Automatic design of convolutional neural network architectures using Multiobjective Evolutionary Algorithms

Technical Report No. CCC-22-006

by

Cosijopii García García

Doctoral Advisors:

Dr. Alicia Morales Reyes, INAOE
Dr. Hugo Jair Escalante Balderas, INAOE

Instituto Nacional de Astrofísica, Óptica y Electrónica

July, 2022
Santa María Tonantzintla, Puebla, CP 72840

Contents

Abstract

Convolutional Neural Networks (CNNs) have made significant contributions to Artificial Intelligence (AI) and have demonstrated exceptional performance in difficult computer vision tasks. In recent years, expert users have developed a number of specialized CNN architectures to deal with complex datasets. However, the automatic design of CNN through Neural Architecture Search (NAS) has gained importance to reduce human intervention.

One of the main NAS challenges is to design less complex and yet highly precise CNNs, where both objectives are in conflict. As an initial approach, CGP-NAS (Cartesian Genetic Programming-Neural Arquitecture Search), a multiobjective evolutionary optimization approach to target NAS for the image classification task, is presented.

The main concept is to use this graphical representation as a layout for CNN architectures. CGP is encoded using real-based solutions representation for ease of adaptation of well-established MOEAs, such as the NSGA-II explored in this study. A preliminary empirical assessment shows CGP-NAS achieves a very competitive performance when compared to other state-of-the-art proposals while significantly reducing the evolved CNN architecture's complexity.

Keywords: *Neural architecture search, CNN, image classification, CGP, multi-objective evolutionary optimization.*

1 Introduction

Convolutional Neural Networks (CNNs) have been widely explored in a variety of tasks related to image processing and computer vision, together with today's computational capabilities that have allowed their implementation [1]. However, CNNs' design and configuration has become increasingly complex, as it requires human expertise [1, 2, 3, 4]. Thus, a new challenging area known as Neural Architecture Search (NAS) emerged as a way to design searching algorithms to discover near optimal and accurate deep architectures that would allow them to widen their application arena.

Image processing and computer vision tasks successfully targeted by CNNs have led the way to tailored designed CNN architectures such as GoogLeNet, ResNet, DenseNet, etc [5]. Considering their high complexity, researchers started to focus on their automatic design through NAS, aiming at finding optimal and accurate CNNs for specific datasets.

In addition to finding highly accurate architectures, real-world scenarios require computationally efficient CNNs. Considering both targets poses a problem with two conflicting objectives: maximizing the model's accuracy and reducing its complexity. Thus, a multiobjective problem needs to be solved in which a solution represents a compromise between both objectives. NAS has approached the problem primarily through a single objective optimization problem. [6, 7, 3, 4]. There have also been a few Multiobjective Evolutionary Algorithms (MOEAs) based proposals such as [8, 5, 2, 9] that have reported promising results using, for example, classical MOEA: NSGA-II (Nondominated Sorting Genetic Algorithm II).

1.1 Motivation

Recent and rapid advances in Deep Learning, particularly in Convolutional Neural Networks (CNNs), present a challenge and a requirement to automate their design and configuration. According to the No Free Lunch (NFL) theorem [10, 11], it is impossible to generalize an optimization algorithm to be robust and efficient for all posing problems, and that the only way to develop an algorithm that performs better than others is through specialization. In NAS and specifically, NAS applied to CNNs, specialization is sought for specific datasets, either for determined applications or for understanding how architectures can influence their performance. NAS has adopted in its framework different searching methods based on three main paradigms [12]: Reinforcement Learning (RL), Evolutionary Algorithms (EAs), and Bayesian Optimization (BO). In this proposal, EAs are studied for their high perfor-

mance when dealing with multiobjective problems, as well as their flexibility in their solution representation and operations design, as they can be adapted and modified according to a problem context. Therefore, this research aims at developing EAs as the main mechanism for searching for accurate and less complex neural architectures. To the best of our knowledge, current research shows that multiobjective algorithms have not been deeply studied when targeting this arena, and an open niche exists for MOEAs development to design competitive CNNs.

1.2 Justification

Neural architecture search is a relatively new area [12]. A number of efforts from the scientific community to take advantage of diverse searching mechanisms and to apply them to this new challenge have been carried out. This automatic design of complex CNNs considering today's computational capabilities is becoming more and more feasible.

Using Evolutionary Computation as a NAS strategy has been carried out through different research proposals. Most of them approach the problem as a single objective, attempting to maximize accuracy for a specific dataset. Few works on multiobjective approaches have been presented, and the majority of them are based on basic algorithmic versions such as NSGA-II and binary representations [2, 8, 5]. Other bio-inspired algorithms such as PSO (Particle swarm optimization) in its multiobjective version have been explored, mostly focusing on maximizing accuracy and minimizing MAdds (Multiply-Adds operations) for a given dataset [2, 8, 5]. Thus, multiobjective evolutionary algorithmic techniques have been scarcely explored through NAS. Solutions representation capabilities, inherent searching mechanisms, and population dynamics are some aspects yet to be studied that can provide highly efficient CNN solutions.

2 Background

This section provides the knowledge basis for the main topics involved in this research. An introduction to evolutionary algorithms and the solution representation explored in this study: Cartesian Genetic Programming (CGP).

After that, a general introduction to the multiobjective optimization arena will conclude with topics related to classification and deep neural networks.

2.1 Evolutionary Algorithms

Evolutionary algorithms (EAs) are a subset of algorithmic techniques within the Evolutionary Computation (EC) area, which has been roughly considered as a scope within Artificial Intelligence (AI), and more accurately as a Soft Computing field of study. EAs are inspired by the process of natural evolution as they try to simulate the main behavior of Darwin's theory of evolution and are commonly applied in search and optimization processes [13]. There are different application contexts for evolutionary algorithms, but the main ones are focused on optimization. In this area, three different kinds of problems are targeted: single, multi, and many-objective optimization problems.

Most evolutionary algorithms share the same general stages. They fall into the category of generate and test algorithms, where the search is guided by stochastic variations through different operators [13]. Evolutionary algorithms are based on a set of solutions or a population. These solutions interact through recombination and mutation (main genetic operations) by mixing information from two or more candidates to create a new one by inducing small changes within solutions and thus introducing diversity. Figure 1 depicts a general scheme of an evolutionary algorithm.

2.2 Cartesian genetic programming

Cartesian genetic programming (CGP) was initially conceived to evolve digital circuits. It was proposed by Miller in 1997[14] as a general form of genetic programming (GP) in 2000 and is called Cartesian because it represents a program using a two-dimensional grid [15]. CGP, unlike Genetic Programming (GP), is based on acyclic graphs for solution representation that allows forwarding connections. CGP shares important features with GP, such as the definition of a set of functions and their arity. Figure 2 shows a CGP solution representation example in which the mapping

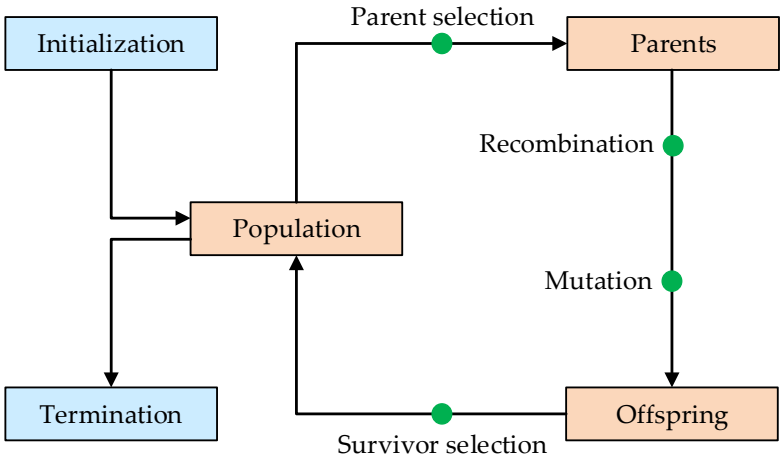


Figure 1: General scheme of Evolutionary algorithms.

of the solution to its phenotype, in this case, a CNN architecture, is shown. However, CGP can also be applied to different areas, such as automatic design of digital circuits, mathematical equations, and even artistic applications [15].

