equal to 1, 2 and 3, respectively. For the case of two objective functions, the Hypervolume would result in the area between the Pareto front and a previously defined reference point, as can be seen in Fig. 2.6. In the case of having three objective functions, the Hypervolume would equal the volume of a set of prisms.

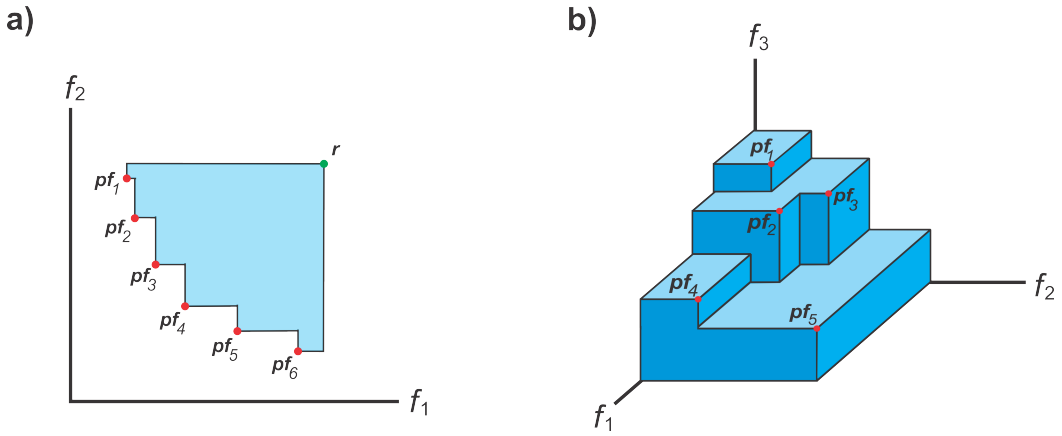

FIGURE 2.6: Visual meaning of the Hypervolume on **a)** a space of two objective functions and **b)** a space of three objective functions.

The definition of a reference point is crucial for the computation of the Hypervolume. A common approach is to normalize the values of each objective function f_i in *PF*. This can be done by using Eq. (2.3):

$$f_i^n = \frac{f_i^n - f_i^{min}}{f_i^{max} - f_i^{min}}, \quad i = 1, ..., m, \quad n = 1, ..., N \tag{2.3a}$$

where f_i^n is the n-th value of the i-th function in *PF*. f_i^{min} and f_i^{max} are the minimum and maximum values of f_i, respectively, in *PF*. After normalizing the Pareto Front, the functions' values will be in the range $[0, 1]$. A commonly used reference point is the *Nadir point*, that consists of the worst possible values of the functions that are inside *PF*. In the scenario of minimization of m normalized objectives, the Nadir point would be the $1 \times m$ vector $\mathbf{r} = (1, ..., 1)$.

A secondary evaluation metric solely consists in measuring diversity. As mentioned before, a diverse distribution in a Pareto Front is useful for decision-making purposes. The so-called *Spacing* (Yen and He 2014) metric is utilized to assess diversity using Eq. 2.4:

$$S = \sqrt{\frac{1}{t}\sum_{i=1}^{t}(d_i - \bar{d})^2} \tag{2.4}$$

where t is the total number of solutions in the objective functions' space, d_i is the Euclidean distance between the $i-$th solution and its closest solution, and $\bar{d}$ is the mean of all distances d_i. A smaller Spacing value corresponds to a better distributed front.

2.2 Convolutional Neural networks

Convolutional Neural Networks (CNN) are a type of Artificial Neural Networks (ANN) that are loosely inspired by the visual cortex (LeCun, Bottou, et al. 1998). One of the characteristics that boosted the popularity of CNNs is their ability of automatically extracting features from data without human intervention. Features from data are useful for learning, however, the manual extraction of these features is an engineering process that also requires domain expertise (LeCun, Bengio, and Hinton 2015).

It is likely that the term *Deep Learning* had been coined mainly due to the CNNs, as a notion of *depth* is associated to them. In fact, the feature extraction occurs through a (normally deep) composition of mathematical functions in the form of layers. It is said that the first layers in a CNN extract low-level features, such as edges in an image. Meanwhile, the medium and last layers are in charge of building mid- and higher-level features, such as shapes and texture, based on the previous ones (Oquab et al. 2014).

A considerable amount of the research on CNNs has dealt with designing and testing new architectures to solve different tasks. Nonetheless, the building blocks used in the construction of these networks have remained mostly unchanged to this day. Next, the most important of these components are explained in detail.

2.2.1 Convolutional Layers

Convolutional layers are the principal part of a CNN. These layers possess interesting properties that make them self-trained feature extractors. Convolutional layers owe their name to the operation they perform: the convolution ($*$). To introduce and explain this concept, the context of image processing is going to be used.

An image can be mathematically represented as a function I that receives pixel coordinates x and y (column and row, respectively) as inputs and maps to the pixel's intensity as output (Gonzalez and Woods 2017). In grayscale images, the intensity p is an scalar in the range $[0, 255]$, thus the mapping of the function is $I : \mathbb{R}^2 \rightarrow \mathbb{R}$. A grayscale image is also represented as a bi-dimensional array, analogous to a *surface*.

Color images such as RGB (acronym of *Red Green Blue*) images, in turn, map the pixels' coordinates x and y to an intensity vector $\mathbf{p}$, whose components (also called *channels*) represent the red, green and blue intensities of the pixel. The mapping of the function is $I : \mathbb{R}^2 \rightarrow \mathbb{R}^3$. A color image is therefore represented as a three-dimensional array (or 3D-Tensor), analogous to a *volume*.

In the convolution, a second function, called *filter* or *kernel*, is spatially translated through the image function. The filters are usually smaller than the image, and are filled with numerical values. When applied in a certain position of I, a filter applies the inner product between its values and the covered region of I. The sub-area or patch in the image that is covered by the filter is known as *receptive field*.

In the framework of ANNs, a filter can also be seen as an artificial neuron, and its values are called weights. A neuron is associated with a *bias* value that is summed at the end of the inner product. Both the weights and the bias are learned parameters during training, using the backpropagation algorithm (Lillicrap et al. 2020). Fig. 2.7 graphically describes the convolution between a filter of size 3×3 and an image.

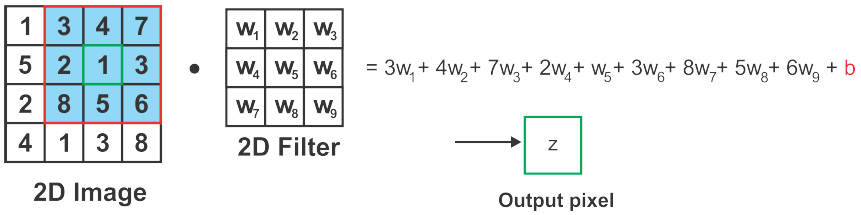

FIGURE 2.7: The convolution operation between a 2D image and a 2D filter of size 3×3. The output pixel will be placed in the same position as in the pixel framed in green in the image. The bias b is summed at the end of the inner product.

The mathematical definition of a bi-dimensional convolution, such as the one performed in image processing, is shown in Eq. (2.5):

$$(f * g)(x, y) = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} g(i, j) f(x - i, y - j) \tag{2.5}$$

where f and g are two bi-dimensional functions, with x and y as their input variables (pixels coordinates), and i and j are the indexing values that shift f through g. N and M are the number of rows and columns, respectively. If the input is an RGB image, the filter must also be a 3D tensor. Fig. 2.8 displays an example of how the convolution is applied.

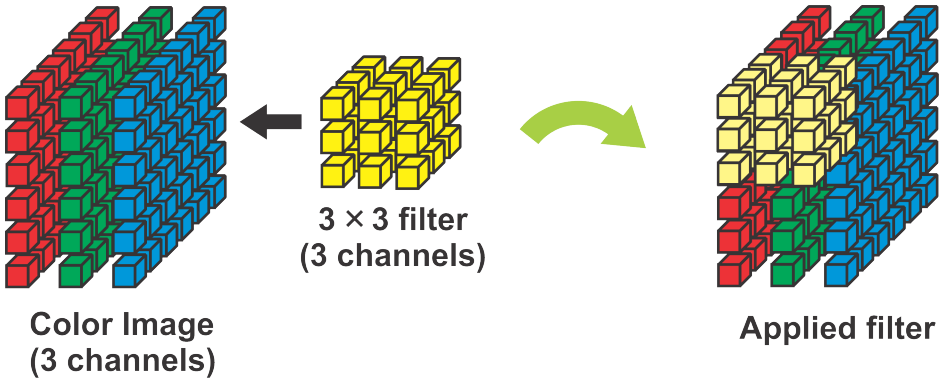

FIGURE 2.8: Convolution between a color RGB image and a 3D filter. Each cube represents a numerical value. The 3D filter is slid through the input tensor and the inner product is applied.

The convolutional layer consists of a collection of filters that are applied to an image. All the filters inside a layer have the same spatial resolution, i.e., size. Although the filters are

translated through the image, each position is processed by different weight values. This is due to the fact that each neuron is *in charge* of covering one section of the input image. This also causes a considerable growth in the number of trainable parameters when many filters are used in a layer. The weights of all the convolutional layers are optimized during training, so as to reduce an error metric.

The number of spaces that a filter is moved to the right and down is called *stride*. Larger stride values reduce the number of neurons per filter, as more regions of the image are ignored by the filter. Optionally, the borders of the image can be filled with "dummy" values so that the centers of the filters can be placed in the original border pixels. This procedure is called *padding*, and zero is a common used value (zero-padding).

It is worth mentioning that the convolutional layer's output size might not be the same as in the input. The output size decreases in proportion to the filter size, the stride size, and the number of padding borders (Fei-Fei, Krishna, and Xu 2020). If the filter and the image are squared (same height and width), then Eq. (2.6) is used to compute the output size:

$$O_{size} = \left\lceil \frac{W - F + 2P}{S} \right\rceil + 1 \tag{2.6}$$

where W is the input image size (given that height and width are equal), F is the filter size, P is the number of borders of padding added to the image, and S is the stride value. The symbols $\lceil \cdot \rceil$ denote the *ceil* function, which returns the closest upwards integer of a given real number. The number of output channels, called *feature maps*, is equal to the number of filters in the layer. For example, let a convolutional layer contain five filters of size 5×5, with stride of 2 and applying 1 border of zero-padding. Consider also an input RGB image (with three channels) of 256×256 pixels (the depth of the filters are adjusted to the depth of the input; 3 in this case). The output of the layer would consist of five feature maps with a spatial resolution of 128×128, which equals a 3D volume of $128 \times 128 \times 5$.

Convolutional layers are linear operations, since the inner product is used in each position. As in standard ANNs, a non-linear function is required to expand the representation power of

the extracted features. This function is also called *activation function*, and plays an important role in the performance of the networks (Ramachandran, Zoph, and Le 2017). The Rectifier Linear Unit (ReLU) (Hahnloser et al. 1998) is commonly used as it favors better optimization results. Eq. (2.7) presents the mathematical definition of the ReLU activation function:

$$ReLU(x) = max(0, x) \tag{2.7}$$

where x is a numerical input and max returns the largest of its inputs. The ReLU activation function is commonly placed after the convolutional layer to transform the extracted features on a non-linear basis. Fig. 2.9 displays the plot of this activation function.

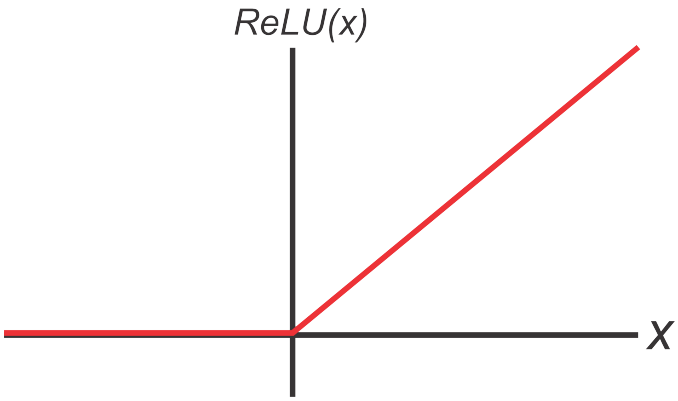

FIGURE 2.9: Plot of ReLU. The x axis represents the input and the y axis represents the output. The function takes the value of 0 when the input is negative, but has a constant positive slope of $45\,\mathrm{deg}$ for positive inputs.

2.2.2 Batch Normalization Layers

Batch normalization (Ioffe and Szegedy 2015) can significantly improve the training of the CNNs. As explained by Ioffe and Szegedy, the batch normalization works against the *Internal Covariate Shift* (ICS). This issue is present as activation values distribution changes in a network during training, and occurs as the weights keep changing while new inputs are passed forward through the architecture. When the ICS is high, the training process might take longer so as to compensate and continually adapt the weights to new distributions.

To better understand batch normalization, it is important to clarify that the inputs of CNNs, e.g., images, are introduced in groups of m inputs. A single of these groups is called *batch*. The error value of the network during each iteration is calculated based on a batch instead of individual images. Therefore, batch normalization adjusts the weights to the incoming batches.

Let a batch B of size m be composed of a $d-$dimensional input $x = (x^{(1)}, ..., x^{(d)})$. The mean and variance of the batch can be computed as μ_B and σ_B^2, respectively. For each component x_i in the batch, the normalization shown in Eq. (2.8) is used:

$$\hat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{2.8}$$

with ϵ being a small constant and is used to avoid divisions by zero. The normalized element $\hat{x_i}$ has now zero mean and a variance of one. In order to avoid constraining the representation power of the previous convolutional layer, another transformation is applied over each normalized value $\hat{x_i}$ as shown in Eq. (2.9):

$$y_i = \lambda \hat{x_i} + \beta \tag{2.9}$$

where λ and β are new learnable parameters associated to each original input x_i inside the batch B. The batch normalization is then mathematically written as the transformation $\mathbf{BN}_{\lambda,\beta} : x_i \rightarrow y_i$. As mentioned in (Santurkar et al. 2018), batch normalization can be applied in the form of a layer, augmenting the CNN architecture repertoire.

2.2.3 Pooling Layers

The pooling layers apply a sub-sampling operation to the input feature maps. Pooling plays an important role in reducing the spatial resolution of the feature maps, which decreases the number of required weights in subsequent convolutional layers. The purpose of pooling is to

merge features into a single one, based on their neighborhood semantics (LeCun, Bengio, and Hinton 2015), i.e., how meaningful features are with respect to surrounding features.

Pooling layers consist of a *pooling kernel*, which is similar to a convolutional filter, except that the former has no weights. As in the convolution, the pooling kernels are slid through the input with a certain stride value and applying padding if needed. The task of pooling is to only extract certain features based on a given criterion.

There are two popular pooling types. The first one is *max pooling*, in which the kernel's output is the maximum value of the covered patch in its current position. The second one is *average* pooling, that returns the average of the values seen by the kernel. Fig. 2.10 illustrates both pooling types.

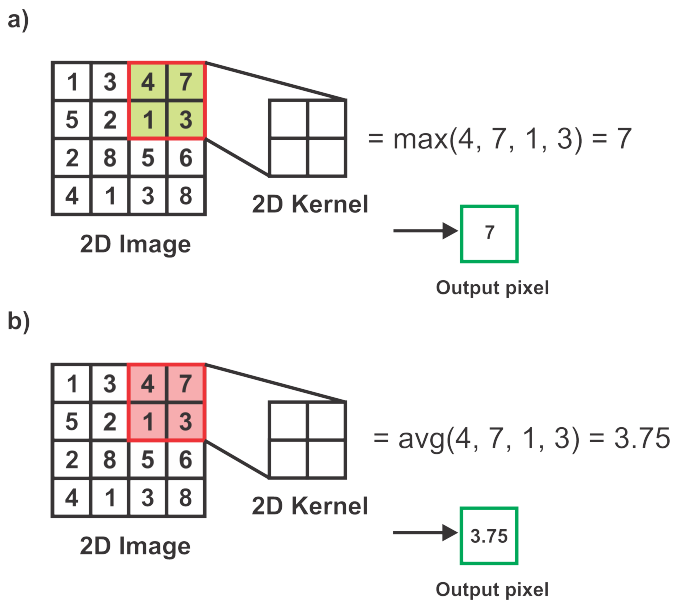

FIGURE 2.10: The pooling operation of a 2D pooling kernel of size 2×2 on a 2D image. **a)** Max pooling, that returns the maximum value of the patch covered by the kernel. **b)** Average pooling, where the returned value is the arithmetic average of the patch values.

As in convolutional layers, the spatial resolution of the outputs of pooling layers can be calculated based on the kernel size, the stride size and the number of padding layers. Given the fact that the input image has a height and width of the same size, and the pooling kernel is squared, Eq. (2.6) can be used.

2.2.4 Fully-Connected Layers

Fully-connected layers are ensembles of artificial neurons. Each neuron is a computational processing unit that is an extended version of Rosenblatt's perceptron (Aggarwal 2018). A neuron receives a series of d inputs $\mathbf{x} = [x_1, ..., x_d]$ through its d connections. Each connection is associated with a weight w_i that is multiplied by the corresponding input x_i. As a first step, the neuron processes the inputs by computing a weighted sum of them. In its vector form, this operation has the form of $z = \mathbf{w}^T\mathbf{x} + b$, with b as the associated bias value of the neuron. This new numerical output is then used as argument for a non-linear function, being ReLU a possible option. The detailed model of a neuron is shown in Fig. 2.11.

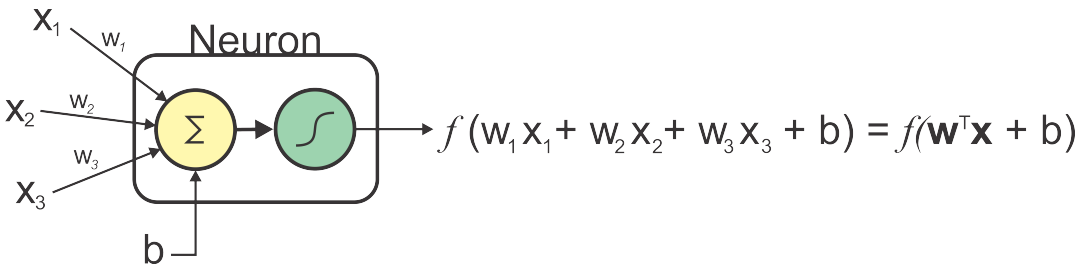

FIGURE 2.11: The model of an artificial neuron with three inputs. The yellow circle represents the weighted sum of inputs and the green circle represents the non-linear activation function f. The bias is represented by b.

A fully-connected layer is a set of neurons that are arranged in such a form that all the information from previous layers is received and processed by all the neurons of a current layer. The stacking of these layers produces a powerful composition of functions, and is called *neural network* or *multi-layer perceptron*. In Fig. 2.12, a simple neural network with three fully-connected layers is presented.

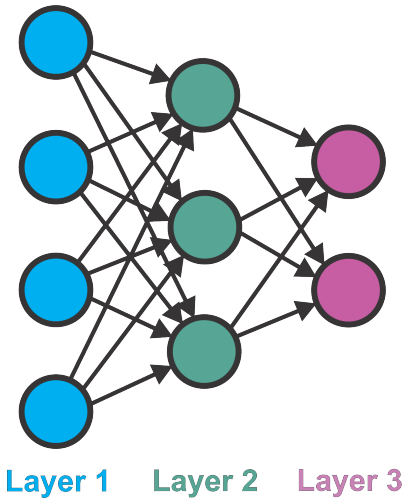

FIGURE 2.12: A simple model of a neural network composed of three fully-connected layers. Each circle represents a neuron, while the edges correspond to the weighted connections.

The origin of the term *fully-connected* comes from the connectivity pattern of these layers; all the neurons in a layer connect to all the neurons in the following layer.

Fully-connected layers are appended at the end of the feature-extraction section of the CNN (convolutional and pooling layers). In the context of this thesis, the fully-connected layers are used for classification purposes; assigning the correct label to an input image. All the feature maps at the end of the last convolutional/pooling layer need to be used as inputs for the first fully-connected layer. A *flatenning* process is applied, which consists in transforming a bi- or three-dimensional array into a one-dimensional array. Fig. 2.13 displays the flattening of two examples of feature maps.

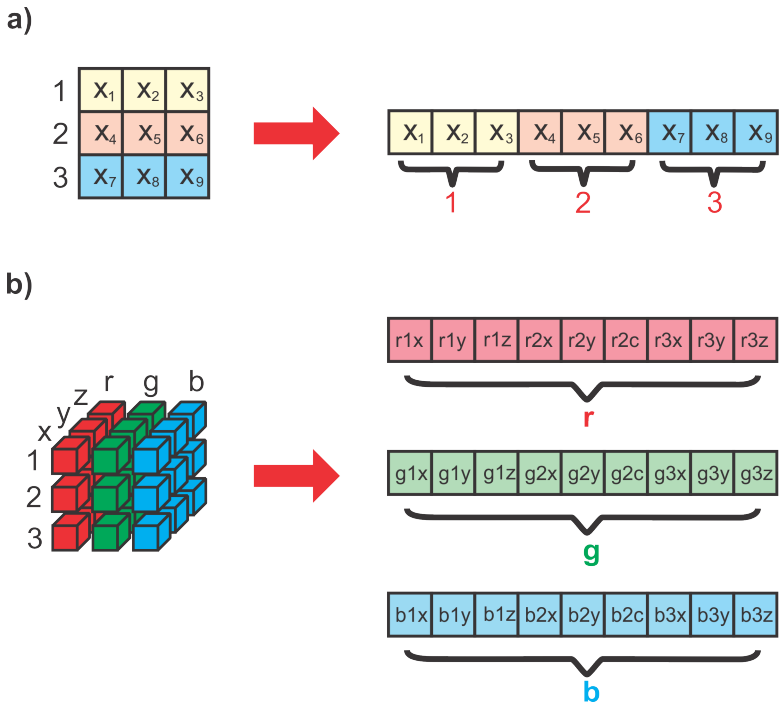

FIGURE 2.13: Two examples of flattening of feature maps. **a)** The flattening of a 2D feature map into a one-dimensional array. **b)** The flattening of a 3D feature map with three channels representing the red, green, and blue colors. As in the 2D example, the three vectors r, g, and b are concatenated into one single vector.

To perform classification, the last fully-connected layer should have as many neurons as the number of classes to predict. Each of these neurons performs a prediction for its corresponding class. Therefore, a probability estimation is to be computed.

Let $\overline{\mathbf{x}} = (x_1, ..., x_J)$ be the vector that contains the J outputs of the penultimate fully-connected layer, being x_j the output of the j-th neuron. The last fully connected layer is composed of K neurons in order to predict K classes. The k-th neuron receives J connections from the previous layers' neurons, each of which has an associated weight w_{jk}. All these weights are grouped in the vector $\mathbf{w}_k = (w_{1k}, ..., w_{Jk})^T$. The first step of the k-th neuron results in the computation of the weighted sum $\mathbf{w}_k^T \overline{\mathbf{x}}$. Each of the K neurons in the last fully-connected layer performs this calculations, then the *softmax* activation function can be applied (Aggarwal 2018).

The softmax function allows the k-th neuron to calculate the probability of assigning the class k given the incoming values $\overline{\mathbf{x}}$. This is expressed as the posterior probability $P(k|\overline{\mathbf{x}})$ and is computed using Eq. (2.10).

$$P(k|\overline{\mathbf{x}}) = \frac{exp(\mathbf{w}_k^T \overline{\mathbf{x}})}{\sum_{i=1}^{K} exp(\mathbf{w}_i^T \overline{\mathbf{x}})} \tag{2.10}$$

The sum of the posterior probabilities of the last layer sums to 1, and the class with the highest probability is selected.

To train the CNNs, the backpropagation algorithm is utilized (Lillicrap et al. 2020), then it is necessary to calculate a differentiable error based on the network's inference. The error is calculated using a *loss function*, and the negative log-likelihood loss function is the most suitable for multi-class classification (Aggarwal 2018). As a reminder, the input data is merged in batches of m images, thus the negative log-likelihood loss is calculated as in Eq. (2.11):

$$L = -\sum_{n=1}^{m} log[P_m(c|\overline{\mathbf{x}})] \tag{2.11}$$

where c corresponds to the real class of the corresponding image in the batch. The logarithm of the likelihood of choosing the actual class corresponds to the confidence on assigning the correct label successfully. The maximization of the logarithm function is achieved by minimizing its negative.

2.2.5 The Dense Architecture

In this thesis, a special emphasis is placed on the Dense Architecture. Residual Networks (ResNet) (He et al. 2016) and Dense Networks (DenseNet) (Huang et al. 2017) greatly improve the performance of CNNs for image classification. The main attribute towards this success was the introduction of *skip connections*.

A skip connection, or shortcut connection, occurs when the output feature maps of a convolutional/pooling layer are transferred not only to the immediate consecutive layer, but also to further, non-consecutive ones. The rationale behind skip connections relates to the gradient vanishing problem (Hanin 2018). When an ANN grows in depth, the gradients sent backwards by backpropagation start to *vanish*, causing that the first layers' weights do not get updated properly. By using skip connections, the gradients from one layer can be transferred not only to the previous consecutive layer, but also to other previous layers in order to update the weights. These breakthroughs allowed to design CNNs that are deeper than their earlier counterparts.

The DenseNet architecture, as its name suggests, is made using a Dense Architecture. The feature extraction part of the CNN is organized with several *DenseBlocks*. Each DenseBlock consists of a number of convolutional layers with 3×3 filters, applying zero-padding and a stride of 1. These layers are connected using a dense connectivity pattern: each layer connects to all the following layers. The convolutional layers' configuration ensures that the output feature maps in all the layers have always the same spatial resolution, hence they can be concatenated to further features maps. Fig. 2.14 illustrates the structure of a DenseBlock.

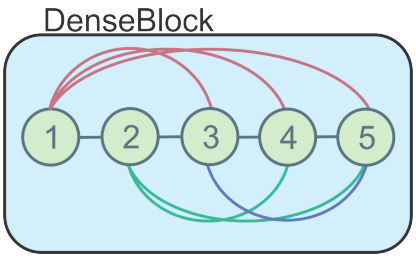

FIGURE 2.14: An example of a DenseBlock containing five convolutional layers. Each layer connects to all its following layers using skip connections.

In DenseNet, there is a hyperparameter called *growth rate*, which determines how many feature maps are received by the k-th layer. The growth rate is the number of filters in the convolutional layers inside the DenseBlock. The number of filters is equal to the number of output feature maps. If the growth rate is 3, for example, the k-th layer would receive the 3 feature maps from the immediate previous layer, as well as the 3 feature maps coming from

all the earlier layers. Therefore, the k-th layer receives $k(k-1)$ inputs. Fig. 2.15 shows how the feature maps are concatenated from the third layer onward.

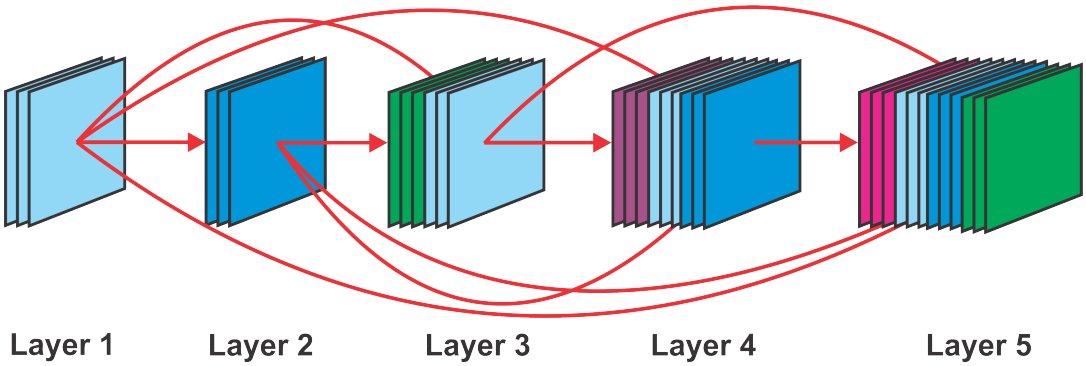

FIGURE 2.15: A detailed example of the feature maps concatenation in a DenseBlock with five convolutional layers. The third layer receives the outputs from the previous layer (first three squares) and the outputs from all the previous layers. This also applies for the following layers.

It is important to notice that when the growth rate and the number of convolutional layers are larger, the number of weights and bias also increase in the network. The authors suggest a growth rate below or equal to 12 (Huang et al. 2017). After a DenseBlock, a 3×3 convolutional operation with a number of filters of half the growth rate of the previous block is applied. Next, batch normalization, ReLU and pooling are used. It will be presented that the DenseBlock has an important role in Neuroevolution.

2.3 Neuroevolution

Neuroevolution is an application of EAs that nowadays has an important impact in one of the most active areas of research: Deep Learning (Baldominos, Saez, and Isasi 2019b; Elsken, Metzen, and Hutter 2019; Galván and Mooney 2020; Stanley, Clune, et al. 2019). Since 2002, Neuroevolution has been positioned as a very effective technique to optimize standard ANNs, however, it was not until 2017 when the CNNs started to increasingly attract the attention to be merged with Neuroevolution. Fig. 2.16 presents a general Neuroevolution framework based on a GA.

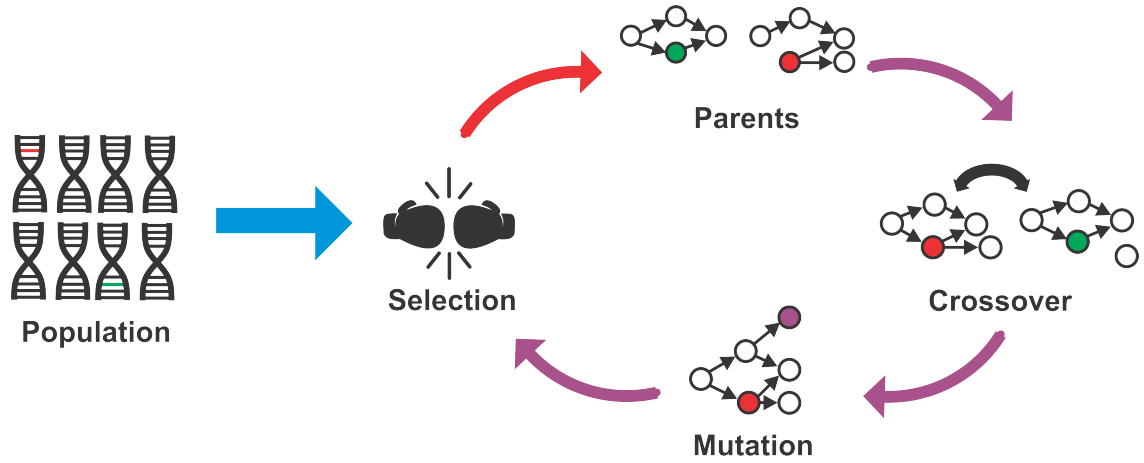

FIGURE 2.16: A general template of Neuroevolution based on the use of a GA. The red arrow indicates that a decoding process from genotype to phenotype might be required.

The primeval publication on Neuroevolution dates back to 1989, in which an EA evolved the weights of a feed-forward ANN (Montana and Davis 1989). Little more than a decade later, a breakthrough had been developed. Neuroevolution of Augmenting Topologies (NEAT) (Stanley and Miikkulainen 2002) introduced several important concepts still relevant to our days. The popularity of NEAT was due, in part, to its ability to automatically design neural networks and to increasingly complexify them as needed depending on the task.

Neuroevolution of CNNs has been prolific in solving the demanding task of designing their architectures. Unlike before, when both the architectures and the weights were evolved simultaneously, current trends in Neuroevolution deal with evolving the architectures and training them using backpropagation. The contributions in this field can be categorized as the following:

- **Neural Encodings and Evolutionary Operators.** This branch is the most popular, where the research focuses on designing new encodings for CNNs as well as evolutionary operators to manipulate them. This has allowed to exploit the wide variety of available EC algorithms.
- **Evaluation Methods.** As backpropagation is utilized, the evaluation of the individuals consists on training the networks with the specified data. This process can be

time-consuming, specially when datasets are big and the learning task is complex. Some papers intend to improve on this issue by proposing alternative evaluation approaches in order to reduce the computational cost of training.

- **Optimization Approaches.** Neuroevolution research has mainly focused on evolving CNNs to maximize their learning metric (e.g., accuracy in classification problems). In the mean time, some directions have also been explored in order to introduce multi-objective Neuroevolution, in which more than one fitness function is optimized.
- **Learning Tasks.** Most of the research on Neuroevolution of CNNs has been applied for the problem of image classification, with MNIST and CIFAR-10 as the most common benchmarks. However, some work has been done towards expanding the application scope of Neuroevolution, whether it is for different classification problems (e.g. different datasets) or to solve other problems such as image segmentation, object detection, text classification, among others.

This thesis concentrates mainly on *neural encodings* as a mean to reduce the computational cost of training the evolved CNNs. Accompanying this proposal, two different fitness mechanisms are also studied, which are using a linear aggregate objective function, and multi-objective optimization. Finally, this work explores the application of Neuroevolution for classification using a dataset different from benchmarks. As mentioned in Chapter 1, benchmark datasets usually include a considerable number of examples, which forces the CNN architectures to grow in size in order to fit the data. On the other hand, other datasets, such as medical imaging datasets, are not as large as benchmarks. Due to this issue, increasing the size of the CNNs would make them prone to overfitting.

As in standard EAs, neural encodings are computational representations for ANNs. The same ideas remain: the neural encodings are the first step in the design of any Neuroevolution algorithm, and they highly impact the complexity of the search space (Chen, Meng, et al. 2019).

Neural encodings of ANNs can be classified in two main categories: *Direct Encodings* and *Indirect Encodings* (Stanley 2004). These two types are analogous to the phenotype-level and

genotype-level encodings in EAs, respectively. A comprehensive literature review revealed that in the context of CNNs, the neural encodings can also be classified within these two groups. The proposed taxonomy of neural encodings, along with the *state-of-the-art*, is provided next.

2.3.1 Indirect Encodings

An indirect encoding is a genotype-level representation that requires a set of rules to be transformed into a phenotype that can be evaluated. An attractive aspect of these encodings is that they do not encode all the information of the networks explicitly, yet all the hyper-parameters of a CNN can be extracted by following the decoding rules. On the other hand, these encodings might suffer from a lack of flexibility when trying to represent more complex CNNs architectures.

2.3.1.1 Binary Encodings

Binary Encodings use binary arrays to represent CNNs topologies. One-dimensional binary arrays were some of the primeval representations in the field of EC, and encompass the benefit of being compatible with several well-known evolutionary operators, specially in GAs.

The decoding of binary encodings tend to be straightforward as presented in the specialized literature. However, these representations are normally fixed in length and, due to the nature of binary numbers, the search spaces are constrained. The reduced flexibility becomes evident as larger networks usually require larger arrays, which at the same time forces the population to grow. However, Binary Encodings excel at representing the connectivity between layers, and are also useful for the problem of pruning a CNN. Binary Encodings are categorized into binary strings and binary matrices.

Binary strings are one-dimensional arrays of bits that have been utilized to represent several hyperparameters of CNNs. Usually, binary strings are divided into sub-strings that are later transformed into real numbers. Baldominos et al. (2018a) utilized a fixed-length binary string of 69 bits. The string is divided between the feature extraction sub-string

(representing convolutional and pooling layers) and the classification sub-string (representing fully-connected layers). Each sub-string is also divided in small strings that correspond to gray codes that can be transformed into actual hyperparameters such as number of convolutional layers, number of filters, filter sizes, pooling type, pooling sizes, number of fully connected layers, number of neurons per layers, learning rate, among others.

Xie and Yuille (2017) introduced the Genetic CNN, in which a CNN is conceived as a sequence of stages, each containing a number of convolutional layers with the same number of filters and the same output sizes. The connectivity between layers is determined by a binary string. Jiang et al. (2020) expanded this encoding by utilizing a second string that determines the hyperparameters of the layers inside each stage. This latter encoding is shown in Fig. 2.17.

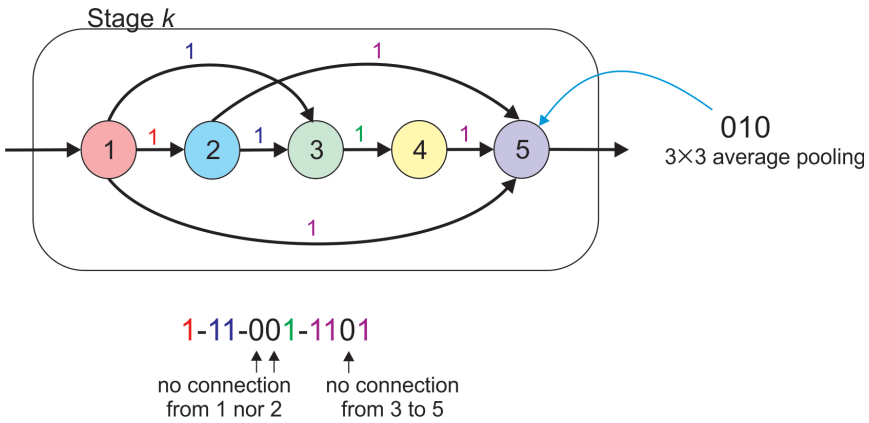

FIGURE 2.17: An example of the binary encoding for a single stage used in Jiang et al. 2020. Each node *i*, except the first one, has $i - 1$ bits specifying connections from the previous nodes. Also, each node is represented by a *3-bit* sub-string. In this example, *010* is related to a 3×3 average pooling.

Some other works have utilized binary strings for Neural Architecture Pruning (NAP). NAP is the *top-down* version of Neuroevolution; instead of building a CNN from scratch, an existing architecture is reduced to some extent until a compromise between performance and computational cost is achieved. If a binary string has a length equal to the number of filters in a CNN, then each bit can determine if a given filter is turned on (1) or off (0). This approach has been exploited in (Fernandes and Yen 2021; Jr. and Yen 2021; Junior and Yen 2019b; Wang, Xu, et al. 2018; Zhou, Gen, and Yi 2021; Zhou, Yen, and Yi 2020).

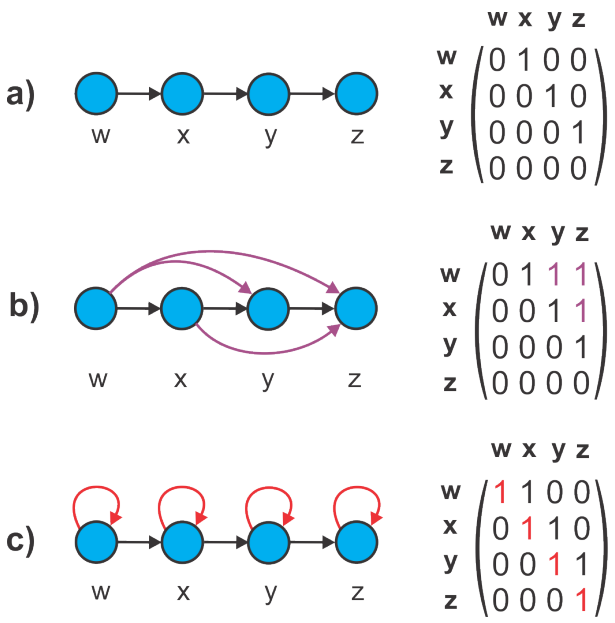

FIGURE 2.18: A binary matrix that describes the connectivity between four convolutional layers. Only the upper-right triangle is considered. **a)** Simple configuration of feed-forward connectivity. **b)** The introduction of skip connections as in DenseNet (Huang et al. 2017). **c)** The main diagonal is never considered, as it corresponds to recurrent connections, which are not allowed in standard CNNs.

Binary matrices are squared, higher-dimensional arrays of boolean values that are utilized as adjacency matrices for fixed-size graphs. These encodings are solely used to represent the connectivity. In an adjacency matrix A of $N \times N$, a layer i connects to the layer j if and only if $A_{ij} = 1$. Notice that $j > i$ as recurrent connections are forbidden in CNNs. Thus, the only important part is the upper-right triangle in A. Fig. 2.18 demonstrates different configurations of a binary matrix and the result in the connectivity of a CNN with four layers. Some representatives of this encoding method can be found in Lorenzo and Nalepa 2018; O'Neill, Xue, and Zhang 2019, 2020.

A summary of Binary Encodings is provided in Table 2.1, regarding the type of encoding, its length, and the hyperparameters that can be represented within the representation.

TABLE 2.1: Summary of Binary Encodings.

	Publications	Hyperparameters	Length
Binary String	Bivaeba et al. (2018), Chung et al. (2020), Yangyang et al. (2019), Baldominos et al. (2018a; 2019a), Chen et al. (2019), Liu et al. (2020), Ye et al. (2020), Zhou et al. (2021; 2020; 2021), Wang et al. (2018), Yuan et al. (2021), Fernandes and Yen (2021; 2019b)	Filters' size, No. of Filters, Pooling Type, Kernel Size, Stride, No. of Neurons, Activation Function, Batch Size, Learning rate, Optimizer	**Fixed**
	Wang et al. (2018a; 2018b), Xie and Yuille (2017), Akut et al. (2019), Ahmad et al. (2020), Ma et al. (2021) Lee et al. (2021), Jiang et al. (2020)	Filters' size, Kernel Size, Pooling Type, Connectivity	**Variable**
Binary Matrix	Lorenzo et al. (2018), O'Neill et al. (2019; 2020)	Connectivity	**Fixed**

2.3.1.2 Grammar Encodings

Grammar Encodings are some of the most scarce representations. These approaches are based on the theory of Grammatical Evolution (O'Neill and Ryan 2001). A CNN here is represented using a formal grammar in Backus-Naur notation. Once this grammar is characterized, it can be used to produce flexible networks with a wide range of changeable parameters. A grammar is defined as a 4-tuple $G = (N, T, S, P)$ where N is a non-empty set of non-terminal symbols, T is a non-empty set of terminal symbols, S is the starting symbol and P is a set of production rules in Backus-Naur form. A grammar allows the production of languages, i.e., all the possible sequences of terminal and non-terminal symbols that can be derived from the starting symbol. A single of these languages, thus, represents a CNN instance.

Baldominos et al. (2018b) utilized this encoding approach, arguing that it avoids redundancies found in their 69-bit strings. An example of a production rule in their encoding has the form: $< conv >::=< n\_kernels >< k\_size >< act\_fn >< pool >$, which means that a convolutional layer is defined by a number of kernels, their sizes, the activation function, and the pooling operation to perform. These elements are non-terminals that can also be generated by other rules, which creates a hierarchical structure.

DENSER (Assuncao, Lourenco, Machado, et al. 2018) is arguably the leading representative of Grammar Encodings. DENSER is composed of two levels: the genetic algorithm level, which represents the *macro-structure* of the CNN as a sequence of layers, and the Dynamic Structured Grammatical Evolution (DSGE) level, which describes the *micro-structure* of the genotype. The DSGE level contains the parameters to configure the GA level. Each element

TABLE 2.2: Summary of Grammar Encodings.

	Publications	Hyperparameters	Length
Grammar Encoding	Baldominos et al. (2018a; 2018b), Lima et al. (2019), Assunçao et al. (2019; 2018; 2019; 2020), Cetto et al. (2019)	Filters' size, No. of Filters, Pooling Type, Kernel Size, Stride, No. of Neurons, Activation Function, Regularization, Dropout, Optimizer, Padding, Batch Norm., Connectivity	Fixed

in the GA level is used as the initial non-terminal symbol for the DSGE grammar. The authors suggest that this representation is flexible enough to also represent other types of ANNs.

The summarized Grammar Encodings are presented in Table 2.2.

2.3.2 Direct Encodings

Direct Encodings have been widely used as shown in the specialized literature. They are phenotype-level representations as there is no rule to transform their parts into actual CNN hyperparameters; all the characteristics of the networks are explicitly defined in the encoding. The popular utilization of these encodings have also brought a large and customized evolutionary operators to handle these representations.

2.3.2.1 Block-chained Encodings

The Block-chained Encodings represent the CNNs using linear structures. This encoding is composed of a sequence of *blocks*, which are abstractions of CNNs' elements. Each block is computationally characterized by a series of hyperparameters that describe the internal elements, such as number of filters or number of neurons, depending on the type of layer. For this reason, any linear data structure, such as the linked list, is suitable for Block-chained Encodings.

Block-chained Encodings are sub-divided into three main categories, depending on the level of abstraction of the blocks; low-level, medium level or high-level blocks. More abstract blocks contain several standard CNN components arranged in a predefined way (Sun, Wang, et al. 2020). Lower abstraction blocks represent single layers only (Prellberg and Kramer 2018). It is possible that during the search, a low-level Block-chained Encoding finds structures similar

to those in medium- and high-level blocks. Nonetheless, the justification of using blocks of higher abstraction lies in the efficiency of some well-known architectures, such as ResNet (He et al. 2016) and DenseNet (Huang et al. 2017).

Low-level blocks are used to represent the simplest of the CNN components, such as convolutional, pooling, and fully-connected layers. Works such as in (Ahmed, Darwish, and El-Sherbiny 2019; Dong, Juan, et al. 2018; Keshari 2017; Young, Rose, Johnston, et al. 2017) use the lowest granularity possible, by using batch normalization, ReLU, and dropout as single blocks. In other approaches, such as in (Fielding and Zhang 2018; Prellberg and Kramer 2018; Rikhtegar, Pooyan, and Manzuri-Shalmani 2016), elements like batch normalization and ReLU are integrated inside convolutional blocks. Sun et al. (2019a) employed convolutional and pooling blocks only, disregarding fully-connected layers. Similar methods are found in (Jalali et al. 2019; Operiano, Iba, and Pora 2020; Ren et al. 2019). Sun et al. (2019c) proposed what can be considered as the standardize version of a low-level Block-chained Encoding, which also includes fully-connected blocks. Fig. 2.19-a) shows an example of this encoding.

Medium-level blocks take abstraction one step further. In this type of encoding, a block contains more than one operation or layer of the same type. This type of block helps to achieve a higher modularity in the final architecture. Moreover, the utilization of more abstract blocks that are linked to well-established concepts in Deep Learning, such as skip connections. On the other hand, there is a trade-off between abstraction and expressiveness. For instance, the hyperparameters of the elements contained by medium-level blocks are usually fixed.

Some approaches combine several layers in a feed-forward fashion into a single block. Sapra et al. (Sapra and Pimentel 2020) defined a chain of medium-level blocks formed by a fixed number of convolutional, pooling or fully-connected layers. Similar methods were used in (Hajewski, Oliveira, and Xing 2020; Loni, Sinaei, et al. 2020; Loni, Zoljodi, et al. 2019; Lu, Deb, et al. 2019). Other authors included skip connections to improve the training phase of the CNNs, particularly the DenseBlock and the ResBlock. Both types of blocks were combined in (Hassanzadeh, Essam, and Sarker 2020a; Sun, Wang, et al. 2020; Sun, Xue, et al. 2019b). Fig. 2.19-b) shows an example of medium-level encoding based on DenseBlocks. Finally, other medium-level blocks contain fixed-size, fixed-structure directed acyclic graphs

(Lu, Whalen, Dhebar, et al. 2020; Tan et al. 2020; Zhou, Zhou, et al. 2020). Real et al. (2019) introduced the so-called *normal cells* and *reduction cells*. The former contains a graph of layers that generates an output whose size is equal to the input. The latter, on the other hand, reduces the output size, usually by half, with respect to the input.

High-level blocks use recently discovered structures that have advanced the performance of human-designed CNNs. Hassanzadeh et al. (2020c) evolved a chain of four possible attention blocks for image segmentation. Each of them contains a configurable sub-block in terms of number of layers, number of filters and their sizes, among other characteristics. In another encoding, a block is made of two parallel chains that merge together and split afterwards (Zhang, Wang, et al. 2020). This representation is shown in Fig. 2.19-c). Other high-level Block-chained Encodings were proposed in (Gottapu and Dagli 2020; Song et al. 2020; Xu et al. 2020; Yao et al. 2020).

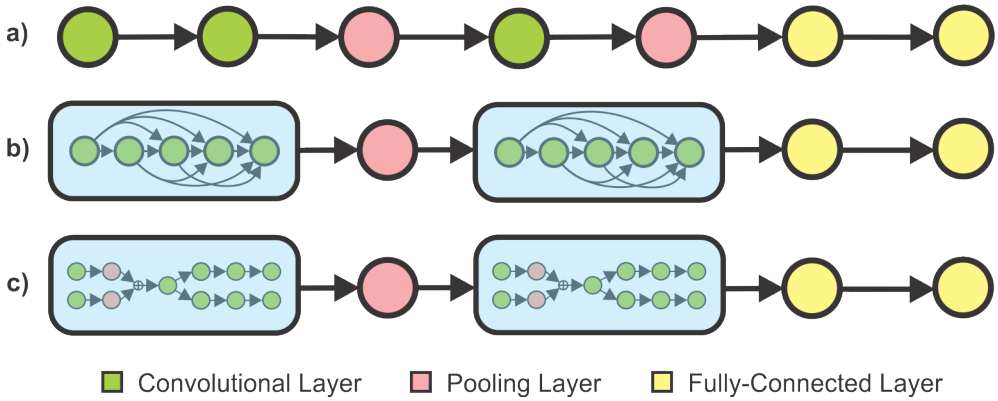

FIGURE 2.19: Different abstraction levels in Block-chained Encodings. **a)** Low-level blocks representing simple layers (Sun, Xue, et al. 2019c). **b)** Medium-level blocks represent DenseBlocks (Wang, Sun, et al. 2019a). **c)** High-level blocks represents more complex arrangements of layers, similar to those of popular hand-made CNNs (Zhang, Wang, et al. 2020).

Block-chained Encodings are organized into the three main categories and presented in Table 2.3.

TABLE 2.3: Summary of Block-chained Encodings.

	Publications	Hyperparameters	Length
Low-level	Fielding & Zhang (2018), Ahmed et al. (2019), Awad et al. (2020), Jalali et al. (2019), Chu et al. (2020), Wang et al. (2021), Tang et al. (2020), Litzinger et al. (2018), Hassanzadeh et al. (2020b;2021)	No. of Filters, Filters' size, Pooling Type, Kernel Size, Stride, No. of Neurons, Activation Function, Learning Rate, Optimizer, Batch Size, Connectivity	Fixed
	Prellberg & Kramer (2018), Young et al. (2017), Dahou et al. (2019), Keshari (2017), Kwasigroch et al. (2020), Johner et al. (2019), Fujino et al. (2019), Loni et al. (2020), Badan et al. (2019), Rikhtegar et al. (2016), Sun et al. (2019a; 2019c), Ren et al. (2019), Operiano et al. (2020) Young et al. (2015), Zhang et al. (2018), Mitschke et al. (2018), Dahal et al. (2020) Fernandes & Yen (2019a), Hadjiivanov & Blair (2019), Strumberger et al. (2019), Kotyan et al. (2020), Zhao et al. (2020), Dong et al. (2020), Martin et al. (2017; 2018; 2020), Vidnerova et al. (2020; 2020), Bochinski et al. (2017), Dong et al. (2018)		Variable
Medium-level	Sapra et al. (2020), Qu et al. (2020), Lu et al. (2019; 2020; 2020), Hajewski et al. (2020), Chen et al. (2019), Tan et al. (2020) Rajagopal et al. (2020), Hasanzadeh et al. (2020a), Prandini et al. (2019)	No. of Layers, No. of Filters, Pooling Type, Activation Function, Optimizer, Connectivity	Fixed
	Loni et al. (2020; 2019), Wang et al. (2019,2019a), Chen et al. (2021), Liu et al. (2018), Zhu et al (2019), Real et al. (2019), Zhou et al. (2020), Sun et al. (2020;2019b), Fernandes & Yen (2021), Ye et al. (2020)		Variable
High-level	Yao et al. (2020), Hassanzadeh et al. (2020c), Xu et al. (2020), Gottapu et al. (2020), Song et al. (2020), Zhang et al. (2020)	Filters' sizes, Connectivity	Fixed

2.3.2.2 Graph-based Encodings

Graph-based Encodings are the non-linear variants of block-chained encodings. As in the latter, graph-based encodings exploit the modularity of the CNNs, by abstracting the layers' characteristics inside units similar to blocks, connected with other units in a network called graph. Graph-based Encodings include the graph encodings and tree encodings.

A graph is a 2-tuple $G = (Q, E)$, where Q is the non-empty set of nodes and E is the non-empty set of edges connecting the nodes. A graph is directed if the edges point to a specific direction. A directed graph is acyclic if it does not contain any loop. When graphs are used to represent CNNs, highly flexible architectures can be achieved. Nonetheless, this flexibility comes at the cost of complexity due to the following: (1) the extraction of hyperparameters from a graph needs to be done carefully, in order to preserve the real structure of the CNN,

(2) the evolutionary operators must be *architecture-aware* by, for example, avoiding the generation of cycles in graphs, as they would not represent reliable CNN architectures.