In the genotype space, a solution is mapped as an integer-based vector divided by segments that represent the function identifier (this refers to a function predefined on a set of functions that are taken as nodes) and its connections. The size of each segment varies depending on the maximum arity of the represented function; in the example, the maximum arity is two. Another CGP characteristic is the inactive nodes that are not expressed in the phenotype (in Figure 2, these are represented by the grey sections), but it is information that is maintained in the genotype and is exploited during the evolutionary process, for example, by changing an important connection between two functions that maps as a big change in the phenotype. Considering this CGP scenario, the crossover operator is not used (but can be implemented). Instead, only mutations occur as random changes in connections and nodes.

The CGP is set up on a fixed-size grid $N_r \times N_c$, with the number of connections between nodes determined by an l variable known as level-back. Normally, the number of inputs and outputs depends on the problem. CGP in its base version is combined with the $(1 + \lambda)$ evolutionary strategy algorithm. However, CGP can be adapted to other evolutionary searching algorithms.

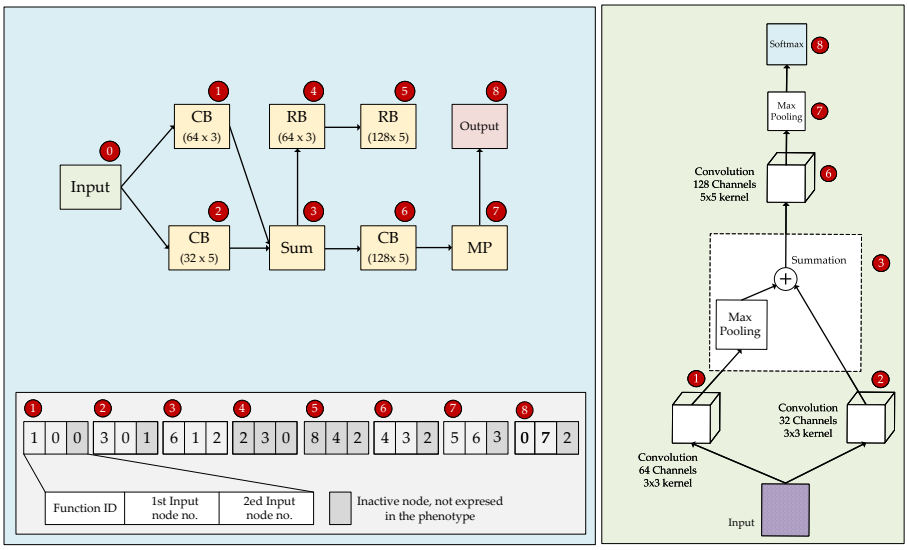


Figure 2: Cartesian Genetic Programming representation. Top-left: individuals represented as acyclic graphs. Bottom-left: integer-based individual representation Right: individual corresponding phenotype (CNN).

2.3 Multiobjective Optimization

In multiobjective problems (MOPs), solution quality is determined by the relationship between several objectives that are in conflict [13, 16]. Solving MOPs implies finding trade-offs among all the objective functions. In these kinds of problems, a set of optimal solutions is obtained instead of a single one, as in the case of single-objective problems. This is because, in multiobjective optimization, it is not possible to find a single optimal solution that optimizes all the objective functions simultaneously. There are alternatives to avoid this problem, for example, combining the fitness of each function and thus obtaining a single measure. Normally, each function will be weighted with some fixed value. This approach is called *weight sum method*[13, 17, 16]. There are also other classic methods such as ϵ-*Constraint method* which only takes one function of a multiobjective problem and the others in conflict are taken as constraints [16, 17]. A classic method is *goal programming*, where the main idea is to find solutions close to predefined objectives for each target. If these solutions are not reached, the task is to derive such objectives and attempt minimization [16]. Multiobjective evolutionary algorithms have the advantage of obtaining a set of Pareto-optimal solutions, and that no previous knowledge of the problem, like weight vectors, is required. Finally, classic methods can not find some Pareto optimal solutions when MOPs are not convex. This relates to the Pareto front's complexity and the difficulty of finding such solutions [16].

2.3.1 Basic concepts

This section explains basic concepts of multiobjective optimization.

- **Decision variables** are represented by a vector $\mathbf{x}$ with n decision variables represented by Equation 1 [17].

$$\mathbf{x} = [x_1, x_2, \ldots, _n] \tag{1}$$

- **Constraints** are imposed by environment characteristics or resources and occur in most optimization problems. They are expressed in the form of mathematical equalities or inequalities, represented in Equations 2 and 3. If the number of equality constraints is greater than the number of decision variables, the problem is over constrained so there are not enough degrees of freedom for optimization [17].

$$h_j = 0 \ , \ j = 1, 2, \ldots, p \tag{2}$$

$$g_i \leq 0 \ , \ i = 1, 2, \ldots, m \tag{3}$$

- **Objective function,** in multiobjective optimization, a set of objective functions are used to evaluate the decision variables vector: $f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x})$ where k is the number of objective functions in a multiobjective problem. The vector of objective functions $\mathbf{f}(\mathbf{x})$ is defined as [17] :

$$\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x})] \tag{4}$$

- **Decision variable and objective function space** are defined by a n-dimensional space in which each coordinate axis corresponds to a component of a decision variables vector $\mathbf{x}$; and the objective function space is defined by a k-dimensional space in which each coordinate axis corresponds to a vector component $\mathbf{f}_k(\mathbf{x})$. Each point in the decision variable space represents a solution, when this vector is evaluated in the objective function, the obtained value represents a point in the objective function space which determines solution's quality. Therefore, a $F : \mathbb{R}^n \rightarrow \mathbb{R}^k$ function maps the space of decision variables to the objective function space [17]. Figure 3 shows this process.

2.3.2 Multiobjective optimization problem

A multiobjective optimization problem involves multiple conflicting objective functions, which must be minimized or maximized simultaneously. Those functions could

be subject to a number of constraints and variable bounds. Mathematically, a Multiobjective Optimization Problem (MOP) is defined as:

General MOP [18]:

$$\left.\begin{aligned} \text{Minimize/Maximize } \mathbf{f}_m(\mathbf{x}), m = 1, 2, \ldots, k; \\ \text{subject to } g_j(\mathbf{x}) \geq 0, j = 1, 2, \ldots, m; \\ h_k(\mathbf{x}) = 0, k = 1, 2, \ldots, p; \\ x_i^{(L)} \leq x_i \leq x_i^{(U)}, i = 1, 2, \ldots, t; \end{aligned}\right\} \tag{5}$$

with k objectives, m and p are the number of inequality and equality constraints. A solution $\mathbf{x} \in \mathbf{R}^n$ is a vector of n decision variables: $\mathbf{x} = [x_1, x_2, \ldots, x_n]$, which satisfy all constraints and variable bounds [18, 17, 16]. The last set of constraints is called variable bounds, which restrict each upper and lower decision variable x_i. These limits are the decision variable space size. If a solution $\mathbf{x}$ satisfies all restrictions and variable bounds, it is known as a **feasible solution**. The set of all feasible solutions is called the **feasible region** [16]. It can be seen in Figure 3 that solutions within the blue area are feasible solutions, and their set determines a feasible area.

2.3.3 Dominance and Pareto optimality

In multiobjective optimization problems, several objectives conflict, this means that more than one optimal solution exists. These solutions are known as *Pareto-optimal* solutions. Definition of a Pareto-optimal solution is related to the domination concept as follows: Pareto dominance [17]: A vector $\mathbf{u} = (u_1, u_2, \ldots, u_k)$ is said to dominate another vector $\mathbf{v} = (v_1, v_2, \ldots, v_k)$ (denoted by $\mathbf{u} \preceq \mathbf{v}$) if and only if $\mathbf{u}$ is partially less than $\mathbf{v}$, this is specified as follows: $\forall i \in \{1, \ldots k\}, u_i \leq v_i$ and $\exists i \in \{1, \ldots k\} : f(u_i) < f(v_i)$.