A standard **graph encoding** is shown in Fig. 2.20. One example of these encodings is the one utilized by Real et al. (2017), however they used edges as operations, e.g., convolutional or pooling, whilst nodes are data holders, containing and concatenating feature maps coming from the edges and applying batch normalization and ReLU.

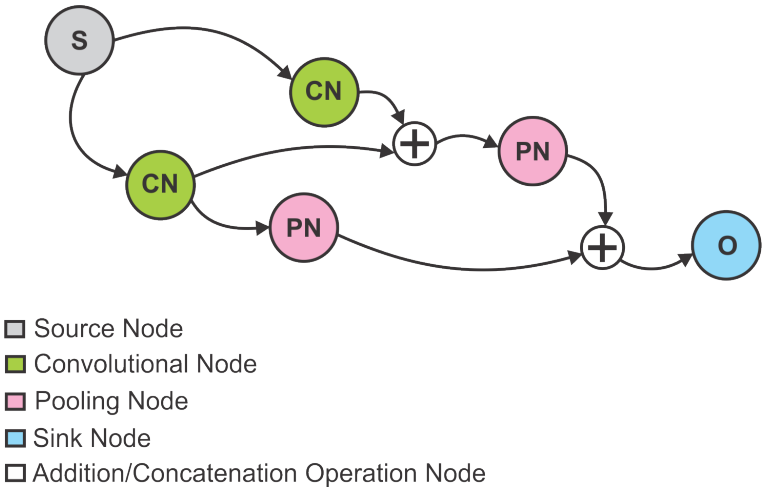

FIGURE 2.20: A graph-based encoding, in which nodes can represent convolutional and pooling layers, as well as addition/concatenation of feature maps.

The second type of Graph-based Encoding is the **tree encoding**, as can be seen in Fig. 2.21. Neither directed nor undirected cycles are present in trees. Furthermore, all the nodes that do not receive any edges are called *leaf nodes*, while the only node that receives edges but does not transfer any edge is called *root node*. One main advantage of tree encodings is that they can be utilized with Genetic Programming, allowing to find powerful compositions of convolutional operators (Bi, Xue, and Zhang 2019; Evans et al. 2018; Homburg et al. 2019; Irwin-Harris et al. 2019; McGhie, Xue, and Zhang 2020).

TABLE 2.4: Summary of Graph-based Encodings.

	Publications	Hyperparameters	Length
Graph	Desell (2017), Zhang et al. (2018), Barnes et al. (2020), Real et al. (2017), Zheng et al. (2020), Su et al. (2020), Chen et al. (2019), Byla et al. (2019), Witsuba et al. (2018), Maziarz et al. (2019), Saltori et al. (2019),Yu et al. 2020, Miikkulainen et al. (2019), Liang et al. (2019), Fernando et al. (2016), Verbancsics et al. (2013) Suganuma et al. (2020; 2017), Kobayashi et al. (2020), Yuan et al. (2020)	No. of Layers, No. of Filters, Filters' size, pooling type, kernel size, Batch Normalization, Connectivity	Variable
Tree	Homburg et al. (2019), Harris et al. (2019), McGhie et al. (2020), Bi et al. (2019), Evans et al. 2018		

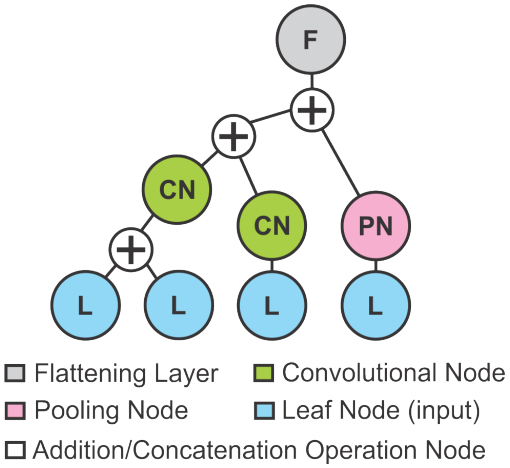

FIGURE 2.21: A tree encoding, which is used in Genetic Programming.

In Table. 2.4, the Graph-based Encodings are summarized and organized into the two available categories.

2.4 Hybrid Encodings

2.4.1 Hybrid Encodings

Hybrid Encodings have raised by combining ideas from the other families of encodings. These hybrid representations are useful to distribute the different motifs of a CNN between different structures. Also, these *out of the ordinary* encodings help tackling the limitations

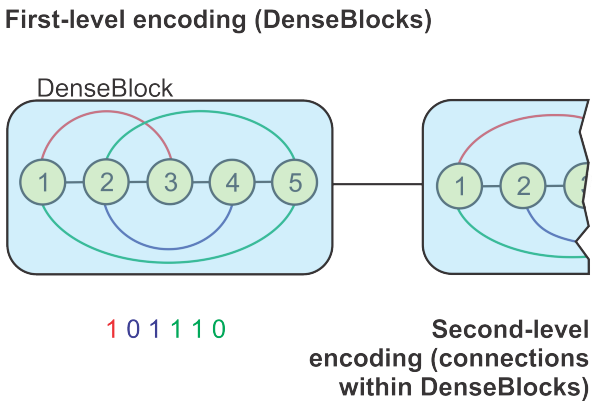

FIGURE 2.22: An example of the Wang encoding. The first-level encoding consists of a block-chained sequence of DenseBlocks, whilst the second-level encoding is a set of binary strings that determine the connectivity of each block.

that the other representations have. For these reasons, the research towards hybrid encodings is highly encouraged. In this category, other encodings based solely on hyperparameters and vectors of real numbers are considered as well.

In this thesis, the hybrid encoding here called as *Wang encoding* is studied in detail. Wang et al. (2019b) combined the Block-chained Encoding (first-level encoding) and the Binary Encoding (second-level encoding). The Block-chained Encoding includes handling and decoding facilities, making the process of building a CNN straightforward. However, these encodings are less flexible on their own. The binary encoding is used to determine the connectivity patterns of the layers. Here, a binary string provides for more flexibility to medium-level blocks, while maintaining the same level of abstraction as in low-level blocks.

In the Wang encoding, a block abstracts several layers in the form of a DenseBlock; each block is described by its number of convolutional layers and the growth rate. However, the dense connectivity pattern inside each DenseBlock is determined by a binary string. Starting from the third layer, a 1 corresponds to a skip connection, whereas a 0 turns off that skip connection. This happens because the first layer receives the input only, and the second layer can only receive the output from the first layer. This encoding allows the EA to independently search for the characteristics of the inner layers as well as their topology. An instance of this representation appears in Fig. 2.22.

TABLE 2.5: Summary of Hybrid Encodings.

	Publications	Hyperparameters	Length
Hybrid	Lu et al. (2019), Zhang et al. (2020), Broni et al. (2020), Wang et al. (2019b), Yang et al. (2020)	No. of Filters, Filters' Size, Connectivity	Variable
	van Wyk & Bosman (2019), Kang et al. (2020), Yotchon et al. (2020), Hu et al. (2020)		Fixed
Hyperparameters	Liu et al. (2019), Oh et al. (2021), Mostafa et al. (2020), Baldeon et al. (2020, Kobayashi et al. (2020), Javaherpi et al. (2020), Wang et al. (2021)	No. of Filters, Filters' Sizes, Pooling Type, Kernel Size, Learning Rate, Optimizer, Activation Function, Connectivity	Fixed

Other Hybrid Encodings combine other representation schemes, such as Block-chained and Graph-based Encodings (Lu, Whalen, Boddeti, et al. 2019; Tan et al. 2020; Zhang, Jin, et al. 2020). Table 2.5 summarizes this family of encodings.

2.5 COVID-19 in Chest X-ray Images

The outbreak of the new SARS-CoV-2 originally found in China at the end of 2019 has evolved into a worldwide public health emergency. As of August 2020, the World Health Organization (WHO) announced more than 12 million positive cases diagnosed solely in the Americas (World Health Organization Last accessed August 22, 2020). Unfortunately, more than 400 thousand deceases have been registered as of the same date.

The testing process of COVID-19 is mainly based on the Reverse-Transcription Polymerase Chain Reaction (RT-PCR) (Ai et al. 2020). As discussed in (Peto 2020), a generalized testing around the population is of vital importance as a measure to eradicate this disease. However, the standard testing method still needs to overcome obstacles. For instance, inadequate and insufficient tests have been reported in abounding scenarios (Beeching, Fletcher, and Beadsworth 2020). On the other hand, with a sensitivity between 30%–60%, a considerable number of COVID-19 carriers could be wrongly diagnosed (Ai et al. 2020).

In response to this complicated situation, the analysis of CXR images has been explored as a potentially faster and effective alternative testing approach. The lung damage caused by COVID-19 is visible in CXR images, even when a negative result is obtained from a RT-PCR test (Kanne et al. 2020), which greatly improves accuracy. Furthermore, computational tools

can also be used to perform this alternative testing at a larger scale. Fig. 2.23 shows an example of three X-ray images from a COVID-19 patient, a viral/bacterial pneumonia patient, and a healthy patient.

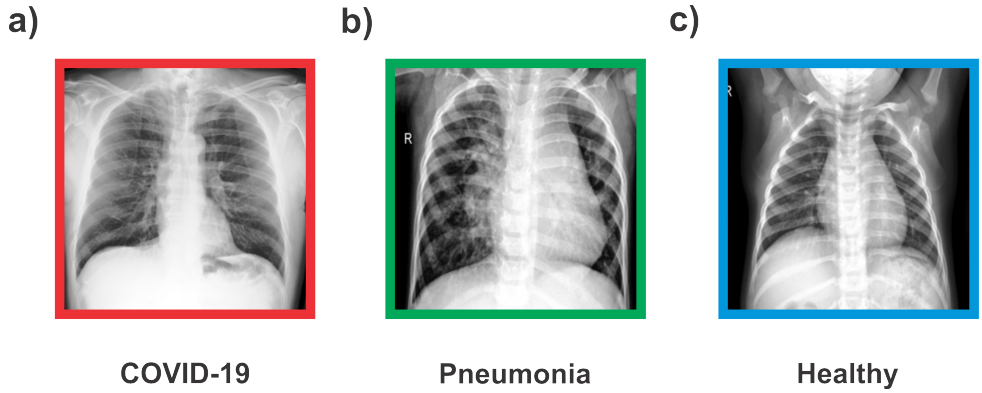

FIGURE 2.23: Chest X-ray images of three different patients: **a)** A patient with COVID-19, **b)** a patient with viral/bacterial pneumonia, and **c)** a healthy patient. Images obtained from (Cohen et al. 2020; Mooney 2017; Wang, Wong, et al. 2020)

CNNs have demonstrated to be competent at automating the analysis of CXR imagery (Baltruschat et al. 2019). During 2020, several hand-crafted architectures, such as DarkCovidNet (Ozturk et al. 2020), Xception + ResNetV50 (Rahimzadeh and Attar 2020), CNN + LSTM (Islam, Islam, and Asraf 2020) have been designed or re-utilized with the aim of solving this problem. Several of these works, such as (**Oh2020**; Altan and Karasu 2020; Civit-Masot et al. 2020), rely on the well-known architectures that have been tested in benchmark problems.

Multi-class image classification consists in correctly assigning a label to an image from a pool of C different classes. For each class c, the *true positives* (TP) are the images that are correctly classified, whilst *true negatives* (TN) are those images that do not belong to the class c and that are correctly classified as such. On the other hand, *false positives* (FP) are images from other classes that are wrongly classified into the class c, and *false negatives* (FN) are the images that belong to class c but are misplaced in other classes.

The aforementioned metrics are used to compute some classification scores that provide for valuable insights on the performance of a classification algorithm. Accuracy is the rate

of correctly classified instances (TP and TF) against the total of instances, as shown in Eq. (2.12).

$$\text{accuracy} = \frac{TP + TF}{TP + TF + FP + FN} \tag{2.12}$$

Accuracy is an overall score. Sensitivity and specificity, however, are calculated per class. The Specificity is the rate of images that are correctly classified into class c (TP) with respect to the total number of images that are placed into the same class (TP and FP), as in Eq. (2.13).

$$\text{specificity} = \frac{TP}{TP + FP} \tag{2.13}$$

Sensitivity (or recall) is the rate of images that are correctly classified into class c (TP) with respect to all the images that originally belonged to that class (TP and FN). Eq. (2.14) shows this computation.

$$\text{sensitivity} = \frac{TP}{TP + FN} \tag{2.14}$$

Table 2.6 shows the characteristics of the *state-of-the-art* on CNNs-based COVID-19 classification. The literature review focused on papers that solved the multi-class classification of CXR images with three classes; COVID-19, pneumonia, and healthy. Each reviewed work indicates the number of images per class, the name of the CNN architecture that was employed, its number of trainable parameters, and the achieved classification scores, which are sensitivity, specificity, and accuracy. Specificity and sensitivity are averaged for the three classes.

2.6 Chapter Summary

In this chapter, the theoretical foundations of this thesis were presented. Evolutionary Algorithms were discussed as powerful bio-inspired search and optimization methods, paying

TABLE 2.6: *State-of-the-art* in COVID-19 CXR imagery multi-class classification based on CNNs. The performance of each architecture is measured by its number of parameter (#Params), sensitivity, specificity, and accuracy.

Authors	Dataset	CNN	#Params	Sensitivity	Specificity	Accuracy
Ozturk et al. (2020)	127 (COVID-19), 500 (pneumonia), 500 (healthy)	DarkCovidNet	1.164	85.35	89.96	87.02
Ucar et al. (2020)	76 (COVID-19), 4290 (pneumonia), 1583 (healthy)	Bayes SqueezeNet	1.263	98.25	99.13	98.26
Rahimzadeh et al. (2020)	180 (COVID-19), 6054 (pneumonia), 8851 (healthy)	Xception + ResNetV50	45.855	87.31	93.99	91.40
Oh et al. (2020)	180 (COVID-19), 6012 (pneumonia), 305 (healthy)	ResNet18	11	85.9	96.4	88.9
Altan et al. (2020)	219 (COVID-19), 1345 (pneumonia), 1341 (healthy)	EfficientNet-B0	5.3	93.61	96.05	95.24
Civit-Masot et al. (2020)	132 (COVID-19), 132 (pneumonia), 132 (healthy)	VGG-16	138	100	91.41	82.81
Dat et al. 2020	127 (COVID-19), 500 (pneumonia), 500 (healthy)	Xception	22.855	97.09	97.29	97.41
Togaçar et al. (2020)	295 (COVID-19), 98 (pneumonia), 65 (healthy)	MobileNet V2 and Squeeze Net	4.635	99.32	99.37	99.34
Islam et al. (2020)	1525 (COVID-19), 1525 (pneumonia), 1525 (healthy)	CNN + LSTM	14.174	99.1	99.6	99.2
Brunese et al. (2020)	250 (COVID-19), 2753 (other), 3520 (healthy)	VGG-16	138	96	98	98

special attention to the Genetic Algorithm. The building blocks of Convolutional Neural Networks have been explained in order to build the notions of Deep Learning. Finally, the concepts related to Neuroevolution were introduced, with an important emphasis on the genetic encodings to represent Convolutional Neural Networks.

This chapter also presented a comprehensive review on the *state-of-the-art* on: (1) genetic encodings for convolutional neural networks, and (2) classification of COVID-19 patients through chest X-ray images.

As an output from this chapter, an expanded version of the literature review on encodings of convolutional neural networks for Neuroevolution has been submitted and accepted to the *IEEE Transactions on Evolutionary Computation*.

CHAPTER 3

Deep Genetic Algorithm

In this chapter, the main proposal of Neuroevolution of CNNs for CXR images is presented under the name of Deep Genetic Algorithm (DeepGA). This chapter comprises (1) the image dataset collection and preprocessing, (2) the algorithm design, and (3) the experimental methodology that is followed to acquire, evaluate, and analyze the empirical results.

3.1 Image Collection and Preprocessing

A CXR image dataset has been built by collecting images from three different classes: (a) COVID-19 patients (Cohen et al. 2020; Tabik et al. 2020; Wang, Wong, et al. 2020), (b) viral/bacterial pneumonia patients (Cohen et al. 2020; Mooney 2017; Wang, Wong, et al. 2020), and (c) healthy patients (Mooney 2017). The dataset is balanced, meaning that the number of images in each class is equal. A total of 2754 images has been acquired, with 918 images per class.

To improve the classification, a series of preprocessing steps were applied to the images, based on the proposal of Oh et al. (2020), who also utilized COVID-19 CXR images for computer vision purposes. The preprocessing consists of the following:

(1) Transforming the images to grayscale (only one channel).
(2) Casting datatype to *float32*.
(3) Equalizing histograms, which consists in obtaining a uniform distribution in the pixels' values (Gonzalez and Woods 2017). In a grayscale image, there are L

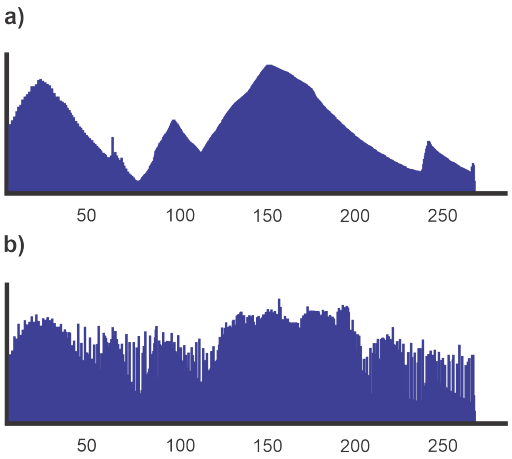

FIGURE 3.1: The distribution of pixels' values in a grayscale image **a)** before Histogram Equalization and **b)** after Histogram Equalization. After this process, a more uniform distribution in grayscale intensities is achieved.

intensity levels, usually ranging from 0 (black) to 255 (white), being $L = 256$. The probability of obtaining a pixel with the value r_k is calculated with Eq. (3.1)-a):

$$P(r_k) = \frac{n_k}{n}, \quad k = 0, 1, ..., L-1 \tag{3.1a}$$

where n_k is the number of pixels with the intensity value k in grayscale, and n_p is the total number of pixels in the image. Histogram equalization maps the pixel values r_k to s_k, and is performed using Eq. (3.2)-a):

$$s_k = \sum_{j=0}^{k} P(r_j), \quad k = 0, 1, ..., L-1 \tag{3.2a}$$

This procedure results in an increased contrast in the final image. Fig. 3.1 shows an example of the distribution of pixels' values before and after Histogram Equalization.

(4) Applying the gamma correction, which linearly transforms the pixels s_{in} values to s_{out} by using Eq. (3.3):

$$s_{out} = \gamma s_{in} \tag{3.3}$$

where $\gamma = 0.5$ for this application.

(5) Resizing the images to 256×256 pixels.

(6) Normalizing the pixels' values to the range $[0, 1]$.

After the aforementioned steps, the images were ready to be used for classification during Neuroevolution.

3.2 DeepGA

The proposed Neuroevolution algorithm is called DeepGA, as it is based on the foundations of the standard GA. This framework comprises three main features: (1) a hybrid encoding, (2) evolutionary operators to handle the hybrid encoding, and (3) a linear aggregating fitness function to evaluate the individuals based on classification accuracy and their number of parameters.

Furthermore, this chapter details the methodology to evaluate the performance of DeepGA at classifying chest X-ray images, and to compare the proposed neural representation against the competitor encoding (Wang, Sun, et al. 2019b). The experimental results are presented and discussed.

3.2.1 Compact Hybrid Encoding

As discussed in Chapter 2, Wang et al. (2019b) utilized a hybrid encoding based on DenseBlocks and binary strings (named *Wang encoding*). Each DenseBlock is characterized by its number of convolutional layers and its growth rate. A binary string determines the dense connectivity pattern of the layers inside a block (see Fig. 2.22). DeepGA is based on this encoding. However, the proposed approach goes one step backwards in the modularity of the representation.

An individual with the Wang encoding might consist of several DenseBlocks, which lead to increase the number of parameters as more blocks are added. In this thesis, a different hybrid

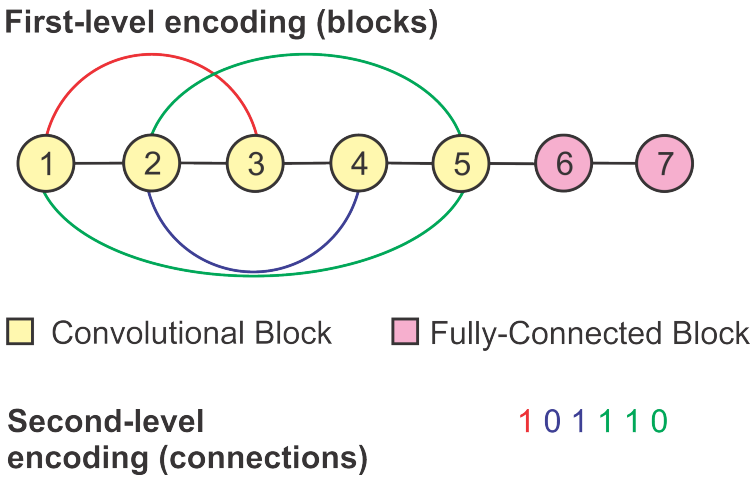

FIGURE 3.2: An example of the proposed encoding. The blocks represent simple convolutional operations instead of a set of convolutional layers. A single binary string is required to define the dense-like connectivity patterns between convolutional blocks.

encoding with simpler, less modular blocks is designed. The first-level encoding considers two different blocks; convolutional blocks and fully-connected blocks. Instead of utilizing DenseBlocks, which have several convolutional layers, a convolutional block is defined only by one convolutional layer and one (optional) pooling operation. The second-level encoding is a binary string that defines the connectivity between convolutional blocks. Fig. 3.2 displays an example of this new representation.

Convolutional blocks abstract the characteristics of a convolutional layer; number of filters and filters' size. The convolution utilized stride of 1 and zero-padding of 1. Additionally, these blocks can use max pooling, average pooling, or no pooling at all, depending on the individual. Hence, the pooling is defined by the pooling type and the kernel size. ReLU and batch normalization are always applied immediately after the convolution. Table 3.1 presents the different values that each hyperparameter can have during the evolution.

Unlike the Wang encoding, where each DenseBlock has its own binary string, this new encoding requires only one binary string per individual. From the third layer onward, each bit corresponds to a received connection from a previous, non-consecutive layer. For example, the fourth convolutional block in Fig. 3.2 may receive inputs from the first and second blocks,

TABLE 3.1: Evolvable hyperparameters in the proposed encoding. Yellow rows correspond to convolutional blocks, while the pink row corresponds to the fully-connected blocks.

	Values
No. of Filters	{2, 4, 8, 16, 32}
Filter size	{2, 3, 4, 5, 6, 7, 8}
Pooling Type	{Max, Avg}
Pooling Size	{2, 3, 4, 5}
No. of Neurons	{4, 8, 16, 32, 64, 128}

thus two bits are required. The connection from the third block (the immediate previous block) is always granted, therefore a bit is not required.

It is important to notice that in this encoding, the output size of each block is not equal to the input size. This issue makes the adjustment of the spatial resolutions of incoming feature maps from skip connections necessary to meet the dimensional requirements of the current block's input. Let the block n receive an input x from the immediate previous block. Given that x_k is the incoming input from the block k through a skip connection, there are three possible scenarios, as well as three actions to take:

- The size of the current input x is **smaller** than the size of the incoming input x_k. Max pooling is applied to reduce the size of x_k according to the size of x. The kernel size is adaptively computed using Eq. (2.6), given that the stride is equal to 1 and no padding is used.
- The size of the current input x is **greater** than the size of the incoming input x_k. Zero-padding is used on x_k until both tensors' dimensions match. This case is less frequent than the previous one.
- The size of the current input x is **equal** to the size of the incoming input x_k. No operation is required, as both tensors are compatible for concatenation.

In Fig. 3.3, these three scenarios are graphically explained.

Fully-connected blocks are always placed at the end of the feature extraction section (simple convolutional blocks or DenseBlocks), and are described by their number of neurons (see Table 3.1). ReLU is the chosen activation function. As the CXR image classification scenario

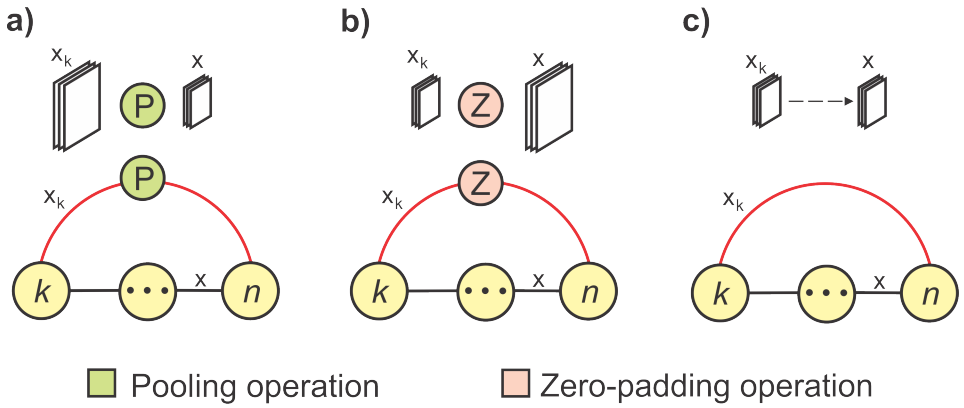

FIGURE 3.3: Adjustments to the spatial resolution of the feature maps transferred through skip connections (x_k) in order to be concatenated with the current input x. **a)** When the size of x_k is greater than the size of x, max pooling is applied on x_k. **b)** When the size of x_k is less than the size of x, zero padding is applied on x_k. **c)** When the size of x_k is equal to the size of x, no operation is needed.

TABLE 3.2: Evolvable hyperparameters in the Wang encoding. Yellow rows correspond to convolutional blocks, while the pink row corresponds to the fully-connected blocks.

	Values
No. of Conv. Layers	{3, 4, 5}
Growth Rate	[3, 12]
No. of Neurons	{4, 8, 16, 32, 64, 128}

is a three-class classification problem, a last fully-connected block with three neurons is always added, using the softmax activation function.

The utilization of this encoding could provide for important advantages, specially in contrast with the Wang encoding. First, utilizing a hybrid encoding allows to independently represent CNNs' motifs, as well as to separately manipulate them. Secondly, using simpler blocks would imply that adding more blocks would not cause an important growth in the number of parameters. In the Wang encoding, in turn, the addition of a DenseBlock implies the utilization of a number of new convolutional layers, requiring new parameters accordingly. Even when using fitness mechanisms that enforce the networks to keep a reduced size, the Wang encoding might be biased towards larger architectures, or limited without fully discovering a more competitive topology. The Wang encoding's hyperparameters values can be found in Table 3.2.

Next, the evolutionary operators of the DeepGA are designed to cope with this new encoding (and the Wang encoding), while taking advantage of the search power of the Genetic Algorithm.

3.2.2 Evolutionary Operators

As in a standard GA, DeepGA utilizes three main operators; parents selection, crossover, and mutation. Special versions of them are designed in order to handle the new hybrid encoding. Furthermore, a variant of each operator is employed to manipulate the Wang encoding and perform experimental comparisons within the same optimization framework.

Parents Selection. In Chapter 2, the deterministic and stochastic tournament selection methods were introduced. In DeepGA, a modification on the stochastic version is proposed in order to further encourage exploration during the search.

The purpose of an alternative parent selection algorithm relates to the possibility of choosing a parent from the population that is not very successful, but whose potential to generate improved offspring is latent. As the networks grow in size during evolution, it is intuitive to expect that larger networks are going to perform better, as deeper CNNs might possess more powerful representation capabilities. However, in order to enforce the search in a more efficient region of the search space, smaller individuals could be useful. For this reason, we propose to use the tournament selection shown in Algorithm 5:

Algorithm 5 Modified Stochastic Tournament Selection

Require: A population *Pop*, probability p_t.
Output: A parent solution *parent*.
  Randomly select s individuals for tournament from *Pop*.
  if $U(0,1) \leq p_t$ **then**
    Choose the individual with the highest fitness as *parent*.
  else
    Choose a random individual as *parent*.
  end if

It can be seen that a probability p_t of choosing the best individual needs to be set. For this research, 0.8 is chosen. It is expected that 80% of the total number of selections is biased

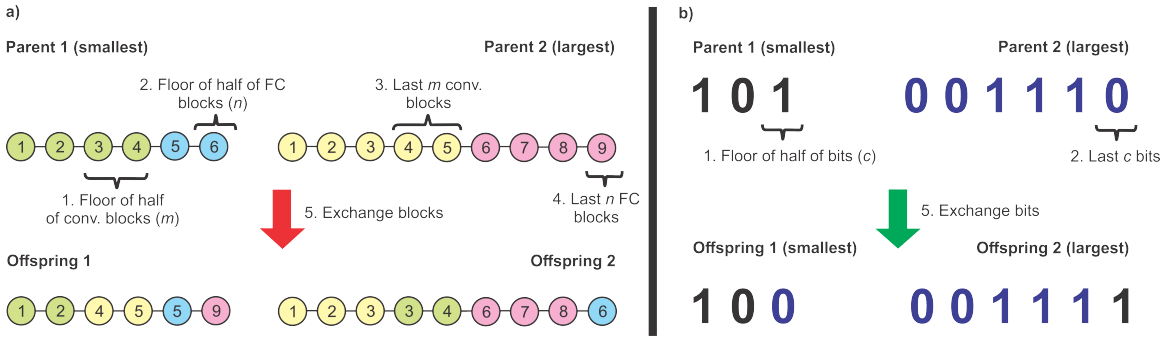

FIGURE 3.4: The crossover operation of DeepGA. **a)** The first-level crossover exchanges the last m convolutional blocks and the last n fully-connected blocks of both parents. **b)** The second-level encoding exchanges the last c bits of both parents' binary strings.

towards the best individual in a tournament, while in the remaining 20% of the cases, another possibly less fit individual is going to be utilized for crossover. It is important to notice that even in the second case, the best individual has a chance to be selected as well.

Crossover. The crossover operator combines two parents to generate two offspring. The recombination must be performed in such a way that the new individuals have information solely from their parents, without any added datum. As the hybrid encoding is formed by two levels, the crossover in DeepGA is carried out in two parts; the first-level crossover and the second-level crossover. This operation results in two offspring individuals.

The first-level crossover consists in exchanging a certain number of blocks between the parents. First, the smallest parent is identified; let m be the floor function of half of the number of convolutional blocks, and let n be the floor function of half the number of fully-connected blocks in the same smallest parent. The first-level crossover consists in exchanging the last m convolutional blocks between both parents, as well as their last n fully-connected blocks. The two new offspring o_1 and o_2 will have the same size as the parents pa_1 and pa_2, respectively.

In the second-level encoding, c is the floor of the half of the bits in the smallest parent's binary string (second-level encoding). The last c bits of both parents are therefore exchanged. Fig. 3.4 presents an example of the entire crossover routine.

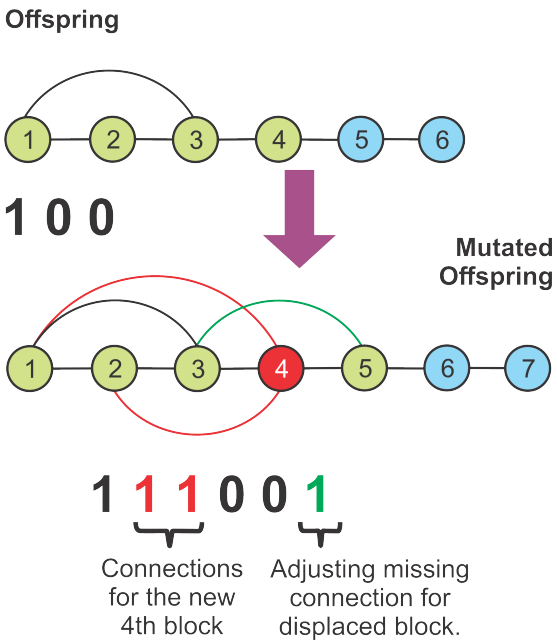

FIGURE 3.5: The mutation of DeepGA. When a new convolutional block is added (red circle in the fourth position), a number of new random bits are introduced in the new block's corresponding position in the binary string. Each displaced block at the right side of the new block requires one new random bit at the matching position of its corresponding sub-string.

For the Wang encoding, a similar operation is performed. As a reminder, each DenseBlock has its own binary string. Thus, when the block exchange is performed, the binary strings are also exchanged between parents. It can be seen that more modular encodings can also involve higher-level operations.

Mutation. The mutation promotes unbiased exploration through the search space, and is applied to the offspring. In DeepGA, the mutation is also divided in two parts: (1) the first-level mutation, which affects the blocks, and (2) and the second-level mutation, that affects the connections. The latter part is simpler, and expresses by choosing a random bit in the binary string of an offspring individual, and flipping its value. The former part can appear in two possible forms: (a) restarting a block (if $U_1(0,1) \leq 0.5$), which means a random change in all the block's hyperparameters, or (b) adding a new block (if $U_1(0,1) > 0.5$). In the proposed encoding, adding a new convolutional block requires to adjust the binary string, as shown in Fig. 3.5.

The decision of adding a convolutional block or a fully-connected block is also dependent on a random uniform number U_2; if $U_2(0,1) \leq 0.5$, a convolutional block is added; otherwise, a fully-connected block is added. If the fully-connected block option is selected, the second-level encoding requires no further change. The blocks addition uses randomly generated hyperparameters.

In the Wang encoding, the same mechanism is applied. The mutation can be expressed as restarting a block or adding a new block, depending on a random number. Furthermore, the addition of a DenseBlock or a fully-connected block is also decided on a stochastic basis. Again, as each DenseBlock possesses its own binary string, there does not exist the need to make supplementary modifications when adding new blocks.

The three evolutionary operators are merged together under the GA framework of Algorithm 3. Elitist replacement is selected, so as to preserve the best individuals as generations go by.

3.2.3 Fitness Function

As mentioned previously in this thesis, this work proposes a Neuroevolution approach that focuses not only on maximizing classification performance, but also on designing competitive CNNs in terms of computational efficiency. In this chapter, a single-objective alternative is explored in order to solve the CXR classification problem.

The computational cost of training a CNN is correlated with its number of trainable parameters. These parameters include the filter's weights, the batch normalization variables, and the connections' weights. As the classification performance is measured in terms of accuracy, the number of parameters need to be included into a new fitness function.

Let *cnn* represent an arbitrary instance from the proposed encoding (or the Wang encoding). The number of parameters of this instance is *NP*. Given a weight value w in the range $[0, 1]$, the fitness function of DeepGA is computed using Eq. (3.4):

$$f(cnn) = (1 - w) * accuracy(cnn) + w * \frac{MP - NP}{MP} \tag{3.4}$$

where *MP* is an user defined parameter that corresponds to the maximum number of parameters that are allowed without penalization. Eq. (3.4) is a weighted sum in which the accuracy and the number of parameters play an important role. The weight w aggregates a proportion of the difference between *NP* and *MP* with respect to the total *MP*. It is worth noticing that when $NP > MP$, the second term of the function is negative, affecting the weighted accuracy. On the other hand, if $NP < MP$, the second term is positive and improves the overall fitness. Moreover, when $MP - NP$ is a larger positive value, the fitness increases proportionally, meaning that smaller CNNs are preferred. Based on preliminary experiments, the weight w is set to 0.3, as it showed to aid at dealing with both objectives (accuracy and complexity) successfully. The maximum allowed number of parameters *MP* is set to 2 million.

The survival selection based on this fitness function is performed using the $\mu + \lambda$ approach from Evolution Strategies: the population μ is merged with the offspring λ, and only the best N individuals are selected as the new population.

This new fitness function is used to guide the search of CNN architectures that improve on the *state-of-the-art* in classification of CXR images. Next, the methodology and experimental settings to assess the performance of DeepGA are detailed.

3.3 Methodology

The purpose of the experiments in this chapter is twofold: first, they aim to test the **Hypothesis 1**, while measuring the performance of DeepGA in comparison to the *state-of-the-art* CNNs for CXR classification (see Table 2.6). Second, the experiments aim to evaluate how a less modular hybrid encoding behaves against a higher modular one, in terms of both accuracy and complexity (**Hypothesis 2**). Both hypotheses are stated again:

- **Hypothesis 1:** A Genetic Algorithm with a compact neural encoding and a bi-objective fitness approach can find Convolutional Neural Networks able to classify lung diseases (including COVID-19) in CXR images with (a) an accuracy of 90%

or more, (b) specificity and sensitivity values within the range of the *state-of-the-art*, and (c) a lower number of parameters with respect to the previously used architectures.

- **Hypothesis 2:** A hybrid encoding based on simple convolutional blocks finds and binary connectivity patterns helps to find less complex networks with respect to a hybrid encoding based on DenseBlocks.

DeepGA is executed 30 times using the proposed hybrid encoding, and 30 times using the Wang encoding. In both cases, the training of the CNNs is achieved using backpropagation, with a learning rate $lr = 1 \times 10^{-4}$ and a batch size of 24 images. Only ten epochs of training are used, as they are enough to evaluate the behavior of the CNNs, as suggested by Sun et al. (2020). During the Neuroevolution, 70% of the image dataset is used for training, while the other 30% is used for validation. The fitness function utilizes the validation accuracy only. The best individual of each execution is reported in terms of its architecture, its fitness value, its accuracy, and its number of parameters. Furthermore, this final CNN is used to compute the sensitivity and specificity for each of the three classes.

In the proposed hybrid encoding, the CNNs are randomly initialized with a number of convolutional blocks in the range $[2, 5]$ and a number of fully-connected blocks in the range $[1, 4]$. The second-level encoding is a binary string with random bits. In the Wang encoding, the number of DenseBlocks is also initialized within the range $[2, 5]$; each DenseBlock is randomly created with a number of convolutional layers in the range $[3, 5]$, in order to have at least one skip connection. The DenseBlocks can have a growth rate in the range $[3, 12]$.

As DeepGA is based on the standard GA, a series of parameters are required: a population size ($N = 20$), the number of generations ($T = 50$), the crossover rate ($\mathit{CXPB} = 0.7$), the mutation rate ($\mathit{MUPB} = 0.3$) and the tournament size ($t_{size} = 5$). These parameters are equal for the two encodings. These parameters were defined based on a small series of preliminary experiments. The computational costs associated to training a large number of CNN architectures per execution prevents the utilization of automatic parameter tuning software such as *IRACE* (López-Ibáñez et al. 2016).

The experiments were executed on the High Performance Computing (HPC) system from the Supercomputing Laboratory of Bajio (*Laboratorio de Supercómputo del Bajío*, in Spanish). These included servers with two Intel Xeon Silver 4214 processors with 12 cores each and 128 GB of RAM. Each server possesses two NVIDIA Titan RTX GPUs with 24 GB. The framework of DeepGA was programmed using Python 3.7 and the PyTorch package for Deep Learning.

3.4 Results and Discussion

The experimental process of DeepGA is provided for 30 samples of different metrics from the two encodings, as discussed previously. The executions closest to the median of the fitness function in both cases are used to generate the convergence plots (see Fig. 3.6). The fitness function, the accuracy, and the number of parameters are plotted independently, and the results of the two encodings are compared.

It is noticeable from Fig. 3.6 that the convergence speed of the proposed encoding seemed to be faster, which leads DeepGA to settle at its final solution in 22 generations, while with the Wang encoding, the settlement occurs at the $48-$th generation. This could be related to an early convergence to a local optima, however, the proposed encoding in the three plots finds better final results in comparison to the Wang encoding, with respect to the three metrics; fitness function, accuracy, and number of parameters.

The samples are now tested using the Kolmogorov-Smirnov (KS) Test to appraise normality in their distributions. Table 3.3 presents the resulting p$-$value of each sample. With a confidence interval of 5%, a p$-$value larger than 0.05 accepts the hypothesis of a sample coming from a normal distribution, otherwise the hypothesis is rejected. In all cases, the samples did not follow a normal distribution.

In view of the results from the KS test, a non-parametric test is used to compare the DeepGA samples from both encodings. The Wilcoxon Rank Sum Test at 5% is chosen. This test is useful to measure if two independent populations have equal distributions. With a p$-$value

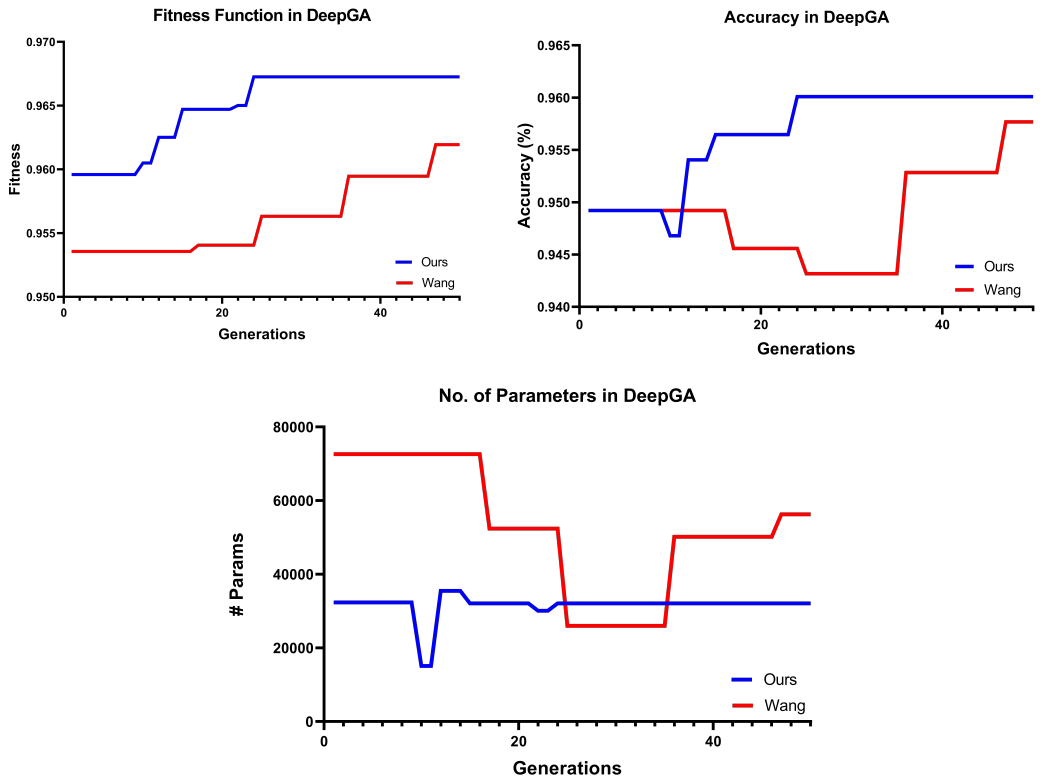

FIGURE 3.6: Convergence plots over 50 generations of the fitness function (upper left), the Accuracy (upper right), and the number of parameters (below) of the median executions. The proposed encoding is plotted in blue, whilst the Wang encoding is plotted in red.

TABLE 3.3: Kolmogorov-Smirnov Test results from the samples obtained with DeepGA using the proposed encoding (our encoding) and the Wang encoding.

	Metric	p-value	KS Result		Metric	p−value	KS Result
Our Encoding	Fitness	8.83×10^{-19}	Not Normal	**Wang Encoding**	Fitness	9.15×10^{-19}	Not Normal
	Accuracy	1.08×10^{-18}	Not Normal		Accuracy	1.23×10^{-18}	Not Normal
	# Params	5.31×10^{-27}	Not Normal		# Params	5.31×10^{-27}	Not Normal
	COVID-19 Speficity	1.51×10^{-18}	Not Normal		COVID-19 Speficity	9.5×10^{-19}	Not Normal
	COVID-19 Sensitivity	5.31×10^{-27}	Not Normal		COVID-19 Sensitivity	5.31×10^{-27}	Not Normal
	Pneumonia Specificity	2.85×10^{-18}	Not Normal		Pneumonia Specificity	1.21×10^{-18}	Not Normal
	Pneumonia Sensitivity	9.51×10^{-18}	Not Normal		Pneumonia Sensitivity	1.43×10^{-18}	Not Normal
	Healthy Specificity	4.62×10^{-18}	Not Normal		Healthy Specificity	1.12×10^{-18}	Not Normal
	Healthy Sensitivity	1.17×10^{-17}	Not Normal		Healthy Sensitivity	6.5×10^{-18}	Not Normal

TABLE 3.4: Mean and standard deviation values of fitness function, accuracy, and the number of parameters (# Params), presented as (M $\pm$ S). Also, the median values and the 25−th and 75−th percentiles. The p−values from the Wilcoxon test (W) indicate when the proposed encoding surpassed the Wang encoding (**+**), when the Wang encoding surpassed the proposed encoding (**-**), or when both yield equal results (**=**).

	Our Encoding				**Wang Encoding**					
Metric	**M $\pm$ S**	**Median**	**P25**	**P75**	**M $\pm$ S**	**Median**	**P25**	**P75**	**p-value**	**W**
Fitness	0.9671 ± 0.0037	0.9670	0.9648	0.9693	0.9619 ± 0.0019	0.9619	0.9606	0.9631	1.47×10^{-7}	**+**
Accuracy	0.9593 ± 0.0056	0.9588	0.9564	0.9625	0.9504 ± 0.0037	0.9504	0.9492	0.9555	1.03×10^{-6}	**+**
# Params	28944 ± 8974.11	30002	24049.5	32438	29770.23 ± 10872.72	29395.5	20858.75	35014.5	0.709	**=**

greater than 0.05, the null hypothesis (H_0) is accepted, in which the two populations are equal. Otherwise, the alternative hypothesis is accepted (H_1), meaning that the populations' distributions are not equal, and there are significant statistical differences. The median value of the samples is used to determine the superiority or inferiority between them, in case of H_1 being accepted. Table 3.4 shows the means, the medians, and standard deviations of the fitness function, the accuracy, and the number of parameters of DeepGA with both encodings. Furthermore, the p−value from the Wilcoxon test obtained from the comparison of each metric is also displayed.

It has been found that DeepGA with the proposed hybrid encoding obtained a better fitness (0.9671 ± 0.0037, median: 0.9670) and accuracy performance (0.9593 ± 0.0056, median: 0.9588) than using the Wang encoding (**fitness:** 0.9619 ± 0.0019, median: 0.9619, and **accuracy:** 0.9504 ± 0.0037, median: 0.9504). These results could be explained because of one or more of the following reasons: (1) the representation power of the proposed encoding is slightly superior, as the transfer of previous feature maps through skip connections provide for more information during classification, (2) networks shallower depth are better at this task, as adding more DenseBlocks implies overfitting the images distribution, (3) simply, the search space covered by the Wang encoding was limited in terms of accuracy and overall fitness.

On the other hand, the number of parameters (# Params) resulted equal using both encodings, which confronts the intuition that was initially presented; a less modular encoding (our encoding) would be less biased towards larger networks in contrast to a more modular encoding (Wang encoding), given equal Neuroevolution settings. The experimental results

TABLE 3.5: Mean and standard deviation (std. dev.) values of specificity and sensitivity with respect to the three classes (COVID-19, pneumonia and healthy patients), presented as (M $\pm$ S), as well as the median and the 25−th and 75−th percentiles. The p−values of the Wilcoxon Rank Sum Test comparing both results from both encodings are also provided.

	Our Encoding				**Wang Encoding**					
Metric	**M $\pm$ S**	**Median**	**P25**	**P75**	**Mean $\pm$ std. dev**	**Median**	**P25**	**P75**	**p-value**	**Wilcoxon**
COVID-19 Specificity	0.9615 ± 0.0074	0.964	0.9578	0.9673	0.9642 ± 0.0041	0.9644	0.9632	0.9658	0.6632	=
COVID-19 Sensitivity	0.965 ± 0.0015	0.9686	0.9607	0.9737	0.9558 ± 0.0096	0.9586	0.9506	0.9619	0.709	=
Pneumonia Specificity	0.9634 ± 0.0143	0.968	0.9589	0.9731	0.9626 ± 0.0057	0.9632	0.9589	0.966	0.1503	=
Pneumonia Sensitivity	0.9478 ± 0.0198	0.9529	0.9424	0.9581	0.9518 ± 0.0064	0.9512	0.9479	0.9562	0.9318	=
Healthy Specificity	0.9646 ± 0.0154	0.9667	0.9608	0.9724	0.9627 ± 0.0052	0.9635	0.9599	0.9658	0.0358	+
Healthy Sensitivity	0.8951 ± 0.0206	0.8977	0.8873	0.9086	0.9052 ± 0.0105	0.9069	0.8989	0.9112	0.0482	-

show that, for this application scenario, both encodings can satisfy the requirement of finding compact CNNs with competitive performance equally. The **Hypothesis 2** is rejected.

In the **Hypothesis 1**, it is argued that DeepGA is able to find CNNs capable of classifying lung diseases in CXR images with (a) at least 90% of accuracy, (b) specificity and sensitivity values within range of the *state-of-the-art* ($[0.8996, 0.996]$ and $[0.8535, 1]$, respectively), and (c) a lower number of parameters with respect to the previously used architectures (minimum value is: 1.164 millions). The obtained results are now statistically analyzed and compared against the methods shown in Table 2.6.

First, the One-Sample Wilcoxon Rank Sum Test and the Sign Test are utilized to evaluate the robustness of the obtained samples. These non-parametric tests use the median as the statistic of reference (Gibbons and Chakraborti 2003). The Wilcoxon Rank Sum Test, unlike the Sign Test, assumes that the data distribution is symmetric about its median. Both tests are chosen to provide for more statistical confidence to this study.

The medians of the samples of DeepGA with both encodings are tested to verify if their distributions are robust around these values. Table 3.6 shows the median values of the accuracy, number of parameters, specificity, and sensitivity. The last two metrics are averaged across the three classes. The p−values of the statistical tests are also introduced. It has been

TABLE 3.6: Statistical tests of the samples of accuracy, number of parameters (# Params), average specificity across classes, and average sensitivity across classes. In the evaluator column, each sample is described by its median, its Wilcoxon Rank Sum Test p−value and its Sign Test p−value.

Metric	Our Encoding	Wang Encoding	Evaluator
Accuracy	0.9588	0.9504	**Median**
	1	0.4049	**Wilcoxon**
	0.6762	0.0223	**Sign**
# Params	30002	29395.5	**Median**
	1	1	**Wilcoxon**
	0.21884	0.8199	**Sign**
Avg. Specificity	0.9652	0.9637	**Median**
	1	1	**Wilcoxon**
	0.548	0.4801	**Sign**
Avg. Sensitivity	0.9395	0.9392	**Median**
	1	1	**Wilcoxon**
	0.538	0.1544	**Sign**

demonstrated that the samples' medians also belong to their arbitrary continuous distributions (as p−values are larger than the confidence value of 0.05 in all cases except one).

By validating the statistical nature of the accuracy samples, it is now possible to ensure that the accuracy of DeepGA for CXR image classification of lung diseases is higher than 90%, both in terms of the median and the mean values. In the related *state-of-the-art* research, the averaged specificity lies between the range $[0.8996, 0.996]$ and the averaged sensitivity lies between the range $[0.8535, 1]$. With the proposed encoding, the mean±std. dev. of the averaged specificity and sensitivity values are 0.9632 ± 0.0092 and 0.9353 ± 0.0145 respectively, showing to be inside the range of performance found in the specialized literature. The same behavior occurs using the Wang encoding, whose averaged specificity and averaged sensitivity are 0.9632 ± 0.0027 and 0.9376 ± 0.0038, respectively.

In Fig. 3.7, the median executions' confusion matrices of the crossvalidation process with both encodings are shown. The confusion matrix is a useful visual resource that allows to analyze the classification performance of an algorithm. Here, a very appealing behavior can be observed, as most images are correctly classified. In terms of COVID-19, it can be seen that the CNNs tend to confuse these images with those from other types of pneumonia. On the other hand, only few images from COVID-19 patients are misclassified as healthy, which is a highly desired feature due to the spreading factor caused by false negative patients.