Pareto Optimal Set [17], for a given MOP and $F(x)$, the POS $\mathcal{P}^*$ is determined by:

$$\mathcal{P}^* = \{\mathbf{x} \in \Omega \mid \neg\exists \mathbf{x}' \in \Omega \; F(\mathbf{x}') \preceq F(\mathbf{x})\} \tag{6}$$

These solutions are represented in the decision variable space. Non-dominated solutions represented in the Pareto optimal set are the best solutions with a trade-off among objectives. When mapping these solutions to the objective function space, a set called Pareto front ($\mathcal{PF}^*$) is defined next [17]:

For a given MOP, $F(X)$ and POS, $\mathcal{P}^*$, the Pareto Front $\mathcal{PF}^*$ can be expressed as:

$$\mathcal{PF}^* = \{\mathbf{u} = F(\mathbf{x}) \mid \mathbf{x} \in \mathcal{P}^*\} \tag{7}$$

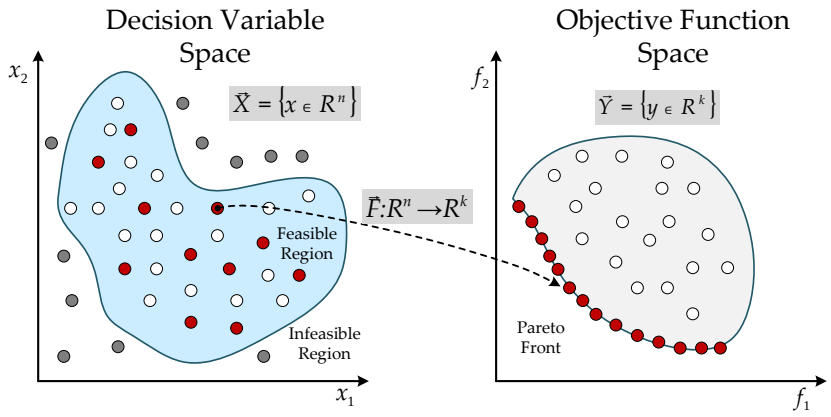


Figure 3: Decision variables space to objective function space mapping. Feasible solutions region is in blue. In the decision variable space, the Pareto optimal set is shown as red dots and its mapping to the objective function space creates the Pareto front.

In Figure 3, a mapping example of solutions from the Pareto optimal set to the objective function space is shown, therefore the Pareto front is created with solutions with a trade-off among objectives.

2.4 Multiobjective evolutionary algorithms

The multiobjective evolutionary algorithms (MOEAs) in comparison with the classic mathematical programming methods mentioned briefly in the last section, have certain characteristics and advantages that make them applicable to solve MOPs [16]. Therefore, in this section different evolutionary algorithms approaches to target multiobjective problems are reviewed. These algorithms can be divided into three main paradigms [19], Pareto based MOEAs, decomposition based MOEAs and indicator based MOEAs. In this proposal, Pareto based MOEAs and decomposition based MOEAs are explained.

2.4.1 Pareto-based MOEAs

Pareto-based MOEAs use a dominance based ranking scheme and combine elitist strategies such those that converge to a global optimal in some problems [13]. Pareto and elitist strategies lead the way or set the basis for one of the most important algorithmic approaches in the area: NSGA-II algorithm proposed by Deb et. al. [20]. Pareto based MOEAs have in common the use of Pareto dominance with

some diversity criteria based on secondary ranking, some algorithms of this class are Multiobjective Genetic Algorithm (MOGA) [21], which was the first MOEA, Pareto Archived Evolutionary Strategy (PAES) [22] uses a mesh in the objective function space to ensure that all regions are visited, Strength Pareto Evolutionary Algorithm (SPEA) [23] which uses a different criterion based on dominance, and it also ranks individuals depending on how many individuals dominate and are dominated, as well as making use of clustering. In a second version (SPEA-2) [24] both previously described criteria are improved.

2.4.2 MOEAs based on Decomposition

An inevitable problem when dealing with MOPs with more than 3 objectives is that dominance begins to be ineffective. Therefore, the idea of ranking the Pareto front is not useful. Therefore, decomposition-based methods have been adapted to MOEAs. These methods can deal with two, three, or many-objective problems. They provide a reliable and powerful alternative for these kinds of problems. One of the most important algorithms in this approach is MOEA/D (Multiobjective Evolutionary Algorithm based on Decomposition), developed by Zhang and Li [25, 13, 26].

2.5 Classification and pattern recognition

Object recognition is based on the assignment of classes to objects, and the process that makes this assignment possible is called classification. The classifier (similar to a human being) does not decide on the class of the object itself; rather, object properties are used to serve this purpose. For example, to distinguish steel from sand, it is not necessary to determine their molecular structures but to describe the materials themselves, properties such as texture, weight, hardness, etc. These detected features are called patterns, and classifiers currently do not recognize objects but their patterns. [27]. A set of properties of the elements is chosen because they describe some characteristics of the objects; these properties are measures that appropriately describe those objects' patterns. Object properties can be either quantitative or qualitative, and their form and characteristics can be various (numerical vectors, strings, etc.). To describe a static object, a numerical elementary description called "features" is used in image analysis; the feature vector $x = x_1, x_2, x_n$ describes the object; the set of all possible patterns form the pattern space X (also called feature space) if the features are correctly chosen, the similarity of the objects in each class will be reflected in their patterns in the pattern space. Classes form clusters in the

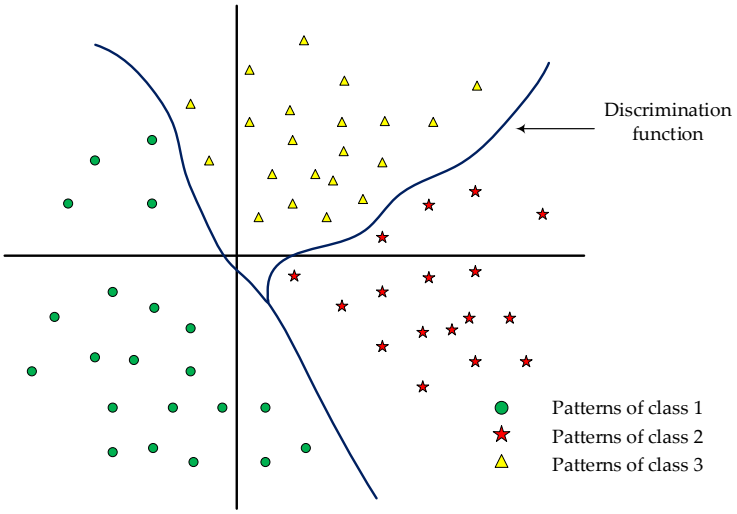


Figure 4: Example of a discrimination function, using three different classes.

feature space, which can be separated by a discrimination curve [27] (or hypersurface in multidimensional feature space) see Figure 4.

2.6 Neural networks

An artificial neuron is an imitation of the process of a biological neuron. A simplified model was created because this is a very complex system to imitate. This simplified model includes the signals coming from other neurons, the threshold function, and output. The simple perceptron was proposed by Rosenblatt and Minsky [28, 29]. It can be seen in Figure 5. The perceptron can be represented with a linear discrimination function (see Equation 8). A single perceptron is equivalent to a linear classifier that classifies patterns into two classes. A perceptron receives a feature vector x as input and produces a scalar as output [28].

$$y = \varphi(\sum_{i=1}^{n}(w_i x_i + w_0)) \tag{8}$$

Where w is composed by weights defining the hyperplane and $varphi$ is a nonlinear function with a threshold, in most cases with the function sigmoid[27].

Neural networks are composed of nodes or units connected directly with links. Each link is associated with a weight w, which determines the strength and shape of the connection. The architecture of the network, the nearby nodes, the intermediate

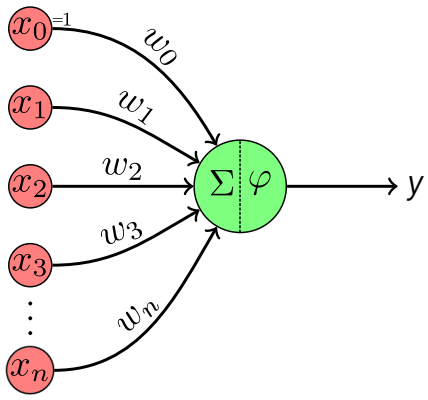


Figure 5: Simple perceptron representation.

nodes and their weights w, the threshold w_0, and the nonlinear function *varphi* define the network's properties [30, 28].

2.6.1 Feedforward networks

Feedforward networks or multilayer perceptrons (MLP) have connections only in one direction. This forms a directed acyclic graph, where each node receives its inputs from the upper nodes and sends the information to the lower nodes without cycles[30, 28]. The weights of each input are randomly initialized, to be subsequently changed during an iterative learning process. It is known that three layers are sufficient to determine any arbitrary discrimination function, assuming that the network consists of a sufficient number of nodes. These three layers are commonly referred to as the input layer, hidden layer, and output layer. A three-layer neural network is shown in Figure 6.