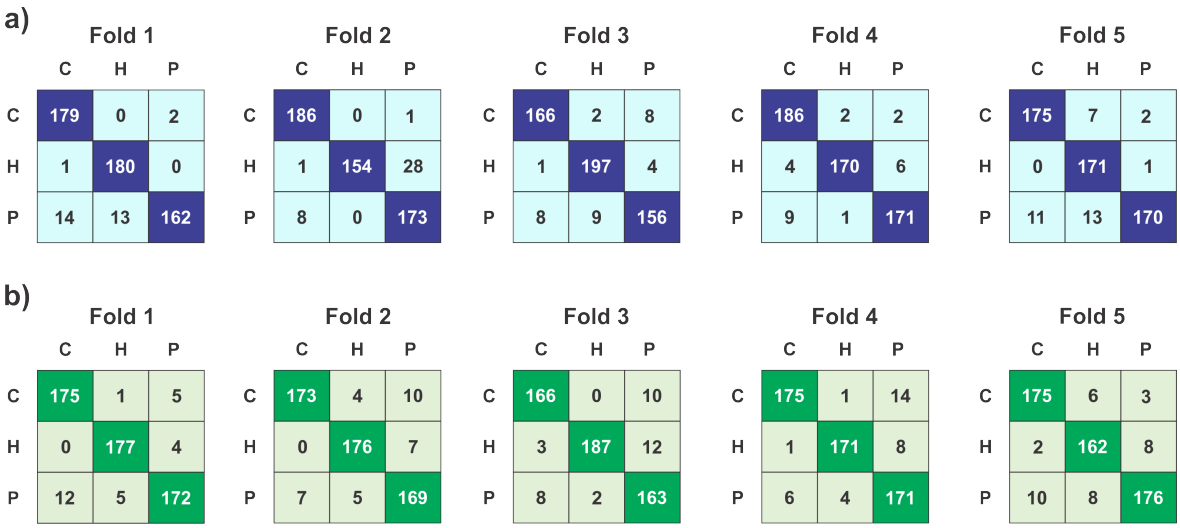

FIGURE 3.7: Confussion matrices of DeepGA with **a)** the proposed encoding, and **b)** the Wang encoding. Both matrices are obtained from 5-fold crossvalidation for COVID-19 (C), Healthy (H), and Pneumonia (P) patients. Rows correspond to the true values, whilst columns correspond to the predicted values.

Until now, the classification performance of the CNNs found by DeepGA has been validated. The median of the number of parameters have also been tested (as seen in Table 3.6). As the p−values for both the Wilcoxon and the Sign tests are greater than 0.05, the median number of parameters can be effectively used as a reliable parameter. The smallest CNN among the *state-of-the-art* is the DarkCovidNet (Ozturk et al. 2020), with 1.164 million parameters. With the proposed encoding, the execution whose fitness is closer to the median value (0.967) achieved 96% accuracy with only 32107 parameters. Using the Wang encoding, the execution whose fitness is closer to the median (0.9619) obtained an accuracy of 95.76% and 56261 parameters. These values demonstrate how Neuroevolution can reach high performances with minor computational costs. In Fig. 3.8, the obtained CNNs from DeepGA are plotted in terms of their accuracy $(\%)$ and their number of parameters (in millions). It is worth noticing that the x-axis is re-scaled to its $log10$ form to facilitate the analysis. DeepGA yields to CNNs with significant performance in terms of accuracy, but with a number of parameters that is two orders of magnitude below those networks in the *state-of-the-art*.

With this final results, the **Hypothesis 1** is accepted under the single-objective approach. The final architectures using both encodings were also analyzed. Fig. 3.9 displays the best

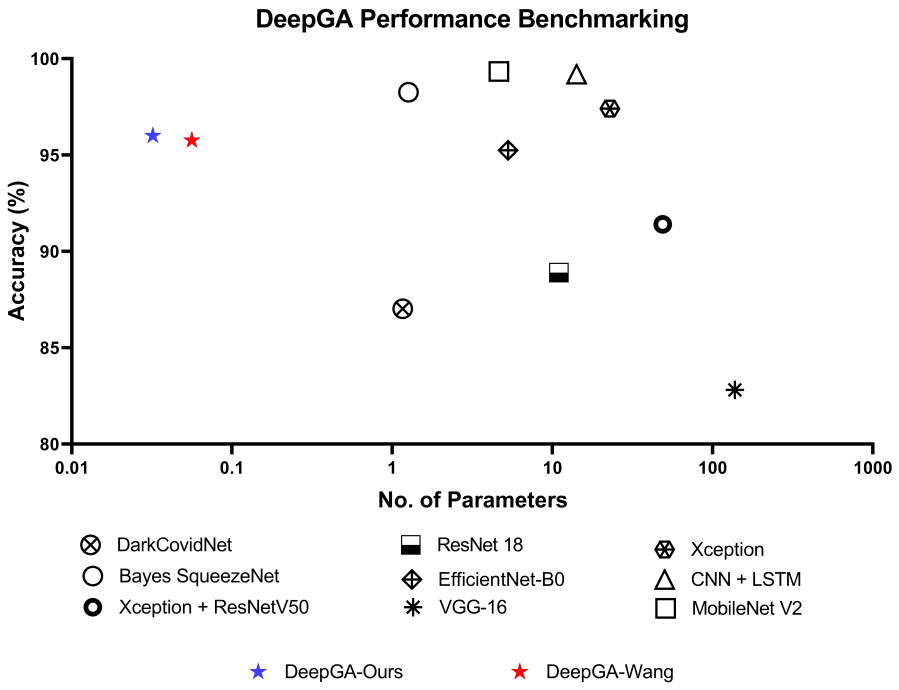

FIGURE 3.8: The performance comparison between the median execution of DeepGA using the proposed encoding (blue star) and the Wang encoding (red star), against the CNNs architectures that have been reported in the literature. Each network is a point composed of the number of parameters in millions (x-axis in $log10$ scale) and the accuracy (y-axis, no scaling).

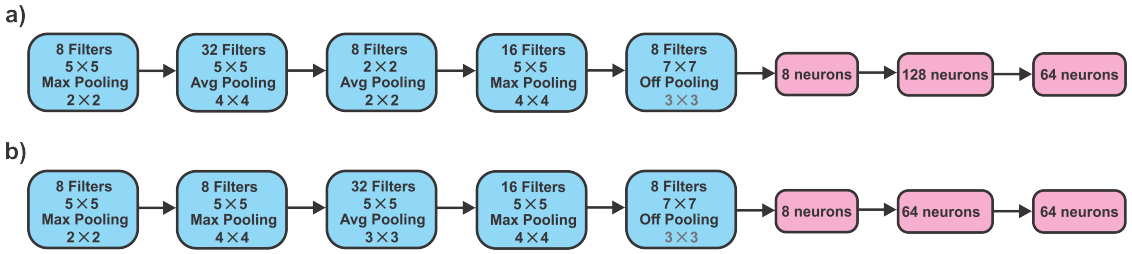

FIGURE 3.9: The best found CNN architecture with the proposed encoding from **a)** the execution closest to the fitness median, and **b)** the execution with the highest fitness.

CNNs architectures of the median and best executions in terms of fitness, using the proposed encoding. Interestingly, none of these individuals required any skip connection to achieve their performance. Furthermore, it is worth noticing that the hyperparameters of both architectures are similar, which could supply for information about the optimality of this particular problem.

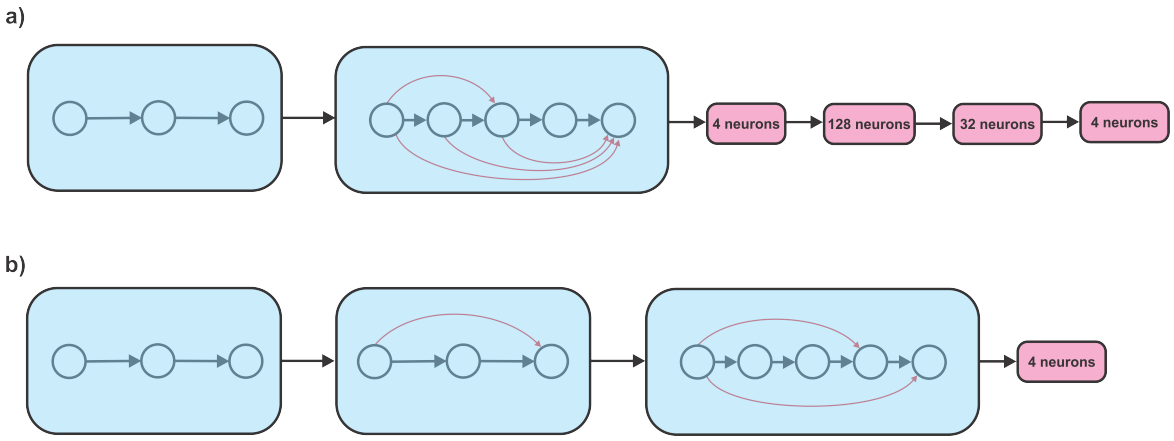

FIGURE 3.10: The best found CNN architecture with the Wang encoding from **a)** the execution closest to the fitness median, and **b)** the execution with the highest fitness.

In Fig. 3.10, the best and median execution's architectures using the Wang encoding are shown. It is noticeable that both results differ between each other, requiring a different number of DenseBlocks and fully-connected blocks. Also, the connectivity patterns in the DenseBlocks are not comparable.

The compactness of the discovered CNNs provides for insights of how impactful EAs can be in face of the challenging design of these networks for very specific problems. Neuroevolution can reduce the burdens of designing and applying Deep Learning algorithms to niche problems while greatly improving the performance and maintaining affordable computational costs.

3.5 Chapter Summary

In this chapter, the image collection and preprocessing procedure have been described, which made possible the process of classification. The Deep Genetic Algorithm (DeepGA) was proposed and detailed, as a Neuroevolution framework based on the standard GA for hybrid encodings.

The research methodology to evaluate the performance of DeepGA using the proposed hybrid encoding and the Wang encoding have also been presented. It has been found that the proposed encoding is able to find CNNs with higher classification quality with respect to

the Wang encoding. However, both encodings can discover neural architectures with the same complexity (number of parameters). In contrast to the *state-of-the-art* in CXR images classification for COVID-19, DeepGA generally equals the performance of the proposed hand-crafted CNNs. Nonetheless, the discovered architectures highly surpassed the published approaches in terms of complexity, which demonstrates the potential of Neuroevolution to transform the process of designing compact convolutional neural networks.

As an output from this chapter, a conference paper was submitted and accepted to the Workshop of Neuroevolution at Work from the ACM Genetic and Evolutionary Computation Conference (GECCO) 2021.

CHAPTER 4

Multi-Objective Deep Genetic Algorithm

In this chapter, the multi-objective version of DeepGA is presented, under the name *Multi-Objective Deep Genetic Algorithm* (MODeepGA). This framework is developed to explore a different fitness approach for Neuroevolution of CNNs. This method is explained next, as well as the utilized methodology to empirically test the advantages of using the proposed encoding against the Wang encoding in a multi-objective optimization scheme.

4.1 MODeepGA

MODeepGA works under the original evolutionary operators used in DeepGA. However, the fitness evaluation does not rely on a linear aggregating fitness function, but in the individual objectives that are taken into account in this work.

As seen in Chapter 2, multi-objective optimization problems are tasks in which a series of conflicting fitness functions are optimized. In this case, these two objectives are the accuracy (equivalent to the classification error in minimization) and the number of parameters. A Pareto Front (PF) is built with the fitness values of the individuals of a population, which is later used to measure the quality of the search.

In MODeepGA, the population cannot be sorted with respect to any of the fitness functions individually, but needs to be sorted in terms of non-dominance, using Algorithm 4 (see Chapter 2). Hence, each individual is always associated with the two fitness values instead of a single one. Those individuals that are non-dominated by each other, but dominate the rest of the population are placed inside the first front, and the reminder is sorted again to

form the second front and so on. As in DeepGA, an $\mu + \lambda$ replacement approach is used; the population μ and offspring λ are merged together, and the first μ individuals of the sorted set are chosen to be part of the next generation's population.

4.2 Methodology

The experiments concerning MODeepGA intend to evaluate **Hypothesis 2**; the proposed encoding and the Wang encoding are compared in terms of the quality of the Pareto Fronts, which is measured with the Hypervolume.

MODeepGA was executed 30 times with each encoding. Each CNN in the population is trained during 10 epochs through backpropagation, with a learning rate $lr = 1 \times 10^{-4}$ and a batch size of 24 images. As in DeepGA, 70% of the image dataset is used for training, while the other 30% is used for validation. The classification error on the validation set is reported as one of the two objective functions, whilst the number of parameters is utilized as the second objective function. The initialization of the individuals in both encodings follow the same guidelines as in DeepGA.

The parameter configuration of MODeepGA is similar to that used in the single-objective DeepGA; a population size $N = 20$ individuals, a crossover rate $\mathit{CXPB} = 0.7$, a mutation rate $\mathit{MUPB} = 0.3$ and a tournament size $t_{size} = 5$. The number of generations, however, has been reduced to $T = 20$ because of the nature of Multi-Objective optimization, in which a number of solutions is always maintained, instead of a single solution. This variety of feasible solutions allows to decrease the number of evaluations.

To compute the Hypervolume, the Pareto Front of each execution is first normalized, in order to achieve an accordance of units. The classification error's range is $[0, 1]$, meanwhile the range of the number of parameters is $(0, \infty)$. Each objective in each Pareto Front is normalized so that both functions' range is $[0, 1]$. Equation (2.3) in Chapter 2 is applied to the 20 solutions of each objective function to normalize the Pareto Front. Once this step has been performed, the Nadir Point can be easily selected as $P_n = (1, 1)$, as it represents the opposite

TABLE 4.1: Kolmogorov-Smirnov test results from the samples of Hypervolume using the proposed encoding (our encoding) and the Wang encoding.

Encoding	p-value	KS Result
Our Encoding	3.18×10^{-17}	Not Normal
Wang Encoding	3.79×10^{-17}	Not Normal

TABLE 4.2: Mean and standard deviation (std. dev.) of the Hypervolume, presented as ($M \pm S$), as well as the median, the 25−th and 75−th percentiles. The p−value of the Wilcoxon test indicate that the proposed encoding outperformed the Wang encoding (**+**).

	Our Encoding				Wang Encoding					
Metric	**M ± S**	**Median**	**P25**	**P75**	**M ± S**	**Median**	**P25**	**P75**	**p-value**	**Wilcoxon**
Hypervolume	0.9468 ± 0.0411	0.958	0.9302	0.9707	0.9084 ± 0.0608	0.92	0.8827	0.9462	0.0071	+

worst possible point in the Pareto Front, hence, it is dominated by all the other solutions. Using P_n as the reference point, a deterministic implementation of the Hypervolume metric computation is deployed using MATLAB (Cao 2021). Furthermore, the metric Spacing is also computed based on the normalized fronts.

4.3 Results and Discussion

After the 30 executions of MODeepGA with the proposed encoding and the Wang encoding, two resulting groups of Pareto Fronts were obtained. The Hypervolume and Spacing metrics are computed for each normalized Pareto Front for each of the 30 executions with each encoding. The Kolmogorov-Smirnov test is applied to the Hypervolume results of both samples of 30 values. Table 4.1 shows the mean and standard deviation of the Hypervolume with the proposed encoding and the Wang encoding, as well as the p−value of the KS test and the resulting conclusion. It can be observed that none of the samples follow a normal distribution.

With the knowledge of the non-parametric nature of the samples, the Wilcoxon Rank Sum Test at 5% was applied to statistically compare the samples of both encodings based on their mean Hypervolume and their standard deviation. These values, along with the result of the Wilcoxon Test are presented in Table 4.2.

TABLE 4.3: Kolmogorov-Smirnov test results from the samples of Spacing using the proposed encoding (our encoding) and the Wang encoding.

Encoding	p-value	KS Result
Our Encoding	4.54×10^{-7}	Not Normal
Wang Encoding	4.54×10^{-7}	Not Normal

TABLE 4.4: Mean and standard deviation (std. dev.) of the Spacing metric, presented as $(M \pm S)$, as well as the median, the $25-$th and $75-$th percentiles. The p$-$value of the Wilcoxon test indicate that the Wang encoding surpassed the proposed encoding (**-**).

	Our Encoding				Wang Encoding					
Metric	**M ± S**	**Median**	**P25**	**P75**	**M ± S**	**Median**	**P25**	**P75**	**p-value**	**Wilcoxon**
Spacing	0.0517 ± 0.2817	2.03×10^{-8}	1.09×10^{-9}	0.0004	0.0001 ± 0.0003	2.98×10^{-6}	3.71×10^{-8}	7.19×10^{-5}	4.77×10^{-11}	-

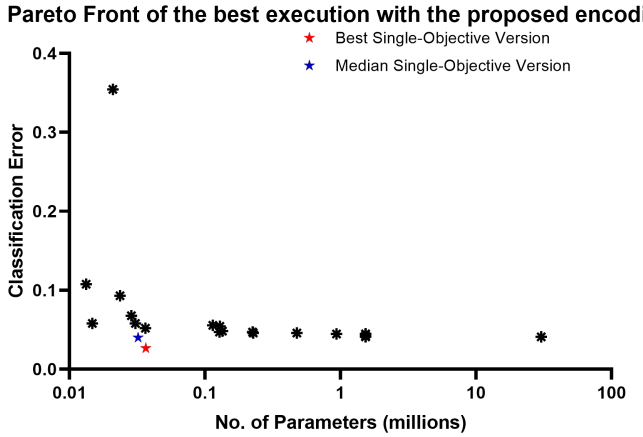

FIGURE 4.1: The Pareto Front of the best execution of MODeepGA with the proposed encoding. Also, the best and median executions from DeepGA (red and blue star, respectively).

The Kolmogorov-Smirnov test at 5% is now applied to the 30 Spacing samples from both encodings. The results shown in Table 4.3 demonstrate that the two samples are non parametric.

The Wilcoxon Rank Sum test at 5% is computed using the samples from both encodings in order to statistically verify significant differences in performance. Table 4.4 shows the resulting p$-$value from the test, showing that MODeepGA with the Wang encoding achieves an overall higher diversity in the final Pareto Fronts.

The best execution of MODeepGA with the proposed encoding obtained a Hypervolume of 1, and its Pareto Front is illustrated in Fig. 4.1.

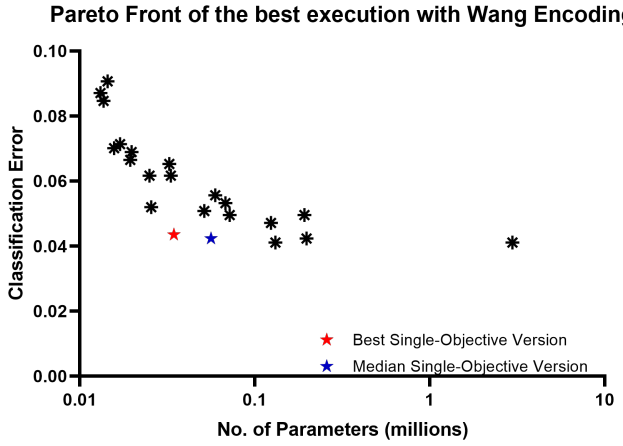

FIGURE 4.2: The Pareto Front of the best execution of MODeepGA with the Wang encoding. Also, the best and median executions from DeepGA (red and blue star, respectively).

With the Wang encoding, the best execution of MODeepGA obtained a Hypervolume of 0.994, and its resulting Pareto Front is shown in Fig. 4.2.

The Pareto Front offers a range of solutions with different trade-offs of the objective functions. In a practical setting, however, a single architecture must be chosen to be deployed. To solve this issue, a technique of Multi-Criteria Decision Making (MCDM) can be utilized (Chiu, Yen, and Juan 2016). The main task of MCDM is to design a so-called *decision maker* able to choose an option out of a set of candidates by taking different criteria or objectives in consideration. As a problem that is closely related to Multi-Objective Optimization, MCDM is here introduced to select one CNN architecture from the final Pareto Fronts.

For this case study, a *knee-based* decision making approach is utilized. A knee-based solution has the following characteristics, based on Chiu et al. (2016):

- Geometrically, it is placed in the knee-region of a Pareto Front, i.e., where the front *bends* (see Fig. 4.3).
- While moving through the knee region, a knee solution can improve one objective significantly without degrading the others.

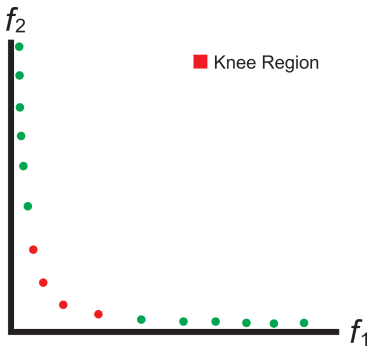

FIGURE 4.3: An example of the solutions that are in the knee region of a Pareto Front.

As has been shown by the aforementioned authors, a knee solution can be found by choosing an individual that minimizes the distance to an ideal point. The ideal point may be defined by the optimization problem if it is known, or can be proposed similarly as in the case of computing the Hypervolume. In this thesis, the ideal point corresponds to $P = (0, 0)$ if the Pareto Fronts are normalized.

For ease of selection by a human user, an MCDM method called *Knee and Boundary Selection* is utilized (Jr. and Yen 2021). This technique returns not only the solution closest to the knee of a Pareto Front, but also two more solutions that have the best advantage in terms of the two objectives. In this method, the *knee solution* corresponds to the individual $x_{knee} = \arg\min_{x \in PF} dist(x, P)$, where PF is the normalized Pareto Front, and $dist$ corresponds to the Euclidean distance. The *boundary light* solution is the individual with the smallest classification error in the front, i.e., $x_{light} = \arg\min_{x \in PF} f_1(x)$. The *boundary heavy* solution is the individual with the smallest number of parameters in the front, i.e., $x_{heavy} = \arg\min_{x \in PF} f_2(x)$.

The search for the knee, the boundary light, and the boundary heavy solutions were performed based on the best execution of MODeepGA with both encodings, based on the Hypervolume. Table 4.5 shows the objective values of the different architectures from both encodings.

The selected architectures from the proposed encoding are shown in Fig. 4.4. It is worth noticing that there is not an important difference between the classification error of the knee and the boundary light solutions. In favor of reducing the number of parameters, a short percentage of accuracy can be sacrificed if needed (maintaining at least 95%).

TABLE 4.5: The objective values of the CNN architectures corresponding to the knee solution, the boundary light solution, and the boundary heavy solution, obtained by the Multi-Criteria Decision Making system.

	Knee		Boundary Light		Boundary Heavy	
Encoding	**Error**	**No. Params**	**Error**	**No. Params**	**Error**	**No. Params**
Ours	0.0459	226479	0.0411	113991	0.1076	13267
Wang	0.0411	131355	0.0411	2970585	0.0870	13139

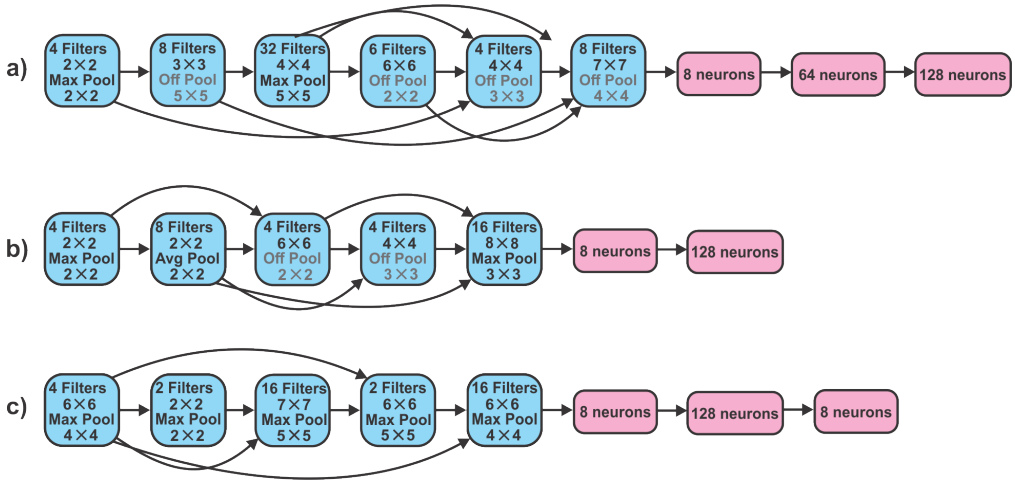

FIGURE 4.4: The three selected encodings obtained by the Knee and Boundary Selection method from the proposed encoding. **a)** Knee solution, **b)** Boundary Light Solution, and **c** Boundary Heavy Solution.

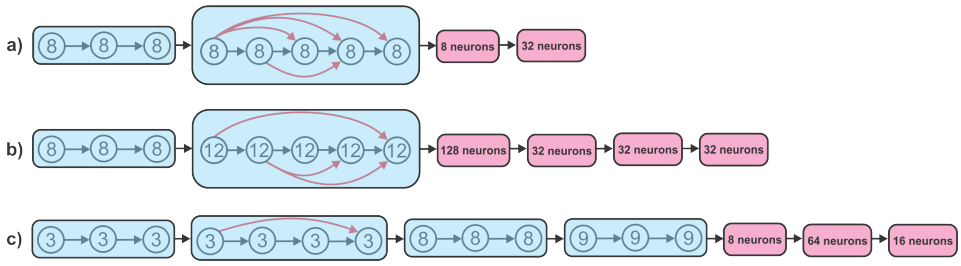

FIGURE 4.5: The three selected encodings obtained by the Knee and Boundary Selection method from the Wang encoding. **a)** Knee solution, **b)** Boundary Light Solution, and **c** Boundary Heavy Solution. Each circle represents a convolutional layer, whose inner circle corresponds to the number of filters (growth rate).

The selected architectures from the Wang encoding are presented in Fig. 4.5. In this case, the knee and boundary light solutions achieved an equal classification error, However, the number of parameters is considerably smaller in the knee solution, thus its selection is encouraged.

4.3.1 Further Studies

Although both, DeepGA and MODeepGA have demonstrated to be competitive at the search of efficient CNN architectures, an important question still arises: *Is Multi-Objective Optimization required to solve this problem?*. The rationale behind this questioning is related to the available computational resources. Evolutionary Multi-Objective optimization is more costly in algorithmic terms than its Single-Objective analogous. In MODeepGA, this condition is particularly easy to perceive as the population holds individuals with higher complexity (and, overall, more complex to train) who still represent a competent trade-off between objectives. This issue also contributed to the constrain of 20 generations in MODeepGA.

MODeepGA comes with the benefit of having a set of solutions from which a user can choose. However, it is important to determine if this benefit surpasses the computational costs hindrance, in contrast to the results obtained with DeepGA. For this section, a series of experiments were carried out in order to provide evidence that helps to provide a conclusion to this concern:

(1) **Experiment 1:** Continuing the best execution until the 50th generation.
(2) **Experiment 2:** Introducing the best solution of DeepGA into the final population of MODeepGA.
(3) **Experiment 3:** Increasing the size of last population of MODeepGA.