The learning of feedforward networks is based on the backpropagation algorithm. The set of patterns is chosen to be fed into the network, the information is carried to the next layers, and the generated output y is the result of the last layer.

The Backpropagation algorithm compares the acquired output y with the required classification ω. The weights w_{ij} are recalculated for adaptation in order to minimize the classification errors [27, 30].

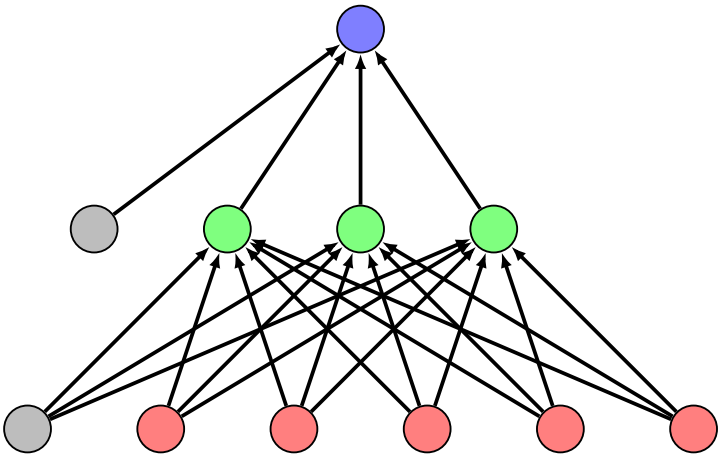


Figure 6: Multilayer perceptron arquitecture, input layer (red), hidden layer (green), and output layer (blue).

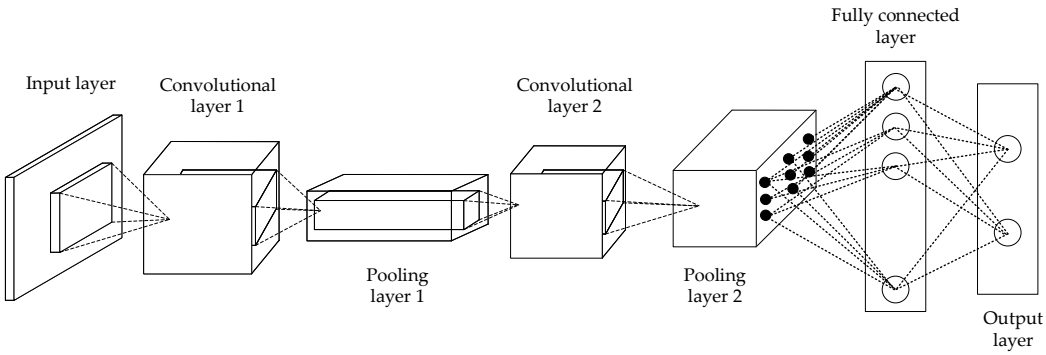


Figure 7: Basic architecture of convolutional neural network.

2.7 Convolutional Neural Networks

Convolutional Neural Networks (CNN) is one of the best known and most widely used deep architectures inspired by the animal visual system, which has had important applications in computer vision [31]. This type of network is known to have a mesh topology. CNNs are feedforward neural networks where inputs are transformed through several sequential layers [28].

The main differences between CNNs and feed-forward neural networks (FFNNs) are that they consist of multiple layers as opposed to a single layer in FFNNs. A CNN consists of three different types of layers: convolutional layers, pooling layers, and fully connected layers [31, 28]. Figure 7 shows a general CNN flow for an image classification task. Usually, the input is a three-dimensional tensor such as an image with height, width, and color channels.

The image enters at the input layer and goes through many convolution processes and pooling layers to generate the representations which feed the last part of the network, the fully connected layer [31].

2.7.1 Convolution layers

Convolution layers are the CNN's core. These layers work as feature extractors. Each neuron in the convolution layers is connected to small regions of the previous layers called *receptive fields* which are known as kernels or filters [31]. This operation is normally defined as seen in Equation 9 in its continuous form or in its discrete form as seen in Equation 10 [28].

$$s(t) = \int x(a)w(t-a)da \tag{9}$$

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{10}$$

Designing convolutional layers requires a number of parameters, depending on the number of filters learned from the previous layer. The layer output is linked to the following parameters: filter size, stride size, and padding. The spatial dimension of the filter is called the kernel size. The stride defines the step with which the filter will slide through the input to create the feature map, and the padding adds zeros at the edges of the input. After obtaining the feature map, the rectified linear unit (ReLU) activation function is normally used, which can contribute to an improvement in the performance of the CNN [31].

2.7.2 Pooling layers

Pooling layers are used to perform spatial invariance translations. Usually, pooling layers are embedded between successive convolution layers to reduce the spatial size of feature maps. Pooling layers reduce the number of parameters and the number of calculations progressively and also help to control overfitting. The pooling function works along the spatial dimension of the input volume and reduces it but keeps the depth constant [31, 28].

The most commonly used methods in CNN are max-pooling and average-pooling. Similar to convolution, pooling operators are performed with a defined filter size and stride. For example, the Max-pooling operator is set to a region and selects the

largest value. On the other hand, the Average-pooling operator gets the average of that region [31].

2.7.3 Fully connected layers

After many learning cycles over progressive layers of convolution and pooling, CNNs extract high-level features from the input data and perform high-level reasoning based on the extracted representation. Finally, fully connected layers are used at the end. Like FFNNs, fully connected layers are one-dimensional and all neurons in the fully connected layer are connected to all neurons in the preceding layer.

Fully connected layers contain most of the CNN parameters and force a computational load for training. In addition to these three layers (convolution, pooling, and fully connected), many types of standardization layers have been used in CNN, among which batch normalization is the most common. Some studies have shown that batch normalization can help accelerate training and reduce the sensitivity of training to weight initialization [31].

2.8 Neural architecture Search

Neural architecture search (NAS) is one of the current topics of Auto Machine Learning (AutoML), starting its boom in 2017 to this day, because of the success that deep learning has had in different fields of application and the complexity of manually refining these models.

NAS is mainly focused on finding the best architecture and hyperparameters of a deep learning model [32]. Empirical studies carried out in recent years show that NAS is able to find better models than those manually tuned [33]. One widely used search strategy is reinforcement learning, but it is focused only on optimizing one objective. On the other hand, evolutionary algorithms can deal in a more effective way by approaching NAS as a multiobjective problem using MOEAS, obtaining in every generation a subset of suboptimal solutions from the Pareto front [33].

The general NAS scheme is shown in Figure 8 , in which three important elements are included: the search space, the search strategy, and the performance estimation strategy. The search space is determined by the problem, and the NAS method would rely on the solutions' representation flexibility and the capability of the searching algorithmic technique.

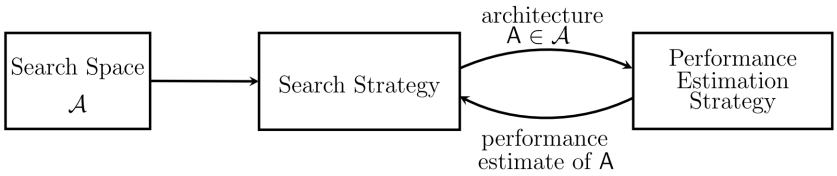


Figure 8: General scheme of the Neural architecture search[12]

The search strategy refers to the type of algorithm to be used, for example, genetic algorithms or reinforcement learning. The performance estimation strategy refers to how the performance of the obtained architecture is measured[12].

3 Related work

Deep architectures, specifically convolutional neural networks (CNN), have been designed by specialists through hands-on work.Recent exponential development of computational capabilities, areas like Neural Architecture Search (NAS), can use those available resources to create different methods that enhance new CNNs' designs. In NAS, methods can be divided into two categories: evolutionary algorithms (EA) and reinforcement learning (RL) approaches [2].

Using EAs to find good performance architectures is not a recent problem; the first approach started in the 90s, and the most important work was in 2002 when Neuroevolution of Augmenting Topologies (NEAT) was proposed [34]. In the original paper, NEAT works with small artificial neural networks (ANNs), finding connections between nodes and at the same time updating the architecture's weights. Multiple approaches originated from NEAT like HyperNeat [35] or ES-HyperNeat [36].

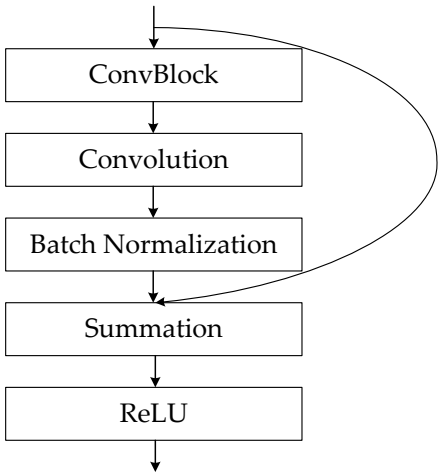
Recently, scientific focus has been driven towards Deep Learning, modifying how these even more complex architectures have to evolve [1, 8].

CNNs' basic perspective is to generate a feature extractor that, through a set of operations, will generate a feature map through each layer of the network to generalize patterns from the input image to finalize in a classifier. Therefore, connections and operations cause a significant effect within base blocks and the complete architecture.

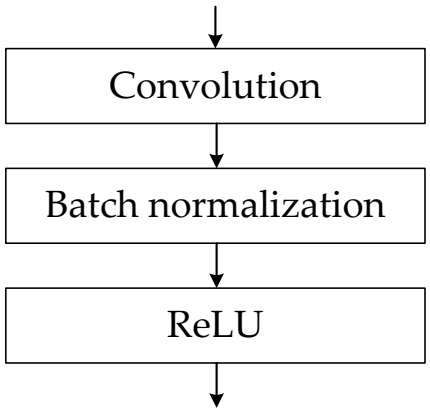
Regarding the search for these architectures, as explained above, NAS is trying to cope with this problem. Therefore, NAS currently has two main search approaches: single-objective and multi-objective.

In [1] CooDeepNEAT is proposed as an extension of NEAT applied to CNNs. The principal contribution was the use of coevolution. silicet, two subpopulations evolve independently, and during fitness evaluation, both are combined. This work uses a genetic algorithm scheme, and the objective was to minimize testing error (100%− Accuracy). In this case, the CIFAR-10 dataset was targeted, and the final result was a model with a 7.3% error.

L. Xie and A. Yuille [4] introduced a method called Genetic CNN. This algorithm proposed a new encoding scheme based on binary representation from Genetic Algorithms (GAs). In their solution representation, every bit in the binary vector represents a connection between nodes. Such nodes are convolutional operations, thus



(a) ResBlock



(b) ConvBlock

the proposed encoding scheme allows to find existing VGG, ResNet, and DenseNet network patterns. These modules where nodes exist are isolated sets of other operations that are performed in a CNN, such as pooling and fully connected layers, which are already fixed in the model and are not evolved. Concerning evolutionary operators, the mutation is performed with a low probability using a bitflip, and the crossover is carried out by switching only different bits in the bit string with a definite probability. The final model was called GeNet and has different structures than those designed manually. Patterns were found that exist in networks such as VGGNet, AlexNet, and GoogLeNet as well as in residual networks, taking as a principal objective the accuracy of the CIFAR-10 dataset.

For comparison purposes, a set of individuals were initialized with a 40-layer ResNet architecture. After evolving the architecture, the final results for the CIFAR-10, CIFAR-100, and SVHN datasets were 5.39%, 25.12%, and 1.71% errors, respectively.

M. Suganuma et al [7] proposed a new method using genetic programming to design CNNs, the approach used Cartesian genetic programming (CGP) as an encoding technique. The main idea is to first define some function blocks, which are defined as operations in CNNs, like max pooling, average pooling, and more complex operations like ResBloks (see Figure 9a) which are a series of convolution layers and contain "shortcut" connections, or more simple ConvBloks (see Figure 9b) which contain a standard convolution process. Finally, the summation and concatenation functions are defined.

The functions explained above are used by CGP's graph encoding proposed by Miller[14]. A solution is represented on a $N_r \times N_c$ grid where R is the number of rows and C the number of columns. This encoding allows the definition of structures like ANNs and, in this case, CNNs, by the nature of the forward connections. The evolutionary algorithm used in this work is a $(1+\lambda)$ evolutionary strategy, and only mutation operations are applied. Two models were created from each dataset, each one differentiated by the type of block. The first model, called CGP-CNN(ResSet) does not contain the convolution blocks in isolation; the ResBlock block has them by definition. The other CGP-CNN (ConvSet) model does not contain ResBlock blocks, only predefined ConvBlocks. Results for CIFAR-10 were as follows: CGP-CNN (ResSet) obtained a 5.01% error and CGP-CNN (ConvSet) a 5.92% error. Both results were the best reported. For the CIFAR-100 dataset, CNN (ConvSet) obtained 26.7% and CNN (ResSet) obtained 25.1%. From the results, it is noted that the CNN (ResSet) model performed better, and the resulting architectures have similarities with ResNet networks.

In [37], a new method to design CNN architectures using blocks based on ResNets and DenseNets for a given dataset is presented. These blocks were called ResBlocks and DenseBlocks, respectively. Also, a pooling layer block is considered. The main idea is to propose a new genotype representation with the advantage of having variable length representation. In the evolutionary process, the genetic algorithm without changes is used with binary tournament selection in addition to the crossover and mutation phases. Individuals with inconsistencies in their encoding are fixed. Testing included CIFAR-10 and CIFAR-100 datasets, with results showing a classification error of 4.3% and 20.85% on CIFAR-10 and CIFAR-100, respectively, and GPU processing days of 27 and 36, respectively. The obtained model was called AE-CNN. Among its most important features is the reduction of the number of model parameters, as in the case of the model for the CIFAR-100 dataset, the architecture found is much smaller than the one found for CIFAR-10.

In [38] a new graph encoding is introduced and applied to NN architecture design. Architectural search algorithms have achieved good performance, but there is a limit to CNN representation. Therefore, a new encoding using a direct acyclic graph was proposed. Five principal blocks were used: Convolution, Summation, Concatenation, Max and average pooling. A solution based on these blocks is created, and summation and concatenation blocks allow residual connections because they allow multiple inputs. In the experimental design, the CIFAR-10 dataset was targeted. In this work, a random search was used to validate the approach. Final results show that the encoding can find an architecture with an accuracy of 92.57% but it is worse when compared with other state-of-the-art approaches.

A benefit noticed by the authors is that the algorithm can find architectures with fewer parameters in comparison to other approaches using only random search.

Bakhshi, Chalup, and Noman [39] proposed an algorithm based on evolutionary algorithms to evolve CNN architectures. In their approach, hyperparameters are included in the evolutionary architecture's process. The problem is represented by predefined blocks that are inspired by VGG architectures, which are a sequence of convolution layers followed by pooling layers and finally a fully connected layer at the end. Therefore, the size of the vector that will represent will be of a fixed size, to later add the hyperparameters, having this way a final vector that will represent the CNN architecture and its hyperparameters, which are are the learning rate, weight decay, momentum, and dropout. To evolve the architectures, a standard genetic algorithm is used, and for evaluation, each architecture was trained for only ten epochs, which the authors determined as enough for the architecture's quality evaluation. Three datasets, CIFAR-10, CIFAR-100, and SVHN, were used to test the designed algorithm. The final results for CIFAR-10, CIFAR-100, and SVHN were 94.75%, 75.90%, and 95.11% respectively.

In conclusion, the proposed algorithm was competitive with the compared models, as well as a reduction in GPU days was reported. It is concluded that the use of partial training for structure evaluation can help in not-so-complex datasets since the results on CIFAR-100 and SVHN were not competitive.

3.1 Multiobjective NAS

In [40] Neuro-Evolution with Multiobjective Optimization (NEMO) is proposed, which is one of the first multiobjective approaches using evolutionary algorithms. A CNN is optimized considering two objectives: the error and a unit of inference time (in milliseconds), aiming to minimize both objectives. The representation is based on an integer vector which is fixed with reference to the LeNet CNN architecture and different numbers of channels are searched for each layer of the network. NSGA-II was used to carry out the evolutionary process, targeted datasets were MNIST and CIFAR-10 and a real-world problem called drowsy behavior recognition task for driver monitoring system.