Firstly, the Single-Objective solutions of DeepGA were compared with those from MODeepGA. The non-dominance criterion is used to determine if the solutions in the final Pareto Fronts are better than those from DeepGA. If MODeepGA consistently finds architectures whose trade-off between accuracy and complexity surpasses those in DeepGA with fewer generations, then the drawn conclusion would be in favor of Multi-Objective Neuroevolution for this instance.

The best solution from the best execution of DeepGA in terms of fitness is utilized as baseline for comparison purposes. In addition, the median solution is also included as it represents a result that is closer to what would be expected after a random execution. The

TABLE 4.6: Dominance relations between the best and median executions of DeepGA with both encodings, with the 600 solutions from the 30 executions of MODeepGA.

Encoding	Execution	Dominated	Dominator	Equivalent
Ours	Best	0%	69.66%	30.33%
	Median	0.66%	5.5%	87.33%
Wang	Best	0%	49.16%	50.83%
	Median	0.16%	3.5%	99%

dominance criterion (see Algorithm 4) compares each of the solutions in the 30 Pareto Fronts from the executions of MODeepGA with the previously mentioned single-objective solutions. From a total of 600 multi-objetive solutions, Table 4.6 shows statistical results of these comparisons. For this and the following tables, the notation consists of the following: **Dominated** when the single-objective solution is dominated by a multi-objective solution (MODeepGA), **Dominator** when the single-objective solution dominates a multi-objective solution, and **Equivalent** when there does not exist dominance between single- and multi-objective solutions.

It can be seen that most multi-objective solutions are dominated by the best executions of DeepGA with both encodings (69.66% and 49.16%, respectively). This behavior changes considerably in terms of the median solution, where most of the multi-objective solutions achieved an equivalence. This phenomenon occurs because, as expected, the median execution obtained a fitness value of less quality than the best execution, thus it was easier for MODeepGA to equal this result. Fig. 4.1 and Fig. 4.2 shows that single-objective solutions are in a more competitive position than their multi-objective counterparts (see Fig. 2.4 in Chapter 2), as single-objective solutions outperforms several multi-objective solutions in terms of at least one of both objective functions, and are equivalent in the other objective function, which corresponds to the dominance criterion.

4.3.1.1 Experiment 1: evolving beyond 20 generations

The current status of the experimental phase of MODeepGA allowed to explore possible configurations that could explore the performance of multi-objective optimization for this problem instance. The first of these configurations consisted in continuing an execution until

TABLE 4.7: Dominance relations between the best and median executions of DeepGA with the 20 solutions from MODeepGA after 50 generations.

Execution	Dominated	Dominator	Equivalent
Best	0 (0%)	14 (70%)	6 (30%)
Median	1 (5%)	0 (0%)	19 (95%)

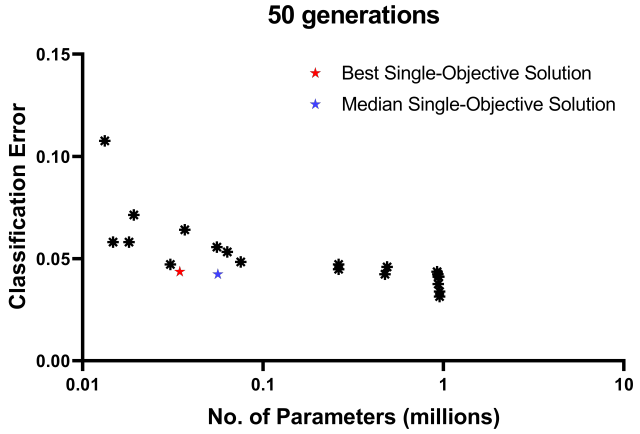

FIGURE 4.6: The Pareto Front obtained from MODeepGA after 50 generations. Also, the best and median executions from DeepGA (red and blue star, respectively).

the 50th generation, to equal the exact number of evaluations performed by DeepGA. The best execution of DeepGA is selected for this experiment, and the resulting dominance relations are shown in Table 4.7. Only the proposed encoding is employed for this experiment.

As in the previous analysis, the best solution of DeepGA dominates most of the solutions from MODeepGA. Multi-objective solutions equal the median solution of DeepGA in most cases. The new Pareto Front from the execution with 50 generations is shown in Fig. 4.6, which also demonstrates that single-objective solutions appear to be in an overall more competitive position.

The original Pareto Front from the best execution of MODeepGA was also plotted together with the new Pareto Front, in order to observe significant changes in the distribution of the solutions. As seen in Fig. 4.7, the new Pareto Front starts to align horizontally, meanwhile the diversity of solutions lie in the number of parameters. This unexpected behavior suggests the

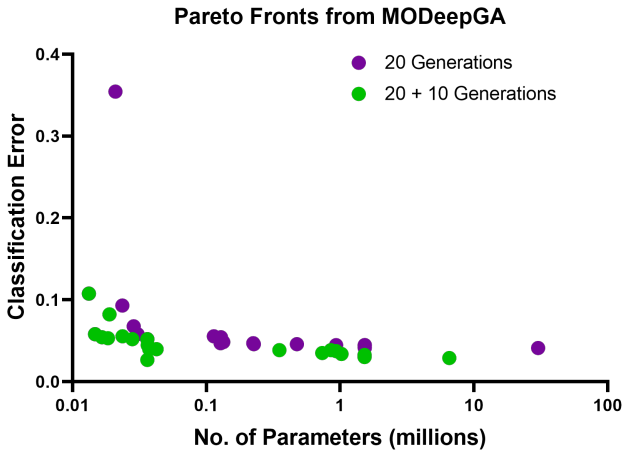

FIGURE 4.7: The Pareto Front from the best execution of MODeepGA with the original 20 generations (green), and the Pareto Front from the same execution continued until the 50−th generation (purple).

possibility of performing a single-objective optimization process on the number of parameters while using the accuracy or classification error as a constraint.

4.3.1.2 Experiment 2: migration of the best DeepGA solution to the MODeepGA population

The next experiment consisted in exploring an experimental setting that could potentially provide for better Pareto Fronts. In this case, the best execution of MODeepGA is utilized; (1) the best solution of DeepGA is introduced to the final population of MODeepGA, (2) the first 10 individuals from the final population of MODeepGA are kept, and (3) the remaining 9 individuals (to form $N = 20$) are randomly generated. Through this method, a higher selection pressure is expected, as a new solution that outperforms all the others has been introduced to the population. The random initialization would help to ensure diversity during search. This experiments was conducted using the proposed encoding for 10 more generations.

Table 4.8 presents the dominance relations of this new execution ($20 + 10$ generations). Contrary to what was originally expected, this new experiment did not achieve an improvement in the final Pareto Fronts with respect to the single-objective solutions. An improvement is

TABLE 4.8: Dominance relations between the best and median executions of DeepGA with the 20 solutions from MODeepGA after the original 20 generations, and with the new execution with the addition of the best DeepGA solution (20 + 10 generations).

Version	Execution	Dominated	Dominator	Equivalent
20 generations	Best	0 (0%)	4 (20%)	16 (80%)
	Median	1 (5%)	19 (95%)	0 (0%)
20 + 10 generations	Best	0 (0%)	12 (60%)	8 (40%)
	Median	8 (40%)	12 (60%)	0 (0%)

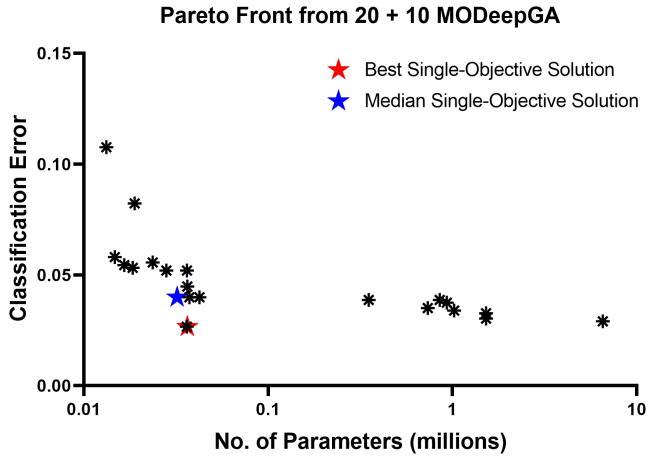

FIGURE 4.8: Pareto Front from MODeepGA with the introduction of the best solution of DeepGA into the population (20 + 10 generations). Also, the best and median executions from DeepGA (red and blue star, respectively).

perceived, however, with respect to the median execution of DeepGA, where more multi-objective solutions achieved to dominate it.

The resulting Pareto Front from this experiment appears in Fig. 4.8. The single-objective solutions have not been surpassed by the Pareto Front yet. It can be also seen that the best solution of DeepGA is still part of the multi-objective population, implying that an improvement has not been achieved.

The original Pareto Front and the newly obtained Paret Front are plotted together in Fig. 4.9 to visually analyze the evolution of the optimization process.

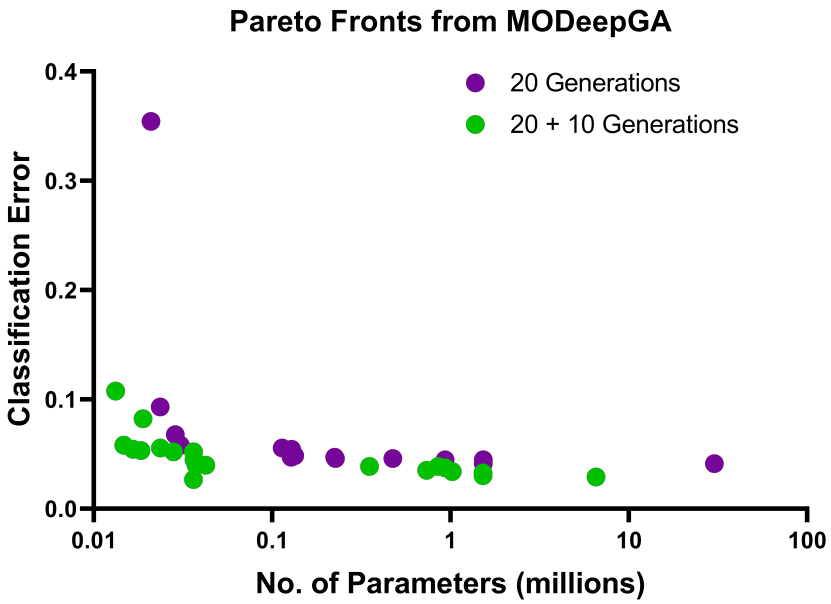

FIGURE 4.9: The Pareto Front from the best execution of MODeepGA with the original 20 generations (purple), and the Pareto Front from the same execution continued for 50 more generations with the best solution of DeepGA added to the population (green).

TABLE 4.9: Spacing and Hypervolume of the original best execution of MODeepGA (20 generations), and with the addition of the best solution of DeepGA (20 + 10 generations).

Version	Spacing	Hypervolume
20 generations	0.001327	1
20 +10 generations	0.000515	0.998

The distribution of the solutions in both fronts in Fig. 4.9 is considerably different. For this reason, Spacing is employed to compare both sets of solutions in terms of their spread. The Hypervolume is also computed. These results are shown in Table 4.9.

The difference of the Hypervolume values between experiments is negligible. Nonetheless, the Spacing metric ranges considerably. In fact, the new experiments outperform the original execution of MODeepGA by one order of magnitude. This indicates that the addition of the best single-objective solution favors an increase of diversity, but it is not enough to improve optimality.

TABLE 4.10: Dominance relations between the best and median executions of DeepGA with the 50 individuals of the Pareto Front of the execution with increased population ($N = 50$) of MODeepGA (20 + 2 generations).

Version	Execution	Dominated	Dominator	Equivalent
20 + 2 generations	Best	0 (0%)	7 (14%)	43 (86%)
	Median	2 (4%)	44 (88%)	4 (8%)

TABLE 4.11: Dominance relations between the best and median executions of DeepGA with the 20 individuals of the best execution of MODeepGA (20 generations), and the first 20 individuals of the execution with an increased population ($N = 50$) (20 + 2 generations).

Version	Execution	Dominated	Dominator	Equivalent
20 generations	Best	0 (0%)	4 (20%)	16 (80%)
	Median	1 (5%)	19 (95%)	0 (0%)
20 + 2 generations	Best	0 (0%)	7 (14%)	43 (86%)
	Median	2 (4%)	44 (88%)	4 (8%)

4.3.1.3 Experiment 3: Increasing the population size towards better diversity

Another possible configuration is the increase of the population size, with a way of increasing diversity, and consequently, improve the optimality of the Pareto Front. A new experiment was performed based on the best execution of MODeepGA. The final population after 20 generations is increased with 30 more individuals (making a total of $N = 50$). The execution is continued for two more generations. As in the previous phases, the dominance relations are shown in Table 4.10.

The dominance with respect to the single-objective solutions is not encouraging. To fairly compare these new results to those obtained with the original execution of MODeepGA, only the first 20 solutions out of 50 are used to calculate the domination relations, as shown in Table 4.11.

There is no clear difference between both experiments, as the number of multi-objective solutions dominated by single-objective ones remain mostly unchanged in both encodings. With the best execution, a very small change can be appreciated. These quantitative results indicate that increasing the population size does not improve the performance in a small number of evaluations. It is worth noticing that increasing the number of generations of this

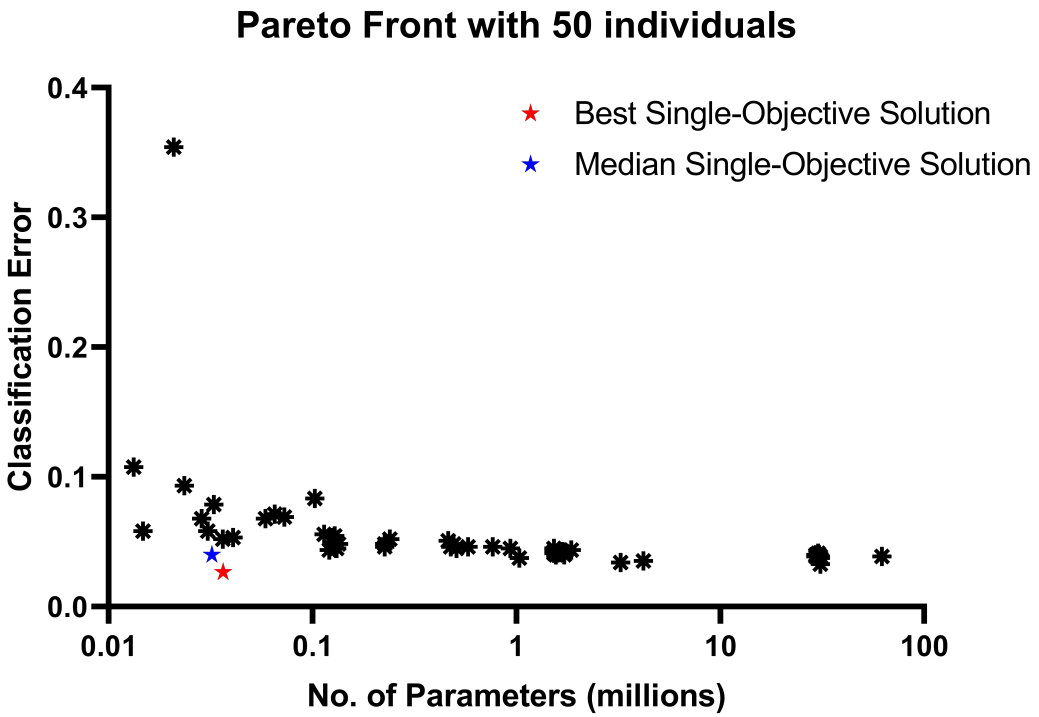

FIGURE 4.10: Pareto Front from MODeepGA with an increased population of 50 individuals. Also, the best and median executions from DeepGA (red and blue star, respectively).

new configuration would increase the number of evaluations by a 375%. In fact, it would be less expensive to run the original version of MODeepGA for 50 generations instead of 20.

The new Pareto Front is shown in Fig. 4.10. As in previous plots, the best and median executions of DeepGA occupy a more competitive position with respect to the solutions in the front.

For a fair visual comparison with the original execution of DeepGA, the first 20 individuals of this new experiment have been plotted along with the original front in Fig. 4.11. A considerable number of solutions are the same in both fronts, which further supports that this approach did not improve the optimality of the final solutions.

Finally, Spacing and Hypervolume are computed to evaluate the new front and to compare these results with the original execution of MODeepGA. Table 4.12 shows that the variation

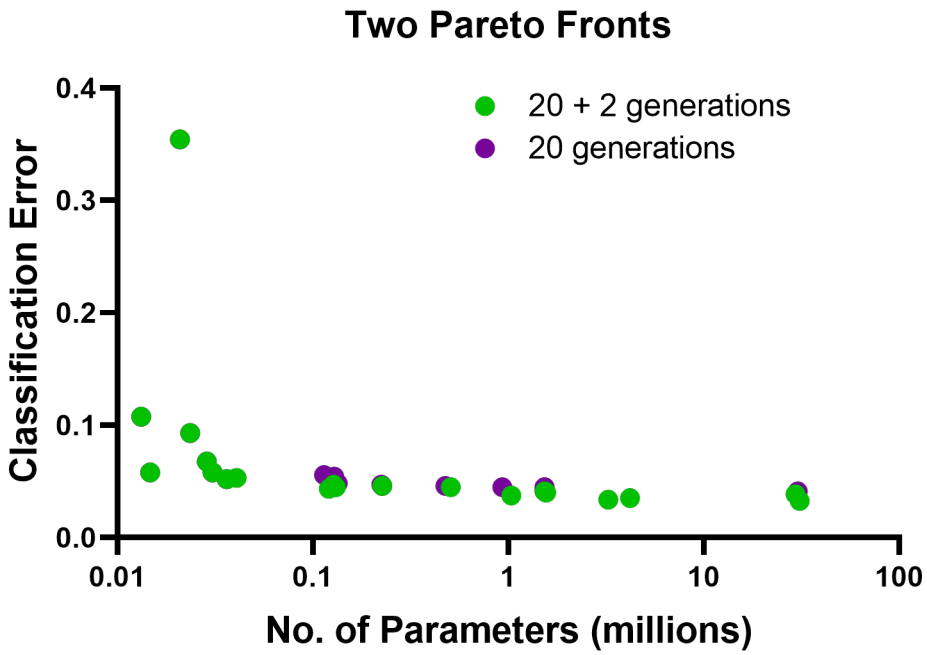

FIGURE 4.11: The Pareto Front from the best execution of MODeepGA with the original 20 generations (purple), and the Pareto Front from the same execution continued for 2 more generations with 30 new solutions in the population (green).

TABLE 4.12: Spacing and Hypervolume of the original best execution of MODeepGA (20 generations), and with the addition of the best solution of DeepGA (20 + 2 generations).

Version	Spacing	Hypervolume
20 generations	0.001327	1
20 + 2 generations	7.59×10^{-9}	0.992

in the Hypervolume values is not considerable. On the other hand, Spacing improves by several orders of magnitude.

The experiments that have been performed so far demonstrate that it is hard to improve the distribution of solutions, although the optimality of the Pareto Fronts remains mostly unchanged in comparison to the original execution of MODeepGA. In theory, utilizing DeepGA is more promising towards obtaining better trade-offs in terms of both accuracy and the number of parameters. In practice, however, MODeepGA has the advantage of being less

computational expensive, with only 220 evaluations, whilst DeepGA requires 520 evaluations. Competitive solutions can still be achieved through MODeepGA.

The variation of the classification error in MODeepGA is in the range $[0.0412, 0.3543]$ with the proposed encoding, and in the range $[0.0412, 0.0907]$ with the Wang encoding. The number of parameters variation with the proposed encoding is in the interval $[0.0132, 30.25]$ millions, whilst with the Wang encoding is in the interval $[0.0131, 2.97]$ millions. The Pareto Front from 50 generations of MODeepGA (Fig. 4.7) also shows that the variation of the classification error across solutions is small compared to that of the number of parameters. With this information, it is possible to establish this instance as a constrained optimization problem, as stated in Eq. (4.1b):

$$\min_{cnn} \quad NP(cnn) \tag{4.1a}$$

$$\text{subject to} \quad error(cnn) \leq 0.05, \tag{4.1b}$$

where cnn is an arbitrary CNN architecture, NP is the objective function of the number of parameters, and $error$ is the classification error as a constraint function. An error of 0.05 corresponds to an accuracy of 95%, which would be a competitive performance that could be of great help in the fight against COVID-19.

4.4 Chapter Summary

In this chapter, the Multi-Objective version of DeepGA has been introduced and discussed (MODeepGA). A series of 60 experiments, similar to those in the previous chapter, have been carried out in order to assess the advantages of using the proposed encoding or the Wang encoding. Multi-objective evaluation metrics, such as Hypervolume and Spacing, have been employed to measure the quality of the solutions. Furthermore, a Multi-Criteria Decision Making approach has been adopted to choose specific architectures from the set of

Pareto optimal solutions. MODeepGA has also been applied to the CXR image classification problem in search of lung diseases.

The experimental results, along with statistical tests, demonstrate that the proposed encoding helps producing a better Hypervolume performance in the final Pareto Fronts than the Wang encoding. In terms of Spacing, on the other hand, the Wang encoding discovered Pareto Fronts with a better distribution. In comparison with DeepGA, the best solution from the 30 executions of MODeepGA did not outperform the best single-objective solution. For this reason, three additional experiments were executed in order to verify how useful is to utilize Multi-Objective optimization to this particular problem instance. It has been empirically found that there is no competitive advantage of MODeepGA over DeepGA (when the total number of evaluations is the same), but in practice, it can be an appealing approach as fewer evaluations were required. Moreover, this problem can be formalized as a constrained optimization problem.

Part of the results obtained in this chapter were included into the conference paper at the Workshop of Neuroevolution at Work of the ACM Genetic and Evolutionary Computation Conference (GECCO) 2021. The further studies that were included in this chapter were performed under the supervision of Dr. Carlos Artemio Coello Coello, during a six-weeks research stay at the *Centro de Investigación y de Estudios Avanzados* (CINVESTAV-IPN).

CHAPTER 5

Conclusions

In this thesis, the automatic design of Convolutional Neural Networks architectures has been explored by means of Evolutionary Computation. As studied and reviewed in Chapter 2, Evolutionary and Swarm Intelligence algorithms are feasible methods to perform the architecture search of CNNs through optimization.

The encoding, which is the computational representation of available solutions in the search space, is of general interest in the field of Evolutionary Computation. Particularly, the design of encodings plays a very important role in the development of bio-inspired algorithms. This condition still holds for Neuroevolution, in which neural encodings determine not only the size of the search space, but also the nature of the potential architectures to be discovered. In spite of the importance of this aspect, the study of neural encodings has received almost no attention from the community, even though its meaningfulness has been clearly stated.

In this study, a specific type of neural encoding, called hybrid encodings, has been analyzed. Hybrid encodings combine elements from other encoding families, aiming to distribute the representation of CNNs among different sub-structures, and thus improving the search. Hybrid encodings have emerged in the recent literature on Neuroevolution, but little experimental evidence exists on their advantages and disadvantages. One notably important characteristic is the implicit complexity that certain encodings might have. This concern is important as: (a) some encodings might entail larger and more complex search spaces, which could hamper the performance of the search, (b) some encodings could be naturally biased towards neural architectures that are more costly than others, in terms of number of trainable parameters. Although Multi-Objective Neuroevolution has been tailored to find trade-offs between accuracy

and complexity (in image classification), it is still not clear how certain encodings benefit or affect the search.

To start with the analysis of neural encodings, a Neuroevolution framework, namely, DeepGA, has been proposed, which includes the following characteristics: (1) it is based on the Genetic Algorithm, and does not contain special components that could obscure the impact of the encoding, (2) it is able to utilize two different hybrid encodings based on blocks at the first level, and binary strings at the second level, and (3) it is adaptable to single-objective optimization with a linear aggregating function based on the accuracy and the number of parameters, and to multi-objective optimization, using both objectives simultaneously. As it has been mentioned, the main interest has been placed on the encodings for CNNs.

The proposed encoding consists on a less modular version of the so-called Wang encoding. The Wang encoding uses DenseBlocks, which are modules that contain several convolutional layers, in which their connectivity is determined by a binary string. The binary strings dictate how to perform the skip connections required to build a DenseNet-like architecture. The proposed encoding is similar in that a binary string represents the same connectivity patterns. However, simple blocks consisting on individual convolutional layers are used. The main rationale of this decision is that using DenseBlocks, which carry more layers and thus more parameters, biases the search towards more complex architectures.

The classification of lung conditions in chest X-ray images has been utilized as a case study to evaluate the impact of both encodings. As the application of CNNs becomes more pervasive in different settings, such as hospitals, designing more efficient and less costly models becomes more important. In view of the unlikely availability of high-performance computing hardware, e.g., GPUs, in hospitals and other medical backgrounds, more compact CNNs would be of an important benefit. Although several approaches to classify lung conditions, including COVID-19, have been published, these models rely on an excessive number of parameters.

Based on a series of experiments based on single- and multi-objective optimization, it has been found that, for this problem instance, both encodings yield similar numerical results, yet with statistically significant differences. In single-objective DeepGA, the proposed encoding

achieved a higher fitness function, which is due to a better accuracy. However, in terms of number of parameters, there was not a significant difference between encodings. In multi-objective DeepGA, the comparison was based on two specific performance measures; Hypervolume and Spacing. In terms of Hypervolume, the proposed encoding obtained a better position in the optimality of the Pareto Fronts (0.9468 ± 0.0411 against 0.9084 ± 0.0608, where a higher value corresponds to a better optimality), whilst in terms of Spacing, a better distribution was obtained with the Wang encoding (0.0517 ± 0.2817 against 0.0001 ± 0.0003, where a smaller value corresponds to a better diversity).

The multi-objective version of DeepGA was further explored in order to better understand how the architecture search becomes more competitive when more than one objective function is introduced. After a series of experiments, it was determined that this problem does not improve with multi-objective optimization. Moreover, it has been found that a possible problem statement can be a constrained optimization problem, in which the number of parameters becomes the main objective, and the accuracy is used as a quality constraint. Due to the *no free lunch theorems*, it is expected that the obtained results of this thesis extrapolate differently to other application problems. For this reason, further experiments are required and encouraged to extend the scope of DeepGA and to better understand hybrid encodings.

Future Work

As a future outlook of this research, more experiments as well a new proposals are encouraged. In the first place, DeepGA and MODeepGA are to be applied using benchmark datasets such as CIFAR-10, CIFAR-100, and Fashion-MNIST, and MNIST. The efficiency of DeepGA in a niche problem scenario has been demonstrated, thus, extending these results to larger-scale problems is the following step.