The authors concluded that their proposal found faster networks that maintained competent accuracy, even though it required a lot of GPU power. This work only focuses on modifying the parameters of predefined networks, but its multiobjective approach focuses more on finding the trade-off between CNN requirements (accuracy

and inference speed). In [2] propose an algorithm called NSGANet for NAS based on the famous multiobjective optimization algorithm NSGA-II, where the main attraction is the use of two objectives: accuracy and FLOPs. In NSGA-Net, solutions are encoded as binary strings [4].

Therefore, crossover and mutation operations are defined accordingly. In the crossover, for both parents that do not share the same bits at the same positions, the offspring are generated by random bits at those positions.

After recombination, mutation is applied with a bit flip at a random position. The evolutionary process is carried out by NSGA-II with an added mechanism to promote exploitation of the searching space based on the Bayesian Optimization Algorithm (BOA)[41], BOA finds relationships among blocks and routes. At this stage, a specific number of individuals is created resulting from the Bayesian network's estimation of the individuals in the current population.

NSGA-Net on the CIFAR-10 dataset obtained a 3.85% error, 3.34 million parameters, and 1290 MFLOPs. These results are competitive in terms of accuracy with other state-of-the-art NAS proposals, while reducing the computational load required to 8 GPU-Days.

Wan, Xue, and Zhang [9] proposed a particle swarm optimization (PSO) neural architecture search algorithm for evolving deep convolutional neural networks for image classification. One of the most important contributions was the development of a new representation based on computer network encoding. The representation used is based on computer networks and IP coding. The subnetworks concept is applied where every series of layers in the CNN is divided into subnetworks with a predefined value. This encoding allows linear networks where layers are connected one after another and cannot be explored, for example, ResNet networks. A benefit of this encoding is that the hyperparameters are included in the encoding, which makes it easier to adapt these parameters to a dataset.

Two versions were proposed, one single-objective (IPPSO) and the other multiobjective (MOCNN), both using the proposed encoding approach. In the single-objective approach, PSO was used to locate the best architecture, using the accuracy of the trained model as the fitness value. To measure the performance of the method, three datasets were chosen: MNIST basic, MNIST with Rotated Digits plus Background Images, and Convex Sets, all used in image classification tasks.Results were competitive to other CNN models, even though the found models were smaller in most cases. Regarding the multiobjective approach, accuracy and FLOPs were the two considered objectives. FLOPs are a way to measure the complexity of the model.

For the comparison of the MOCNN model, the authors used the DenseNet-121 model and the CIFAR-10 dataset. Both were trained in the same way. The classification error of the DenseNet-121 model was 94.77% while the MOCNN model was 95.51%. Although the exact number of FLOPs is not mentioned in the paper, the authors mentioned that the MOCNNN model obtained promising results.

In [42] an evolutionary multiobjective NAS algorithm was presented. Important points were the use of an NSGA-Net[2] inspired encoding, as well as a linear block-based design, where each block contains internal nodes. Unlike NSGA-Net, the objectives were to minimize the classification error and the parameters of the network. An important feature was the use of weight inheritance to reduce the number of epochs to train the network, since this method allows the inheritance of the weights of both parents to generate the offspring, and with this, the new architecture required less training. The multiobjective algorithm implemented apparently is a variant of NSGA-II, but it was not stated in the paper. To evaluate the algorithm named MOGIG-Net, CIFAR-10 and CIFAR-100 datasets were used, obtaining a 2.01% classification error and 3.7 million parameters in CIFAR-10, and a 14.38% classification error and 3.7 million parameters in CIFAR-100. The total time reported was 14 days. The technical specifications for experiment execution were not reported, making it more difficult to calculate the GPU-days. It was concluded that the addition of the weight inheritance method to this proposal helped to find better architectures, although complexity was not specified since the second objective was to minimize the number of parameters.

In [8] a new version of NSGANet was proposed, called NSGANetV1. In this approach, two objectives were tackled using the classification accuracy and architecture complexity (FLOPs) concerning the general approach. The NSGA-II algorithm was used by dividing the evolutionary process into exploration and exploitation. Exploration was based on selection, crossover, and mutation operations. Exploitation was based on the construction of a certain number of offspring using a Bayesian network. NSGANet can search the space of CNN architectures in addition to searching at the network level (e.g. # number of channels, # number of layers). The representation used is that of a direct acyclic graph divided into 5 main blocks. These blocks are spatially decreasing (and therefore increasing the number of channels) until the fully connected layer, each block having 5 internal nodes. Each internal node is based on operations like Max Pooling, Average Pooling, as well as convolution operations like local binary convolution or dilated convolution. Regarding mutation and crossover operators, they act at the block level as well as at the internal node level. To measure the performance of this proposal, CIFAR-10 and CIFAR-100 datasets were used as well as 5 models (NSGANetv1-A0-NSGANetv1-A4), each starting from A0 with

the lowest number of parameters to A4 with the highest number of parameters. In the CIFAR-10 dataset, the best performance was obtained with the A4 model with 97.98% Acc and 4.0M parameters, and in the CIFAR-100 with the A4 model also with 85.62% ACC and 4.1M parameters. The A1, A2, and A3 models were transferred to the ImageNet dataset and the A3 model obtained 76.2% of Top-1 Acc and 93.0% of Top-5 Acc. Overall, NSGANetV1 proved to be an algorithm with performance both in GPU-days as well as in models with fewer parameters and good accuracy.

After [8], the same authors proposed [5] called NSGANetV2, whose initial premise was to assist with surrogate models for both more efficient search and evaluation of individuals (CNNs training). Therefore, two surrogate approaches were established, the high-level approach assisting the search, which was based on an online learning algorithm that learns the architecture during a generation and can estimate the fitness values. Regarding the second level, a surrogate model called the Supernet model was used to inherit the weights from the new individuals to have pre-trained structures and thus reduce training time. The same representation is used in [5], which employs a directed acyclic graph to divide an individual into blocks, which are further subdivided into nodes. The search is assigned to these nodes, both the operation to be performed as well as the size of the channel. All of this is encoded in a vector of integers. The general process was to generate individuals and assign them the supernet weights. After that, these individuals were trained and added to a model that will learn from these structures. At the same time, the NSGA-II algorithm tries to evolve these structures, and fitness will be obtained. To verify the proposal, they used CIFAR-10 and ImageNet datasets. The following results were obtained for CIFAR-10: 98.4% of Top-1 acc exceeded the other state-of-the-art algorithms.In addition to obtaining a model with less complexity, In the same way, for Imagenet, 75.9% of Top-1 acc was obtained. In general, the algorithm outperformed the other compared state-of-the-art algorithms in speedup due to the surrogate models, which drastically reduced the epochs for CNN training. Table 1 shows a summary of the state-of-the art related algorithmic proposals.

Table 1: Reviewed works of the state of the art

Year	Name	Application	Benchmark	EC	Encoding	Model	Objetive	Proposal
2017	A genetic programming approach to design convolutional neural network architectures [6]	Image Classification	CIFAR-10	CGP, ES	Graph, Integer	CNN	Accuracy	CGP-CNN
2019	Evolution of Deep Convolutional Neural Networks Using Cartesian Genetic Programming [7]	Image Classification	CIFAR-10, CIFAR-100	CGP, ES	Graph, Integer	CNN	Accuracy	CGP-CNN
2017	NEMO : Neuro-Evolution with Multiobjective Optimization of Deep Neural Network for Speed and Accuracy [40]	Drowsiness Recognition for Driver Monitoring System	CIFAR-10, MNIST	NSGA-II	Integer	CNN	Accuracy Inference Speed	NEMO
2020	NSGANetV2: Evolutionary Multi-objective Surrogate-Assisted Neural Architecture Search [8]	Image Classification	Aircraft, CIFAR-10, CIFAR-100, CINIC-10 DTD, Flowers102, ImageNet, Pets, STL-10	NSGA-II	Integer	CNN	Accuracy MAdds	MSuNAS NSGA-NetV2
2020	Multi-Objective Evolutionary Design of Deep Convolutional Neural Networks for Image Classification [5]	Image Classification	CIFAR-10, CIFAR-100 ImageNet	NSGA-II	Block	CNN	Accuracy FLOPs	NSGA-NetV1
2019	NSGA-Net: Neural Architecture Search using Multi-Objective Genetic Algorithm [2]	Image Classification	CIFAR-10	NSGA-II	Binary	CNN	Accuracy FLOPs	NSGA-Net
2019	Evolving Deep Neural Networks [1]	Image Captioning	CIFAR-10	GA	NEAT-DNN	CNN, LSTM	Accuracy	CooDeepNeat DeepNeat
2020	Designing Convolutional Neural Network Architectures Using Cartesian Genetic Programming [43]	Image Classification Inpainting Denoising	CIFAR-10, CIFAR-100	CGP, ES	Graph, Integer	CNN	Accuracy	CGP-CNN
2017	Genetic CNN [4]	Image Classification	CIFAR-10 ILSVRC2012	GA	Binary	CNN	Accuracy	GeNet
2020	Particle Swarm Optimization for Evolving Deep Convolutional Neural Networks for Image Classification: Single and Multi-Objective Approaches [9]	Image Classification	CIFAR-10, Convex Sets MNIST Basic, MRDBI	OMOPOSO PSO	IP-based	CNN	Accuracy FLOPs	IPPSO MOCNN
2020	Fast Evolution of CNN Architecture for Image Classification[39]	Image Classification	CIFAR-10, CIFAR-100 SVHN	GA	Integer	CNN	Accuracy	GAnet