As mentioned before, the multi-objective version of DeepGA did not show a clear advantage over the single-objective version. Some possible additions to the framework are encouraged in order to better exploit the capabilities of Multi-Objective Optimization. First, an improved

sorting mechanism is to be included. In the Nondominated Sorting Genetic Algorithm (NSGA-II) (Deb et al. 2002), the population is sorted not only in terms of dominance, but also with respect to diversity. This mechanisms provides the means to ensure that the solutions are sufficiently apart from each other. Introducing this approach, would in essence transform DeepGA into an instance of NSGA-II for Neuroevolution of CNNs.

Secondly, a third objective function can be added, which could drive the Pareto Fronts towards better regions of the function space. To this end, it is also required to ask *which other aspect of a CNN can be optimized?*. The accuracy and the complexity of the networks have already been considered. Some other authors include objective functions such as the inference time or the number of floating point operations. Unfortunately, these objectives are closely related to the number of parameters of the CNNs. Another highly important aspect of CNNs has not been explored before; Explainability.

Explainability is a young field within Machine Learning, which deals with designing mechanisms through which the inference of Neural Networks can be understood by humans (Tjoa and Guan 2020). Defining novel ways in which the Explainability of a CNN could be quantitatively measured is a next step of this research. Furthermore, if Explainability emerges from certain architectures, it is hypothesized that it could also emerge from evolution. To the best of the author's knowledge, the only work of Neuroevolution of CNNs that has slightly dealt with Explainability is (Kobayashi and Nagao 2020). However, in that work Explainability was not introduced as an objective function, nor the framework was Multi-Objective Optimization. This path is an *open door* towards what could be very important innovations.

In the third place, it is expected that this thesis motivates the further study of other neural encodings, including different configurations of hybrid encodings. As no other formal study that evaluates the impact of encodings exists, further experiments would provide a very important insight to the field of Neuroevolution. Moreover, the design of novel *out of the ordinary* encodings is highly encouraged. Some interesting questions about encodings still remain; *How does modularity emerges more easily from certain encodings? Could this emergence be controlled? What are the foundational relationships withing encodings? Are different encodings instances from one generalized representation for CNNs?* It has

been discussed by Zador (2019) how the genome of mammals has a representation power fairly small in comparison to the number of neural wiring connections it encodes. Thus, more research in the field of Indirect Encodings is also encouraged in order to discover computational methods able to encode larger CNNs with less resources.

The encoding is the first and most important element in the design of an Evolutionary Algorithm. The evolution of computational brains is not the exception. Thus, there are hopes that the study on neural encodings bring important discoveries for years to come, not only in Neuroevolution, but also in the field of Deep Learning.

Bibliography

Aggarwal, C. C. (2018). *Neural Networks and Deep Learning. A Textbook.* 1st ed. Springer.

Ahmad, M. et al. (2020). "Image Classification Based on Automatic Neural Architecture Search Using Binary Crow Search Algorithm". In: *IEEE Access*, pp. 189891–189912. DOI: 10.1109/ACCESS.2020.3031599.

Ahmed, A. A., S. M. S. Darwish, and M. M. El-Sherbiny (2019). "A Novel Automatic CNN Architecture Design Approach Based on Genetic Algorithm". In: *A Novel Automatic CNN Architecture Design Approach Based on Genetic Algorithm*. Springer, pp. 473–482. DOI: 10.1007/978-3-030-31129-2_43.

Ai, T. et al. (2020). "Correlation of Chest CT and RT-PCR Testing for Coronavirus Disease 2019 (COVID-19) in China: A Report of 1014 Cases". In: *Radiology* 296.2. DOI: 10.1148/radiol.2020200642.

Akut, R. and S. Kulkami (2019). "NeuroEvolution : Using Genetic Algorithm for optimal design of Deep Learning models". In: *2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*. IEEE, pp. 1–6. DOI: 10.1109/ICECCT.2019.8869233.

Altan, A. and S. Karasu (2020). "Recognition of COVID-19 disease from X-ray images by hybrid model consisting of 2D curvelet transform, chaotic salp swarm algorithm and deep learning technique". In: *Chaos, Solitons & Fractals* 140. DOI: 10.1016/j.chaos.2020.110071.

Assuncao, F., J. Correira, et al. (2019). "Automatic Design of Artificial Neural Networks for Gamma-Ray Detection". In: *IEEE Access* 7, pp. 110531–110540. DOI: 10.1109/ACCESS.2019.2933947.

Assuncao, F., N. Lourenco, P. Machado, et al. (2018). “DENSER: deep evolutionary network structured representation”. In: *Genetic Programming and Evolvable Machines* 20.1, pp. 5–35. DOI: 10.1007/s10710-018-9339-y.

— (2019). “Fast DENSER: Efficient Deep NeuroEvolution”. In: *European Conference on Genetic Programming*. Springer, pp. 197–212. DOI: 10.1007/978-3-030-16670-0_13.

Assuncao, F., N. Lourenco, B. Ribeiro, et al. (2020). “Incremental Evolution and Development of Deep Artificial Neural Networks”. In: *European Conference on Genetic Programming*. Springer, pp. 35–51. DOI: 10.1007/978-3-030-44094-7_3.

Awad, N., N. Mallik, and F. Hutter (2020). “Differential Evolution for Neural Architecture Search”. In: *1st Workshop on Neural Architecture Search at International Conference on Learning Representations (ICLR)*.

Badan, F. and L. Sekanina (2019). “Optimizing Convolutional Neural Networks for Embedded Systems by Means of Neuroevolution”. In: *International Conference on Theory and Practice of Natural Computing*. Springer, pp. 109–121. DOI: 10.1007/978-3-030-34500-6_7.

Badrinarayanan, V., A. Kendall, and R. Cipolla (2017). “SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.12, pp. 2841–2495. DOI: 10.1109/TPAMI.2016.2644615.

Baker, J. E. (1985). “Adaptive Selection Methods for Genetic Algorithms”. In: *Proceedings of the 1st International Conference on Genetic Algorithms*. Association for Computing Machinery, pp. 101–111.

Baldeon-Calisto, M. and S. K. Lai-Yuen (2020). “AdaResU-Net: Multiobjective adaptive convolutional neural network for medical image segmentation”. In: *Neuroecomputing* 392. DOI: 10.1016/j.neucom.2019.01.110.

Baldominos, A., Y. Saez, and P. Isasi (Mar. 2018a). “Evolutionary convolutional neural networks: An application to handwriting recognition”. In: *Neurocomputing* 283.29, pp. 38–52. DOI: 10.1016/j.neucom.2017.12.049.

— (Apr. 2018b). “Evolutionary Design of Convolutional Neural Networks for Human Activity Recognition in Sensor-Rich Environments”. In: *Sensors* 18.4. DOI: 10.3390/s18041288.

— (Mar. 2019a). “Hybridizing Evolutionary Computation and Deep Neural Networks: An Approach to Handwriting Recognition Using Committees and Transfer Learning”. In: *Complexity* 2019. DOI: 10.1155/2019/2952304.

— (Mar. 2019b). “On the automated, evolutionary design of neural networks: past, present, and future”. In: *Neural Computing and Applications* 32, pp. 519–545. DOI: doi:10.1007/s00521-019-04160-6.

Baltruschat, I. M. et al. (2019). “Comparison of Deep Learning Approaches for Multi-Label Chest X-Ray Classification”. In: *Nature Scientific Reports* 9.6381. DOI: 10.1038/s41598-019-42294-8.

Barnes, D. K. et al. (2020). “A First Step Toward Incremental Evolution of Convolutional Neural Networks”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Association for Computing Machinery, pp. 115–116. DOI: 10.1145/3377929.3389916.

Beeching, N. J., T. E. Fletcher, and M. B. Beadsworth (2020). “Covid-19: testing times”. In: *British Medical Journal* 369. DOI: 10.1136/bmj.m1403.

Bhandari, D., C. A. Murthy, and S. K. Pal (1996). “Genetic Algorithm with Elitist Model and Its Convergence”. In: *Radiology* 10.6, pp. 731–747. DOI: 10.1142/S0218001496000438.

Bi, Y., B. Xue, and M. Zhang (2019). “An Evolutionary Deep Learning Approach Using Genetic Programming with Convolutional Operators for Image Classification”. In: *Proceedings of the 2019 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, pp. 3197–3204. DOI: 10.1109/CEC.2019.8790151.

Bibaeva, V. (2018). “Using Metaheuristics for Hyper-Parameter Optimization of Convolutional Neural Networks”. In: *2018 IEEE 28th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, pp. 1–6. DOI: 10.1109/MLSP.2018.8516989.

Bochinski, E., T. Senst, and T. Sikora (2017). “Hyper-parameter optimization for convolutional neural network committees based on evolutionary algorithms”. In: *2017 IEEE*

International Conference on Image Processing (ICIP). IEEE, pp. 3924–3928. DOI: 10.1109/ICIP.2017.8297018.

Boyd, S. and L. Vandenberghe (2004). *Convex Optimization*. 1st ed. Cambridge University Press.

Brindle, A. (1981). "Genetic algorithms for function optimization". Edmonton, Canada: Department of Computer Science, University of Alberta. DOI: 10.7939/R3FB4WS2W.

Broni-Bediako, C. et al. (2020). "Evolutionary NAS with Gene Expression Programming of Cellular Encoding". In: *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. DOI: 10.1109/SSCI47803.2020.9308346.

Brunese, L. et al. (2020). "Explainable Deep Learning for Pulmonary Disease and Coronavirus COVID-19 Detection from X-rays". In: *Computer Methods and Programs in Biomedicine* 196. DOI: 10.1016/j.cmpb.2020.105608.

Byla, E. and W. Pang (2019). "DeepSwarm: Optimising Convolutional Neural Networks Using Swarm Intelligence". In: *Proceedings of UK Workshop on Computational Intelligence*. Springer, pp. 119–130. DOI: 10.1007/978-3-030-29933-0_10.

Calimeri, F. et al. (2017). "Biomedical Data Augmentation Using Generative Adversarial Neural Networks". In: *International Conference on Artificial Neural Networks*. Ed. by A. Lintas et al. Vol. 10614. Springer, pp. 626–634. DOI: 10.1007/978-3-319-68612-7_71.

Cao, Y. (2021). *Hypervolume Indicator*. URL: https://www.mathworks.com/matlabcentral/fileexchange/19651-hypervolume-indicator.

Cetto, T. et al. (2019). "Size/Accuracy Trade-Off in Convolutional Neural Networks: An Evolutionary Approach". In: *Proceedings of the INNS Big Data and Deep Learning Conference*. Ed. by Springer, pp. 17–26. DOI: 10.1007/978-3-030-16841-4_3.

Chen, X., Y. Sun, et al. (2021). "Evolving Deep Convolutional Variational Autoencoders for Image Classification". In: *IEEE Transactions on Evolutionary Computation*. DOI: 10.1109/TEVC.2020.3047220.

Chen, Y., G. Meng, et al. (2019). "RENAS: Reinforced Evolutionary Neural Architecture Search". In: *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, pp. 4782–4791. DOI: 10.1109/CVPR.2019.00492.

Chen, Y., T. Yang, et al. (2019). "DetNAS: Backbone Search for Object Detection". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., pp. 6642–6652. URL: http://papers.nips.cc/paper/8890-detnas-backbone-search-for-object-detection.pdf.

Chen, Z., Y. Zhou, and Z. Huang (2019). "Auto-creation of Effective Neural Network Architecture by Evolutionary Algorithm and ResNet for Image Classification*". In: *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. IEEE, pp. 3895–3900.

Chiu, W.-Y., G. G. Yen, and T.-K. Juan (2016). "Minimum Manhattan Distance Approach to Multiple Criteria Decision Making in Multiobjective Optimization Problems". In: *IEEE Transactions on Evolutionary Computation* 20.6, pp. 972–985. DOI: 10.1109/TEVC.2016.2564158.

Chollet, F. (2017). "Xception: Deep Learning with Depthwise Separable Convolutions". In: *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE.

Chu, X., B. Zhang, and R. Xu (2020). "Multi-objective Reinforced Evolution in Mobile Neural Architecture Search". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, pp. 99–113. DOI: 10.1007/978-3-030-66823-5_6.

Chung, H. and K.-s. Shin (2020). "Genetic algorithm-optimized multi-channel convolutional neural network for stock market prediction". In: *Neural Computing and Applications* 32, pp. 7897–7914. DOI: 10.1007/s00521-019-04236-3.

Civit-Masot, J. et al. (2020). "Deep Learning System for COVID-19 Diagnosis Aid Using X-ray Pulmonary Images". In: *Applied Sciences* 10.13. DOI: 10.3390/app10134640.

Cohen, J. P. et al. (2020). "COVID-19 Image Data Collection: Prospective Predictions Are the Future". In: *arXiv 2006.11988*. URL: https://github.com/ieee8023/covid-chestxray-dataset.

Dahal, B. and J. Zhan (2020). "Effective Mutation and Recombination for Evolving Convolutional Networks". In: *Proceedings of the 3rd International Conference on Applications of Intelligent Systems*. Association for Computing Machinery. DOI: 10.1145/3378184.3378187.

Dahou, A. et al. (2019). “Arabic Sentiment Classification Using Convolutional Neural Network and Differential Evolution Algorithm”. In: *Computational Intelligence and Neuroscience* 2019. DOI: 10.1155/2019/2537689.

Deb, K. (2007). “Current trends in evolutionary multi-objective optimization”. In: *International Journal for Simulation and Multidisciplinary Design Optimization* 1.1, pp. 1–8. DOI: 10.1051/ijsmdo:2007001.

Deb, K. et al. (2002). “A fast and elitist multiobjective genetic algorithm: NSGA-II”. In: *IEEE Transactions on Evolutionary Computation* 6.2, pp. 183–197. DOI: 10.1109/4235.996017.

Desell, T. (2017). “Large scale evolution of convolutional neural networks using volunteer computing”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. Association for Computing Machinery, pp. 127–128. DOI: 10.1145/3067695.3076002.

Dong, J.-D., A.-C. Juan, et al. (2018). “PPP-Net: Platform-aware Progressive Search for Pareto-optimal Neural Architectures”. In: *International Conference on Learning Representations (ICLR) Workshop*.

Dong, J., L. Zhang, et al. (2020). “A Memetic Algorithm for Evolving Deep Convolutional Neural Network in Image Classification”. In: *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 2663–2669. DOI: 10.1109/SSCI47803.2020.9308162.

Drezner, Z. and T. D. Drezner (July 2020). “Biologically Inspired Parent Selection in Genetic Algorithms”. In: *Annals of Operations Research* 287, pp. 161–183. DOI: 10.1007/s10479-019-03343-7.

Eiben, A. and J. Smith (2015a). *Introduction to Evolutionary Computing*. 2nd ed. Springer.

Eiben, A. E. and J. Smith (2015b). “From evolutionary computation to the evolution of things”. In: *Nature* 521, pp. 476–482. DOI: 10.1038/nature14544.

Elsken, T., J. H. Metzen, and F. Hutter (2019). “Neural Architecture Search: A Survey”. In: *Journal of Machine Learning Research* 20, pp. 1–21.

Evans, B. et al. (2018). “Evolutionary Deep Learning: A Genetic Programming Approach to Image Classification”. In: *Proceedings of the 2018 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, pp. 1–6. DOI: 10.1109/CEC.2018.8477933.

Faria, M. P. P., R. M. S. Julia, and L. B. P. Tomaz (2019). “Improving FIFA Player Agents Decision-Making Architectures Based on Convolutional Neural Networks Through Evolutionary Techniques”. In: *Brazilian Conference on Intelligent Systems (BRACIS)*. Springer, pp. 371–386. DOI: 10.1007/978-3-030-61377-8_26.

Fei-Fei, L., R. Krishna, and D. Xu (2020). *Convolutional Neural Networks (CNNs/ConvNets)*. URL: https://cs231n.github.io/convolutional-networks/.

Fernandes, F. E. and G. G. Yen (2021). “Automatic Searching and Pruning of Deep Neural Networks for Medical Imaging Diagnostic”. In: *IEEE Transactions on Neural Networks and Learning Systems*. DOI: 10.1109/TNNLS.2020.3027308.

Fernando, C. et al. (2016). “Convolution by Evolution: Differentiable Pattern Producing Networks”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Association for Computing Machinery, pp. 109–116. DOI: 10.1145/2908812.2908890.

Fielding, B. and L. Zhang (2018). “Evolving Image Classification Architectures With Enhanced Particle Swarm Optimisation”. In: *IEEE Access* 6, pp. 68560–68575. DOI: 10.1109/ACCESS.2018.2880416.

Fujino, S. et al. (2019). “Evolutionary deep learning based on deep convolutional neural network for anime storyboard recognition”. In: *Neurocomputing* 338, pp. 393–398. DOI: 10.1016/j.neucom.2018.05.124.

Galván, E. and P. Mooney (June 2020). “Neuroevolution in Deep Neural Networks: Current Trends and Future Challenges”. In: *ArXiv Preprints*. URL: https://arxiv.org/pdf/2006.05415.pdf.

Gibbons, J. D. and S. Chakraborti (2003). *Nonparametric Statistical Inference*. 4th ed. Marcel Dekker.

Girshick, R. (2015). “Fast R-CNN”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. IEEE. DOI: 10.1109/ICCV.2015.169.

Goldberg, D. E. and K. Deb (1991). "A Comparative Analysis of Selection Schemes Used in Genetic Algorithms". In: *Foundations of Genetic Algorithms* 1, pp. 69–93. DOI: 10.1016/B978-0-08-050684-5.50008-2.

Gonzalez, R. C. and R. E. Woods (2017). *Digital Image Processing*. 4th ed. Pearson.

Gottapu, R. D. and C. H. Dagli (2020). "Efficient Architecture Search for Deep Neural Networks". In: *Procedia Computer Science* 168, pp. 19–25. DOI: 10.1016/j.procs.2020.02.246.

Hadjiivanov, A. and A. Blair (2019). "Epigenetic evolution of deep convolutional models". In: *2019 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, pp. 1478–1486.

Hahnloser, R. H. et al. (May 1998). "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit". In: *Nature* 405, pp. 947–951. DOI: 10.1038/35016072.

Hajewski, J., S. Oliveira, and X. Xing (2020). "Evolving deep autoencoders". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Association for Computing Machinery, pp. 123–124. DOI: 10.1145/3377929.3390011.

Han, Y., J. Kim, and K. Lee (Nov. 2016). "Deep Convolutional Neural Networks for Predominant Instrument Recognition in Polyphonic Music". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 25.1, pp. 208–221. DOI: 10.1109/TASLP.2016.2632307.

Hanin, B. (2018). "Which Neural Net Architectures Give Rise to Exploding and Vanishing Gradients?" In: *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*. Vol. 31. Curran Associates, Inc., pp. 582–591.

Hassanzadeh, T., D. Essam, and R. Sarker (2020a). "2D to 3D Evolutionary Deep Convolutional Neural Networks for Medical Image Segmentation". In: *IEEE Access* 8, pp. 212298–212314. DOI: 10.1109/ACCESS.2020.3039496.

— (2020b). "EvoU-Net: an evolutionary deep fully convolutional neural network for medical image segmentation". In: *Proceedings of the 35th Annual ACM Symposium on Applied Computing*. Association for Computing Machinery. DOI: 10.1145/3341105.3373856.

— (2020c). “Proceedings of the 35th Annual ACM Symposium on Applied Computing”. In: *Proceedings of the International Conference on Digital Image Computing: Techniques and Applications (DICTA)*.

— (2021). “2D to 3D Evolutionary Deep Convolutional Neural Networks for Medical Image Segmentation”. In: *IEEE Transactions on Medical Imaging* 40.2, pp. 712–721. DOI: 10.1109/TMI.2020.3035555.

He, K. et al. (2016). “Deep Residual Learning for Image Recognition”. In: *2016 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. 1st ed. University of Michigan Press.

Homburg, J. D. et al. (2019). “Constraint Exploration of Convolutional Network Architectures with Neuroevolution”. In: *International Work-Conference on Artificial Neural Networks*. Springer, pp. 735–746. DOI: 10.1007/978-3-030-20518-8_61.

Hu, M. et al. (2020). “APENAS: An Asynchronous Parallel Evolution Based Multi-objective Neural Architecture Search”. In: *2020 IEEE International Conference on Parallel and Distributed Processing with Applications, Big Data and Cloud Computing, Sustainable Computing and Communications, Social Computing and Networking*. IEEE.

Huang, G. et al. (2017). “Densely Connected Convolutional Networks”. In: *2017 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, pp. 4700–4708.

Ioffe, S. and C. Szegedy (2015). “Batch normalization: accelerating deep network training by reducing internal covariate shift”. In: *Proceedings of the 32nd International Conference on Machine Learning*. Vol. 37. JMLR, pp. 448–456.

Irwin-Harris, W. et al. (2019). “A Graph-Based Encoding for Evolutionary Convolutional Neural Network Architecture Design”. In: *2019 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, pp. 546–553. DOI: 10.1109/CEC.2019.8790093.

Islam, M. Z., M. M. Islam, and A. Asraf (2020). “A combined deep CNN-LSTM network for the detection of novel coronavirus (COVID-19) using X-ray images”. In: *Informatics in Medicine Unlocked* 20. DOI: 10.1016/j.imu.2020.100412.

Jalali, S. M. J. et al. (2019). “Optimal Autonomous Driving Through Deep Imitation Learning and Neuroevolution”. In: *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. IEEE. DOI: 10.1109/SMC.2019.8914582.

Javaheripi, M. et al. (2020). “GeneCAI: genetic evolution for acquiring compact AI”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Association for Computing Machinery, pp. 350–358. DOI: 10.1145/3377930.3390226.

Jiang, J. et al. (Mar. 2020). “Efficient network architecture search via multiobjective particle swarm optimization based on decomposition”. In: *Neural Networks* 123. DOI: 10.1016/j.neunet.2019.12.005.

Johner, F. M. and J. Wassner (2019). “Efficient Evolutionary Architecture Search for CNN Optimization on GTSRB”. In: *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*. IEEE. DOI: 10.1109/ICMLA.2019.00018.

Jong, K. A. D. (1975). “An analysis of the behavior of a class of genetic adaptive systems”. University of Michigan.

Jr., F. E. F. and G. G. Yen (2021). “Pruning Deep Convolutional Neural Networks Architectures with Evolution Strategy”. In: *Information Sciences* 552, pp. 29–47. DOI: 10.1016/j.ins.2020.11.009.

Junior, F. E. F. and G. G. Yen (2019a). “Particle swarm optimization of deep neural networks architectures for image classification”. In: *Swarm and Evolutionary Computation* 49, pp. 62–74. DOI: 10.1016/j.swevo.2019.05.010.

— (2019b). “Pruning Deep Convolutional Neural Networks Architectures with Evolution Strategy”. In: *Information Sciences* 552, pp. 29–47. DOI: 10.1016/j.ins.2020.11.009.

Kang, D. and C. W. Ahn (2020). “Efficient Neural Network Space with Genetic Search”. In: *Proceedings of the International Conference on Bio-Inspired Computing: Theories and Applications*. Springer. DOI: 10.1007/978-981-15-3415-7_54.

Kanne, J. P. et al. (2020). “Essentials for Radiologists on COVID-19: An Update–Radiology Scientific Expert Panel”. In: *Radiology* 296.2. DOI: 10.1148/radiol.2020200527.

Keshari, R. (2017). *DEvol: Automated deep neural network design via genetic programming*. URL: https://github.com/RohitKeshari/devol.

Khan, A. et al. (2020). “A survey of the recent architectures of deep convolutional neural networks”. In: *Artificial Intelligence Review* 53, pp. 5455–5516. DOI: 10.1007/s10462-020-09825-6.

Kim, Y. (2014). “Convolutional Neural Networks for Sentence Classification”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, pp. 1746–1751.

Kizhevsky, A., I. Sutskever, and G. E. Hinton (2012). “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., pp. 1097–1105. DOI: doi:10.1109/WACV.2015.71. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Kobayashi, M., S. Arai, and T. Nagao (2020). “Evolutionary Generative Contribution Mappings”. In: *2010 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. IEEE. DOI: 10.1109/SMC42975.2020.9283014.

Kobayashi, M. and T. Nagao (2020). “An evolution-based approach for efficient differentiable architecture search”. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*. Association for Computing Machinery, pp. 131–132. DOI: 10.1145/3377929.3390003.

Kotyan, S. and D. V. Vargas (2020). “Towards evolving robust neural architectures to defend from adversarial attacks”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Association for Computing Machinery, pp. 135–136. DOI: 10.1145/3377929.3389962.

Kwasigroch, A., M. Grochowski, and A. Mikolajczyk (2020). “Neural Architecture Search for Skin Lesion Classification”. In: *IEEE Access*. DOI: 10.1109/ACCESS.2020.2964424.

LeCun, Y., Y. Bengio, and G. Hinton (2015). “Deep Learning”. In: *Nature* 521, pp. 436–444. DOI: 10.1038/nature14539.

LeCun, Y., L. Bottou, et al. (May 1998). “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11, pp. 2278–2324. DOI: 10.1109/5.726791.

Lee, S. et al. (2021). “Genetic Algorithm Based Deep Learning Neural Network Structure and Hyperparameter Optimization”. In: *Applied Sciences* 11.2. DOI: 10.3390/app11020744.

Li, Y. et al. (Oct. 2019). “Evolving deep convolutional neural networks by quantum behaved particle swarm optimization with binary encoding for image classification”. In: *Neurocomputing* 362.14, pp. 156–165. DOI: 10.1016/j.neucom.2019.07.026.

Liang, J. et al. (2019). “Evolutionary neural AutoML for deep learning”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Association for Computing Machinery, pp. 401–409. DOI: 10.1145/3321707.3321721.

Lillicrap, T. P. et al. (Apr. 2020). “Backpropagation and the brain”. In: *Nature Reviews Neuroscience* 21, pp. 335–346. DOI: 10.1038/s41583-020-0277-3.

Lima, R. H., A. T. Pozo, and R. Santana (2019). “Automatic Design of Convolutional Neural Networks using Grammatical Evolution”. In: *2019 8th Brazilian Conference on Intelligent Systems (BRACIS)*. IEEE, pp. 329–334. DOI: 10.1109/BRACIS.2019.00065.

Litzinger, S., A. Klos, and W. Schiffmann (2018). “Compute-Efficient Neural Network Architecture Optimization by a Genetic Algorithm”. In: *Proceedings of the International Conference on Artificial Neural Networks*. Springer, pp. 886–893. DOI: 10.1007/978-3-030-30484-3_32.