Table 1: Reviewed works of the state of the art

Year	Name	Application	Benchmark	EC	Encoding	Model	Objetive	Proposal
2019	A Graph-Based Encoding for Evolutionary Convolutional Neural Network Architecture Design [38]	Image Classification	CIFAR-10	Random search	Graph	CNN	None	Random search
2020	Completely Automated CNN Architecture Design Based on Blocks [37]	Image Classification	CIFAR-10, CIFAR-100	GA	Block	CNN	Accuracy	AE-CNN
2021	A Multi-Objective Evolutionary Approach Based on Graph-in-Graph for Neural Architecture Search of Convolutional Neural Networks [44]	Image Classification	CIFAR-10, CIFAR-100	NSGA-II	Block	CNN	Accuracy Parameters	MOGIG-Net

4 Research Development

This section states the formal components for this research proposal, including problem statement, research questions, hypothesis, objectives, scope and limitations, expected contributions, methodology, and work plan.

4.1 Problem Statement

State-of-the-art CNNs are becoming more complex and their performance depends more and more on their architecture and hyperparameters configuration, in addition to their high processing needs. Moreover, plenty of research has been done by developing specialized architectures for specific tasks [45]. However, due to the lack of DNNs understanding, it is still difficult to determine CNN's performance without empirical benchmark testing.

Accurate and less complex architectures are desirable in scenarios where applications and user requirements need models in which time is an important variable. However, searching for those optimal CNNs architectures becomes a time-consuming black-box optimization task. Many research works have been developed to automate CNN architectural search, this area is known as Neural Architecture Search (NAS). Several of those works have posed this problem as a single objective optimization problem while targeting the model's accuracy for a specific dataset [7, 39, 4]. Recent works have reevaluated NAS and approached it as a multiobjective optimization one since more than one objective is involved in CNN's development [8, 5, 9]. Important variables such as the number of parameters as well as the architectural complexity, usually measured in MAdds, take an important impact on the final model's performance strongly related to the application context. NAS's transition to the multiobjective arena has been slow. Recent research has focused on more classical Pareto-based algorithms. However, algorithms such as indicator-based MOEAs or Decomposition remain unexplored and a promising niche for research.

Therefore, it is observed that solutions representation for NAS is critical for the algorithm itself and its searching operations. It is necessary to create new forms for solutions representation that are flexible and can also allow the evolution of hyperparameters in the evolutionary search.

4.2 Research Questions

- Do different abstraction levels for solutions representation and population dynamics in multiobjective evolutionary algorithms positively affect Neural Ar-

chitecture Search for significantly less complex and highly accurate Convolutional Neural Networks?

- Does Multiobjective Evolutionary Algorithms improve Neural Architecture Search for significantly less complex and highly accurate Convolutional Neural Networks?
- What kind of multiobjective evolutionary paradigms can lead to less complex and highly efficient CNN architectures?

4.3 Hypothesis

Solutions representation in evolutionary algorithms expresses the elements involved in convolutional neural networks, such as layer operations and hyperparameters; that in combination with the searching operations and population dynamics in multiobjective evolutionary algorithms would improve neural architecture search in terms of at least two objectives: accuracy and multiply-add operations.

4.4 General objective

To design, develop, and evaluate a solution representation for convolutional neural networks in conjunction with a multiobjective evolutionary algorithm that successfully targets image classification tasks in order to achieve competitive performance in terms of accuracy and Multiply-Adds operations in comparison to state-of-the-art solutions

4.5 Specific objectives

- To develop a strategic solution representation of all the elements involved in CNN's architecture, including Operations and Hyperparameters.
- To design evolutionary operators according to the proposed solutions representation and to determine the application strength of each one.
- To develop a new neural architecture search algorithm based on a multiobjective evolutionary algorithm for the automatic design of convolutional neural networks.
- To evaluate and to validate the performance of the proposed NAS algorithm concerning other popular state-of-the-art evolutionary NAS.

4.6 Scope and Limitations

The main focus of this proposal is to develop a multiobjective evolutionary algorithm for the automatic design of CNN architectures to target image analysis tasks such as image classification. It is also considered to extend the application arena for the proposed NAS algorithm to image inpainting and denoising problems.

4.7 Expected Contributions

- A new representation for evolutionary algorithms in order to automatically design compact and efficient CNNs.
- A multiobjective evolutionary algorithm for the automatic design of convolutional neural network architectures that achieve competitive performance in terms of accuracy and multiply-adds operations.
- An in-depth study of MOEA characteristics such as representation, population dynamics, and operators' influence on CNN architecture design.

4.8 Methodology

The proposed research methodology is as follows.

- To analyze different proposed solutions representations at different abstraction levels for CNNs encoding.
- To develop a new representation for CNNs architectures. It is considered to explore mixed solutions representation and therefore ad-hoc evolutionary operations while exploring population dynamics in multiobjective evolutionary algorithms.
- To discriminate among CNNs architectures how their internal structures work on extracting data features. Thus, essential blocks at different abstraction levels can be determined. Their granularity from fine blocks implementing single operations like convolution or pooling to coarse blocks with multiple encapsulated operations such as Resblock [7] performing several chained operations are considered.
- To analyze the state of the art NAS proposals based on multiobjective evolutionary algorithms.

- To assess NAS paradigms based on multiobjective evolutionary algorithms in their different forms such as Pareto-based, indicator-based, or decomposition-based algorithms to build on the proposed NAS algorithmic approach.
- To develop a NAS algorithm that integrates the proposed solutions representation and ad-hoc evolutionary operations and population dynamics in a multiobjective evolutionary algorithm.
- To thoroughly evaluate the proposed NAS algorithm. A number of specific datasets for image tasks will be targeted. A number of design and development cycles are considered in order to achieve competitive performance.

4.9 Work Plan

Table 2 shows a preliminary work plan. Every year is divided in semesters. We are considered 3 designs and development phases for the proposed NAS based on MOEAs approach.

4.10 Publications Plan

- Can real-based encoding improve Multiobjective Evolutionary Neural Architecture Search?, **The European Conference on Genetic Programming (EuroGP)**
- Conference/journal oriented paper with partial results, **Genetic and Evolutionary Computation Conference (GECCO), MIT Evolutionary Computation**.
- Journal paper, **IEEE transactions on evolutionary computation**

Table 2: Proposed Schedule (Green done)

Activities	2021		2022		2023		2024	
	S1	S2	S1	S2	S1	S2	S1	S2
Background								
Analysis of the state of the art								
Doctoral proposal								
Preparation								
Defense								
First Iteration								
Analysis and selection of NAS algorithms								
CNNs analysis								
Representation analysis								
NAS algorithm design								
NAS algorithm implementation								
NAS algorithm evaluation								
Conference paper Writing								
Second Iteration								
NAS algorithm design								
NAS algorithm implementation								
NAS algorithm evaluation								
Journal Paper Writing								
Third iteration								
NAS algorithm final design								
NAS algorithm implementation								
NAS algorithm evaluation								
Journal Paper Writing								
Thesis								
Writing								
Dissertation defense								

5 Preliminary Results

The following are the preliminary results achieved during the first year of research. The main idea was to design two levels of encoding representation for CNN's. This is an initial version of the proposed algorithm called CGP-NAS (Cartesian Genetic Programming-Neural Arquitecture Search).