Liu, H., K. Simonyan, et al. (2018). “Hierarchical Representations for Efficient Architectures Search”. In: *Proceedings of the International Conference on Learning Representation*.

Liu, J., S. Zhou, et al. (2020). “Block Proposal Neural Architecture Search”. In: *IEEE Transactions on Image Processing* 30, pp. 15–25. DOI: 10.1109/TIP.2020.3028288.

Liu, P., M. D. E. Basha, et al. (2019). “Deep Evolutionary Networks with Expedited Genetic Algorithms for Medical Image Denoising”. In: *Medical Image Analysis* 54, pp. 306–315. DOI: 10.1016/j.media.2019.03.004.

Loni, M., S. Sinaei, et al. (2020). “DeepMaker: A multi-objective optimization framework for deep neural networks in embedded systems”. In: *Microprocessors and Microsystems*. DOI: 10.1016/j.micpro.2020.102989.

Loni, M., A. Zoijodi, et al. (2020). “DenseDisp: Resource-Aware Disparity Map Estimation by Compressing Siamese Neural Architecture”. In: *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. DOI: 10.1109/CEC48606.2020.9185611.

Loni, M., A. Zoljodi, et al. (2019). “NeuroPower: Designing Energy Efficient Convolutional Neural Network Architecture for Embedded Systems”. In: *International Conference on Artificial Neural Networks (ICANN)*. Springer, pp. 208–222. DOI: 10.1007/978-3-030-30487-4_17.

López-Ibáñez, M. et al. (2016). “The irace package: Iterated Racing for Automatic Algorithm Configuration”. In: *Operations Research Perspectives* 3, pp. 43–58. DOI: 10.1016/j.orp.2016.09.002.

Lorenzo, P. R. and J. Nalepa (2018). “Memetic evolution of deep neural networks”. In: *Proceedings of the 2018 Genetic and Evolutionary Computation Conference*. Association for Computing Machinery, pp. 505–512. DOI: 10.1145/3205455.3205631.

Lu, Z., K. Deb, et al. (2019). “NSGANetV2: Evolutionary Multi-objective Surrogate-Assisted Neural Architecture Search”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, pp. 35–51. DOI: 10.1007/978-3-030-58452-8_3.

Lu, Z., G. Sreekumar, et al. (May 2020). “Neural Architecture Transfer”. In: *ArXiv Preprint*. URL: https://arxiv.org/abs/2005.05859.

Lu, Z., I. Whalen, V. Boddeti, et al. (2019). “NSGA-Net: Neural Architecture Search using Multi-Objective Genetic Algorithm”. In: *ArXiv Preprint*. URL: https://arxiv.org/abs/1810.03522.

Lu, Z., I. Whalen, Y. Dhebar, et al. (2020). “Multi-Objective Evolutionary Design of Deep Convolutional Neural Networks for Image Classification”. In: *IEEE Transactions on Evolutionary Computation*. DOI: 10.1109/TEVC.2020.3024708.

Ma, A. et al. (2021). "SceneNet: Remote sensing scene classification deep learning network using multi-objective neural evolution architecture search". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 172, pp. 171–188. DOI: 10.1016/j.isprsjprs.2020.11.025.

Martin, A. et al. (2017). "Evolving Deep Neural Networks architectures for Android malware classification". In: *2017 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, pp. 1659–1666. DOI: 10.1109/CEC.2017.7969501.

Martín, A., R. Lara-Cabrera, et al. (2018). "EvoDeep: A new evolutionary approach for automatic Deep Neural Networks parametrisation". In: *Journal of Parallel and Distributed Computing* 117, pp. 180–191. DOI: 10.1016/j.jpdc.2017.09.006.

Martín, A., V. M. Vargas, et al. (2020). "Optimising Convolutional Neural Networks using a Hybrid Statistically-driven Coral Reef Optimisation algorithm". In: *Applied Soft Computing* 90. DOI: 10.1016/j.asoc.2020.106144.

Maziarz, K. et al. (2019). "Evolutionary-Neural Hybrid Agents for Architecture Search". In: *Proceedings of the International Conference on Learning Representations (ICLR)*.

McGhie, A., B. Xue, and M. Zhang (2020). "GPCNN: Evolving Convolutional Neural Networks using Genetic Programming". In: *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. DOI: 10.1109/SSCI47803.2020.9308390.

Mezura-Montes, E. and C. A. C. Coello (2008). "An empirical study about the usefulness of evolution strategies to solve constrained optimization problems". In: *International Journal on General Systems* 37.4, pp. 443–473. DOI: 10.1080/03081070701303470.

Miikkulainen, R. et al. (2019). "Evolving Deep Neural Networks". In: *ArXiv Preprint*. URL: https://arxiv.org/abs/1703.00548.

Mirjalili, S. et al. (2019). *Nature-Inspired Optimizers. Theories, Literature Reviews and Applications*. 1st ed. Springer.

Mitschke, N. et al. (2018). "Gradient Based Evolution to Optimize the Structure of Convolutional Neural Networks". In: *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE, pp. 3438–3442. DOI: 10.1145/2834892.2834896.

Montana, D. J. and L. Davis (1989). “Training feedforward neural networks using genetic algorithms”. In: *Proceedings of the 11th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann Publishers Inc., pp. 762–767.

Mooney, P. (2017). *Chest X-Ray Images (Pneumonia)*. URL: https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia.

Mostafa, S. S. et al. (2020). “Multi-Objective Hyperparameter Optimization of Convolutional Neural Network for Obstructive Sleep Apnea Detection”. In: *IEEE Access* 8, pp. 129586–129599. DOI: 10.1109/ACCESS.2020.3009149.

N. Kumar, N. N. D. ad, V. Kumar, and D. Singh (2020). “Deep Learning System for COVID-19 Diagnosis Aid Using X-ray Pulmonary Images”. In: *IRBM*. DOI: 10.1016/j.irbm.2020.07.001.

O’Neill, D., B. Xue, and M. Zhang (2019). “The Evolution of Adjacency Matrices for Sparsity of Connection in DenseNets”. In: *2019 International Conference on Image and Vision Computing New Zealand (IVCNZ)*. IEEE. DOI: 10.1109/IVCNZ48456.2019.8961027.

— (2020). “Neural architecture search for sparse DenseNets with dynamic compression”. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. Association for Computing Machinery, pp. 386–394. DOI: 10.1145/3377930.3390178.

O’Neill, M. and C. Ryan (Aug. 2001). “Grammatical Evolution”. In: *IEEE Transactions on Evolutionary Computation* 5.4, pp. 349–358. DOI: 10.1109/4235.942529.

Oh, B. K. and K. Kim (2021). “Multi-Objective Optimization Method to Search for the Optimal Convolutional Neural Network Architecture for Long-Term Structural Health Monitoring”. In: *IEEE Access*. DOI: 10.1109/ACCESS.2021.3057687.

Oh, Y., S. Park, and J. C. Ye (2020). “Deep Learning COVID-19 Features on CXR using Limited Training Data Sets”. In: *IEEE Transactions on Medical Imaging* 39.8, pp. 2688–2700. DOI: 10.1109/TMI.2020.2993291.

Operiano, K. R. G., H. Iba, and W. Pora (2020). “Neuroevolution Architecture Backbone for X-ray Object Detection”. In: *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. DOI: 10.1109/SSCI47803.2020.9308453.

Oquab, M. et al. (2014). “Learning and Transferring Mid-Level Image Representations using Convolutional Neural Networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, pp. 1717–1724.

Ozturk, T. et al. (2020). “Automated detection of COVID-19 cases using deep neural networks with X-ray images”. In: *Computers in Biology and Medicine* 121. DOI: 10.1016/j.compbiomed.2020.103792.

Peto, J. (2020). “Covid-19 mass testing facilities could end the epidemic rapidly”. In: *British Medical Journal* 368. DOI: 10.1136/bmj.m1163.

Prellberg, J. and O. Kramer (2018). “Lamarckian Evolution of Convolutional Neural Networks”. In: *International Conference on Parallel Problems Solving from Nature*. Springer, pp. 424–435. DOI: 10.1007/978-3-319-99259-4_34.

Qu, X., J. Wang, and J. Xiao (2020). “Evolutionary Algorithm Enhanced Neural Architecture Search for Text-Independent Speaker Verification”. In: *INTERSPEECH 2020*. ISCA.

Rahimzadeh, M. and A. Attar (2020). “A modified deep convolutional neural network for detecting COVID-19 and pneumonia from chest X-ray images based on the concatenation of Xception and ResNet50V2”. In: *Informatics and Medicine Unlocked* 19. DOI: 10.1016/j.imu.2020.100360.

Rajagopal, A. et al. (2020). “A Deep Learning Model Based on Multi-Objective Particle Swarm Optimization for Scene Classification in Unmanned Aerial Vehicles”. In: *IEEE Access* 8, pp. 135383–135393. DOI: 10.1109/ACCESS.2020.3011502.

Ramachandran, P., B. Zoph, and Q. V. Le (2017). “Searching for Activation Functions”. In: *ArXiV Preprints*. URL: https://arxiv.org/abs/1710.05941.

Rawat, W. and Z. Wang (2017). “Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review”. In: *Neural Computation* 29.9, pp. 2352–2449. DOI: 10.1162/neco_a_00990.

Real, E., A. Aggarwal, et al. (2019). “Regularized Evolution for Image Classifier Architecture Search”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. AAAI. DOI: 10.1609/aaai.v33i01.33014780.

Real, E., S. Moore, et al. (2017). “Large-scale evolution of image classifiers”. In: *Proceedings of the 34th International Conference on Machine Learning*. Vol. 70. JMLR, pp. 2902–2911. DOI: 10.5555/3305890.3305981.

Redmon, J. et al. (2017). “You Only Look Once: Unified, Real-Time Object Detection”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 9351. IEEE, pp. 779–788.

Ren, J. et al. (2019). “EIGEN: Ecologically-Inspired GENetic Approach for Neural Network Structure Searching From Scratch”. In: *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, pp. 9051–9060. DOI: 10.1109/CVPR.2019.00927.

Rikhtegar, A., M. Pooyan, and M. T. Manzuri-Shalmani (2016). “Genetic algorithm-optimised structure of convolutional neural network for face recognition applications”. In: *IET Computer Vision* 10.6, pp. 559–566. DOI: 10.1049/iet-cvi.2015.0037.

Ronneberger, O., P. Fischer, and T. Brox (2015). “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2015*. Vol. 9351. Springer, pp. 6642–6652. DOI: 10.1007/978-3-319-24574-4_28.

Saltori, C. et al. (2019). “Regularized Evolutionary Algorithm for Dynamic Neural Topology Search”. In: *Proceedings of the International Conference on Image Analyisis and Processing*. Ed. by Springer, pp. 219–230. DOI: 10.1007/978-3-030-30642-7_20.

Santurkar, S. et al. (2018). “How Does Batch Normalization Help Optimization?” In: *Advances in Neural Information Processing Systems 31*. Curran Associates, Inc., pp. 2483–2493. URL: http://papers.nips.cc/paper/7515-how-does-batch-normalization-help-optimization.pdf.

Sapra, D. and A. D. Pimentel (2020). “An evolutionary optimization algorithm for gradually saturating objective functions”. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. Association for Computing Machinery, pp. 886–893. DOI: 10.1145/3377930.3389834.

Shang, K. et al. (2020). “A Survey on the Hypervolume Indicator in Evolutionary Multi-objective Optimization”. In: *IEEE Transactions on Evolutionary Computation*. DOI: 10.1109/TEVC.2020.3013290.

Song, D. et al. (2020). “Efficient Residual Dense Block Search for Image Super-Resolution”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34, pp. 12007–12014. DOI: 10.1609/aaai.v34i07.6877.

Srivastava, N. et al. (2014). “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15.56, pp. 1929–1958.

Stanley, K. O. (2004). “Efficient Evolution of Neural Networks Through Complexification”. Austin, TX: Department of Computer Sciences, The University of Texas at Austin.

Stanley, K. O., J. Clune, et al. (2019). “Designing neural networks through neuroevolution”. In: *Nature Machine Intelligence* 1.1, pp. 24–35.

Stanley, K. O. and R. Miikkulainen (May 2002). “Evolving Neural Networks Through Augmenting Topologies”. In: *Evolutionary Computation* 10.2, pp. 99–127. DOI: 0.1162/106365602320169811.

Strumberger, I. et al. (2019). “Designing Convolutional Neural Network Architecture by the Firefly Algorithm”. In: *2019 International Young Engineers Forum (YEF-ECE)*. IEEE, pp. 59–65. DOI: 10.1109/YEF-ECE.2019.8740818.

Su, B., R. Li, and H. Zhang (2020). “Evolving Deep Convolutional Neural Network for Intrusion Detection Based on NEAT”. In: *2020 23rd International Symposium on Wireless Personal Multimedia Communications (WPMC)*. IEEE. DOI: 10.1109/WPMC50192.2020.9309451.

Suganuma, M., M. Kobayashi, et al. (2020). “Evolution of Deep Convolutional Neural Networks Using Cartesian Genetic Programming”. In: *Evolutionary Computation* 28.1, pp. 141–163. DOI: 10.1162/evco_a_00253.

Suganuma, M., S. Shinichi, and T. Nagao (2017). “A genetic programming approach to designing convolutional neural network architectures”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Association for Computing Machinery, pp. 497–504. DOI: 10.1145/3071178.3071229.

Sun, Y., H. Wang, et al. (2020). “Surrogate-Assisted Evolutionary Deep Learning Using an End-to-End Random Forest-Based Performance Predictor”. In: *IEEE Transactions on Evolutionary Computation*, pp. 350–364. DOI: 10.1109/TEVC.2019.2924461.

Sun, Y., B. Xue, et al. (2019a). “A Particle Swarm Optimization-Based Flexible Convolutional Autoencoder for Image Classification”. In: *IEEE Transactions on Neural Networks and Learning Systems* 30.8, pp. 2295–2309. DOI: 10.1109/TNNLS.2018.2881143.

— (2019b). “Completely Automated CNN Architecture Design Based on Blocks”. In: *IEEE Transactions on Neural Networks and Learning Systems* 31.4. DOI: 10.1109/TNNLS.2019.2919608.

— (2019c). “Evolving Deep Convolutional Neural Networks for Image Classification”. In: *IEEE Transactions on Evolutionary Computation* 24.2, pp. 394–407. DOI: 10.1109/TEVC.2019.2916183.

— (2020). “Automatically Designing CNN Architectures Using Genetic Algorithm for Image Classification”. In: *IEEE Transactions on Cybernetics*, pp. 1–5. DOI: 10.1109/TCYB.2020.2983860.

Tabik, S. et al. (2020). *COVIDGR dataset and COVID-SDNet methodology for predicting COVID-19 based on Chest X-Ray images*. arXiv: 2006.01409 [eess.IV].

Taghanaki, S. A. et al. (2020). “Deep semantic segmentation of natural and medical images: a review”. In: *Artificial Intelligence Review*. DOI: s10462-020-09854-1.

Tan, H. et al. (2020). “Efficient Evolutionary Neural Architecture Search (NAS) by Modular Inheritable Crossover”. In: *International Conference on Bio-Inspired Computing: Theories and Applications*. Springer, pp. 761–769. DOI: 10.1007/978-981-15-3425-6_61.

Tang, H. et al. (2020). “Searching Efficient 3D Architectures with Sparse Point-Voxel Convolution”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, pp. 685–702. DOI: 10.1007/978-3-030-58604-1_41.

Tjoa, E. and C. Guan (2020). “A Survey on Explainable Artificial Intelligence (XAI): Toward Medical XAI”. In: *IEEE Transactions on Neural Networks and Learning Systems*. DOI: 10.1109/TNNLS.2020.3027314.

Togacar, M., B. Ergen, and Z. Comert (2020). "COVID-19 detection using deep learning models to exploit Social Mimic Optimization and structured chest X-ray images using fuzzy color and stacking approaches". In: *Computers in Biology and Medicine* 121. DOI: 10.1016/j.compbiomed.2020.103805.

Ucar, F. and D. Korkmaz (2020). "COVIDiagnosis-Net: Deep Bayes-SqueezeNet based diagnosis of the coronavirus disease 2019 (COVID-19) from X-ray images". In: *Medical Hypotheses* 140. DOI: 10.1016/j.mehy.2020.109761.

Vargas-Hákim, G.-A., E. Mezura-Montes, and E. Galván (2020). "Evolutionary Multi-Objective Energy Production Optimization: An Empirical Comparison". In: *Mathematical and Computational Applications* 25.2. DOI: 10.3390/mca25020032.

Verbancsics, P. and J. Harguess (2013). "Generative NeuroEvolution for Deep Learning". In: *ArXiv Preprint*. URL: https://arxiv.org/abs/1312.5355.

Vidnerová, P. and R. Neruda (2020). "Multi-objective Evolution for Deep Neural Network Architecture Search". In: *Proceedings of the International Conference on Neural Information Processing*. Springer, pp. 270–281. DOI: 10.1007/978-3-030-63836-8_23.

Vidnerová, P., Š. Procházka, and R. Neruda (2020). "Multi-objective Evolution for Convolutional Neural Network Architecture Search". In: *Proceedings of the International Conference on Artificial Intelligence and Soft Computing*. Springer, pp. 261–270. DOI: 10.1007/978-3-030-61401-0_25.

Wang, B., Y. Sun, et al. (2018a). "A Hybrid Differential Evolution Approach to Designing Deep Convolutional Neural Networks for Image Classification". In: *Australasian Joint Conference on Artificial Intelligence*. Springer, pp. 237–250. DOI: 10.1007/978-3-030-03991-2_24.

— (2018b). "Evolving Deep Convolutional Neural Networks by Variable-Length Particle Swarm Optimization for Image Classification". In: *2018 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, pp. 1–8. DOI: 10.1109/CEC.2018.8477735.

— (2019a). "Evolving Deep Neural Networks by Multi-Objective Particle Swarm Optimization for Image Classification". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. Association for Computing Machinery, pp. 490–498. DOI: 10.1145/3321707.3321735.

Wang, B., B. Xue, and M. Zhang (2019). “Particle Swarm Optimisation for Evolving Deep Neural Networks for Image Classification by Evolving and Stacking Transferable Blocks”. In: *ArXiv Preprint*. URL: https://arxiv.org/abs/1907.12659.

— (2021). “Surrogate-Assisted Particle Swarm Optimization for Evolving Variable-Length Transferable Blocks for Image Classification”. In: *IEEE Transactions on Neural Networks and Learning Systems*. DOI: 10.1109/TNNLS.2021.3054400.

Wang, B., Y. Sun, et al. (2019b). “A Hybrid GA-PSO Method for Evolving Architecture and Short Connections of Deep Convolutional Neural Networks”. In: *Proceedings of the Pacific Rim International Conference on Artificial Intelligence*. Springer, pp. 650–663. DOI: 10.1007/978-3-030-29894-4_52.

Wang, L., A. Wong, et al. (2020). *Actualmed COVID-19 Chest X-ray Dataset Initiatives*. URL: https://github.com/agchung/Actualmed-COVID-chestxray-dataset.

Wang, W. and L. Zhu (2021). “Reliable Network Search Based on Evolutionary Algorithm”. In: *2021 International Conference on Computer, Control and Robotics (ICCCR)*. IEEE. DOI: 10.1109/ICCCR49711.2021.9349406.

Wang, Y., C. Xu, et al. (2018). “Towards Evolutionary Compression”. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery. DOI: 10.1145/3219819.3219970.

Whitley, L. D. (1989). “The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best”. In: *Proceedings of the 3rd International Conference on Genetic Algorithms*. Association for Computing Machinery, pp. 116–123.

Witsuba, M. (2018). “Deep Learning Architecture Search by Neuro-Cell-Based Evolution with Function-Preserving Mutations”. In: *Proceedings from the Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, pp. 243–258. DOI: 10.1007/978-3-030-10928-8_15.

World Health Organization (Last accessed August 22, 2020). *WHO Coronavirus Disease (COVID-19) Dashboard*. URL: https://covid19.who.int/.

Wyk, G. J. V. and A. S. Bosman (2019). “Evolutionary Neural Architecture Search for Image Restoration”. In: *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 1–8. DOI: 10.1109/IJCNN.2019.8852417.

Xie, L. and A. Yuille (2017). “Genetic CNN”. In: *The IEEE International Conference on Computer Vision (ICCV)*. IEEE, pp. 1379–1388.

Xu, H. et al. (2020). “CurveLane-NAS: Unifying Lane-Sensitive Architecture Search and Adaptive Point Blending”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, pp. 689–704. DOI: 10.1007/978-3-030-58555-6_41.

Yang, Z. et al. (2020). “CARS: Continuous Evolution for Efficient Neural Architecture Search”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE.

Yao, L. et al. (2020). “SM-NAS: Structural-to-Modular Neural Architecture Search for Object Detection”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 12661–12668. DOI: 10.1609/aaai.v34i07.6958.

Ye, Q., Y. Sun, et al. (2020). “A Distributed Framework for EA-Based NAS”. In: *IEEE Transactions on Parallel and Distributed Systems* 32.7, pp. 1753–1764. DOI: 10.1109/TPDS.2020.3046774.

Ye, W., R. Liu, et al. (2020). “Quantum-Inspired Evolutionary Algorithm for Convolutional Neural Networks Architecture Search”. In: *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. DOI: 10.1109/CEC48606.2020.9185727.

Yen, G. G. and Z. He (2014). “Performance Metric Ensemble for Multiobjective Evolutionary Algorithms”. In: *IEEE Transactions on Evolutionary Computation* 18.1, pp. 131–144. DOI: 10.1109/TEVC.2013.2240687.

Yotchon, P. and Y. Jewajinda (2020). “Hybrid Multi-population Evolution based on Genetic Algorithm and Regularized Evolution for Neural Architecture Search”. In: *2020 17th International Joint Conference on Computer Science and Software Engineering (JCSSE)*. IEEE. DOI: 10.1109/JCSSE49651.2020.9268416.

Young, S. R., D. C. Rose, T. Johnston, et al. (2017). “Evolving Deep Networks Using HPC”. In: *Proceedings of the Machine Learning on HPC Environments*. Association for Computing Machinery, pp. 1–7. DOI: 10.1145/3146347.3146355.

Young, S. R., D. C. Rose, T. P. Karnowski, et al. (2015). “Optimizing Deep Learning Hyper-Parameters through an Evolutionary Algorithm”. In: *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*. Association for Computing Machinery, pp. 1–5. DOI: 10.1145/2834892.2834896.

Yu, Q. et al. (2020). “C2FNAS: Coarse-to-Fine Neural Architecture Search for 3D Medical Image Segmentation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE.

Yuan, G., B. Xue, and M. Zhang (2020). “A Graph-Based Approach to Automatic Convolutional Neural Network Construction for Image Classification”. In: *2020 35th International Conference on Image and Vision Computing New Zealand (IVCNZ)*. IEEE. DOI: 10.1109/IVCNZ51579.2020.9290492.

Yuan, W., B. Dong, et al. (2021). “Evolving Multi-Resolution Pooling CNN for Monaural Singing Voice Separation”. In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 29, pp. 807–822. DOI: 10.1109/TASLP.2021.3051331.

Zador, A. M. (Nov. 2019). “A critique of pure learning and what artificial neural networks can learn from animal brains”. In: *Nature Communications* 10.3770. DOI: 10.1038/s41467-019-11786-6.

Zhang, H., Y. Jin, et al. (2020). “Sampled Training and Node Inheritance for Fast Evolutionary Neural Architecture Search”. In: *ArXiv Preprint*. URL: https://arxiv.org/abs/2003.11613.

Zhang, H., S. Kiranyaz, and M. Gabbouj (2018). “Finding Better Topologies for Deep Convolutional Neural Networks by Evolution”. In: *ArXiv Preprint*. URL: https://arxiv.org/abs/1809.03242.

Zhang, N., J. Wang, et al. (2020). “Multi-objective Cuckoo Algorithm for Mobile Devices Network Architecture Search”. In: *International Conference on Artificial Neural Networks (ICANN)*. Springer. DOI: 10.1007/978-3-030-61609-0_25.

Zhang, Q., B. Li, and Y. Wu (2018). “Evolutionary Structure Optimization of Convolutional Neural Networks for Deployment on Resource Limited Systems”. In: *International Conference on Intelligent Computing*. Springer, pp. 742–753. DOI: 10.1007/978-3-319-95933-7_82.

Zhao, Z., M. Jiang, et al. (2020). “Improving Deep Learning based Optical Character Recognition via Neural Architecture Search”. In: *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, pp. 1–7. DOI: 10.1109/CEC48606.2020.9185798.

Zhao, Z.-Q., P. Zheng, et al. (2019). “Object Detection With Deep Learning: A Review”. In: *IEEE Transactions on Neural Networks and Learning Systems* 30.11, pp. 3212–3232. DOI: 10.1109/TNNLS.2018.2876865.

Zheng, X. et al. (2020). “Rethinking Performance Estimation in Neural Architecture Search”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, pp. 11353–11362. DOI: 10.1109/CVPR42600.2020.01137.

Zhou, D., X. Zhou, et al. (2020). “EcoNAS: Finding Proxies for Economical Neural Architecture Search”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. DOI: 10.1109/CVPR42600.2020.01141.

Zhou, Y., G. G. Gen, and Z. Yi (2021). “A Knee-Guided Evolutionary Algorithm for Compressing Deep Neural Networks”. In: *IEEE Transactions on Cybernetics* 51.3, pp. 1626–1638. DOI: 10.1109/TCYB.2019.2928174.

Zhou, Y., G. G. Yen, and Z. Yi (2020). “Evolutionary Compression of Deep Neural Networks for Biomedical Image Segmentation”. In: *IEEE Transactions on Neural Networks and Learning Systems* 31.8, pp. 2916–2929. DOI: 10.1109/TNNLS.2019.2933879.

— (2021). “Evolutionary Shallowing Deep Neural Networks at Block Levels”. In: *IEEE Transactions on Neural Networks and Learning Systems*. DOI: 10.1109/TNNLS.2021.3059529.

Zhu, H. et al. (2019). “EENA: Efficient Evolution of Neural Architecture”. In: *The IEEE International Conference on Computer Vision (ICCV) Workshops*. IEEE. URL: http://openaccess.thecvf.com/content%5C_ICCVW%5C_2019/html/NeurArch/Zhu%5Clinebreak%5C_EENA%5C_Efficient%5C_Evolution%5C_of%5C_Neural%5C_Architecture%5C_ICCVW%5C_2019%5C_pa%5Clinebreak%20per.html.