At a first level, the Cartesian Genetic Programming (CGP) representation through acyclic graphs allowing feed-forward connections is used and represented using integer-based vectors. At a second level, CGP's based represented solutions are converted to real-based vector encoding. Multiobjective evolutionary search is performed in the continuous domain, so evolutionary operations are accordingly defined.

5.1 Solutions representation for CNN architectures

This study is based on CGP's solution representation for CNNs at the genotypic level for later transformation to a real domain vector following Clegg et al. research [46]. The search space is determined by blocks that define functions and their parameters. In Table 3, the function set used in these preliminary results is presented. Convolution and pooling standard functions in CNNs are the main ones in the function set [7, 47, 48, 6].

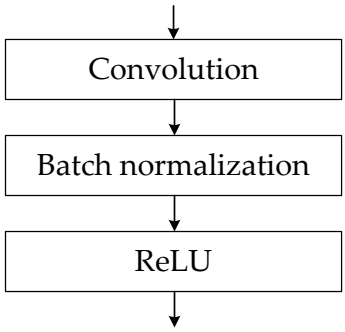
Table 3: Functions set with corresponding variations and arity. *For $C = 32$ there is not 1×1 variation

Block type	Symbol	Variation	Arity
ConvBlock	$CB(C, k)$	$C \in \{32^*, 64, 128\}$ $k \in \{1 \times 1, 3 \times 3, 5 \times 5\}$	1
ResBlock	$RB(C, k)$	$C \in \{32, 64, 128\}$ $k \in \{3 \times 3, 5 \times 5\}$	1
Max Pooling	MP	-	1
Average Pooling	AP	-	1
Summation	Sum	-	2
Concatenation	Concat	-	4

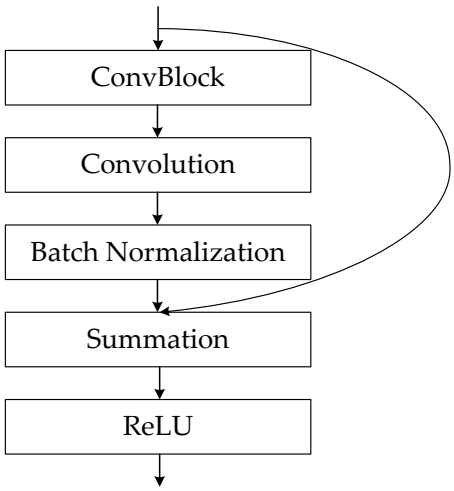
Figure 10a shows the ConvBlock based on three operations: convolution, batch normalization, and ReLU. Input data for ConvBlock and the other blocks is feature maps defined by rows, columns, and channels. Figure 10b draws the ResBlock, which implies a more complex set of operations as it involves a shortcut connection based on the ResNet network [49]. This shortcut connection divides an input by two and

applies a tensor summation. The ConvBlock and ResBlock have different settings that depend on the channel's output and used kernel.

The pooling operation is configured with a 2×2 receptive field and a stride of two. This function is for input down-sampling and works the same for Max and Average pooling. The difference lies in considering the average or maximum values of the receptive field.



(a) ConvBlock has three operations.



(b) ResBlock has a ConvBlock plus five operations. It reduces the vanishing gradient problem.

Figure 10: ConvBlock and ResBlock general structure .

Summation and concatenation inputs are tensors that can be of different sizes. A summation operation is element-wise and channel by channel. Thus, it receives two tensors size $M_1 \times N_1 \times C_1$ and $M_2 \times N_2 \times C_2$.

Max pooling is applied to the larger sized tensor to try matching dimensions, if they are still different, the smaller tensor is expanded with zeros in the channel dimension, therefore the output is $min(M_1, M_2) \times min(N_1, N_2) \times max(C_1, C_2)$.

For concatenation, four feature maps are joined by the channel dimension. Max pooling is applied to the largest input to match its size. Thus, the output size is $min(M_1, M_2) \times min(N_1, N_2) \times (C_1 + C_2)$. In this study, concatenation's arity was set to 4. It receives 4 feature maps, unlike concatenation in [7], that only receives 2 feature maps. First, it concatenates two inputs: $min(M_1, M_2) \times min(N_1, N_2) \times (C_1 + C_2)$ and $min(M_3, M_4) \times min(N_3, N_4) \times (C_3 + C_4)$. After, Max pooling is applied to the largest feature map to match the entries, thus obtained outputs are concatenated

following the same rules. The final output is: $min(min(M_1, M_2), min(M_3, M_4)) \times min(min(N_1, N_2), min(N_3, N_4)) \times ((C_1 + C_2) + (C_3 + C_4))$.

5.1.1 Solutions encoding - decoding

The encoding - decoding process from integer-based to real-based vectors is detailed next. Equation 11 defines a range in which $func_k$ is the current function identifier and $func_{total}$ is the total number of functions in the function set. For this range, a uniform random number is generated to represent a function in the real domain.

$$rfunc_k \in \left[\frac{func_k}{func_{total}}, \frac{func_k + 1}{func_{total}}\right] \tag{11}$$

Equation 12 defines a range for function inputs in the real domain with the objective of mapping connections for that node, this operation is applied to all connections. An input value ($nodeinput$) and its node number ($nodeTerm$) are used to calculate this range. In Figure 2, every node is identified by a node number (red circle on top each block).

$$rinput_j \in \left[\frac{nodeinput_j}{nodeTerm}, \frac{nodeinput_j + 1}{nodeTerm}\right] \tag{12}$$

Equation 13 decodes the function identifier by multiplying the gene value by the total number of functions. On the other hand in Equation 14 the value of each connection is obtained by multiplying the gene value by the node number. It should be noted that the real-based or integer-based vector must be separated by segments. The most left position per segment represents the function and the other values represent the connections.

$$func_k = \lfloor gene_i \times func_{total} \rfloor \tag{13}$$

$$input_j = \lfloor gene_i \times NodeTerm \rfloor \tag{14}$$

Figure 11 shows a an example where the Equations 11 and 12 mentioned above were applied, two genotypes are observed, the top one integer-based and the bottom one converted to real-based, in this example nine total functions were considered.

After encoding, a real-based vector is obtained and its size is calculated as follows:

$$dV = (nT + O) * (mA + 1) \tag{15}$$

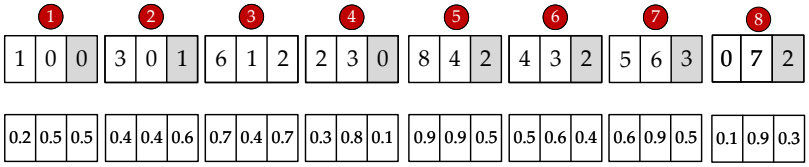


Figure 11: Conversion example between an integer-based solution representation in CGP and its real-based representation by applying the functions defined above.

where nT is the total number of nodes, O is the number of outputs and mA is the maximum arity in the function set. Considering this calculated vector size a new encoded population is determined. Equation 15 calculates dV as the number of decision variables to optimize by CGP-NAS.

5.2 Evolutionary searching engine

For the multiobjective evolutionary neural architecture search through the solutions encoding using real-based vectors, the well established NSGA-II is deployed [20].

Using continuous domain solutions representation opens the possibility to other evolutionary and bio-inspired multiobjective optimization algorithms. In this study, as an initial approach and due to its high algorithmic performance in the multiobjective arena, NSGA-II was considered as the evolutionary searching engine. The NSGA-II algorithm was only modified by adding the encoding and decoding operations. Figure 12 shows the general scheme of NSGA-II, and the steps for encoding and decoding the solutions were added, the following steps are those of the original algorithm. Therefore, NSGA-II is applied in its full capacity with all its advantages. Moreover, even though the original CGP only considers applying mutation and does not recommend crossover. Once, solutions are encoded in the continuous domain, SBX(simulated binary crossover) crossover operation and polynomial mutation, that can be considered as default operations in NSGA-II, can be applied. The objective functions seek to minimize the classification error rate as well as the model's complexity measured in MAdds (Multiply-adds).

5.3 Experimental settings

This section describes the benchmark datasets used for empirical evaluation and results validation together with the experimental algorithmic settings that are defined for a fair comparison to other previously proposed approaches.

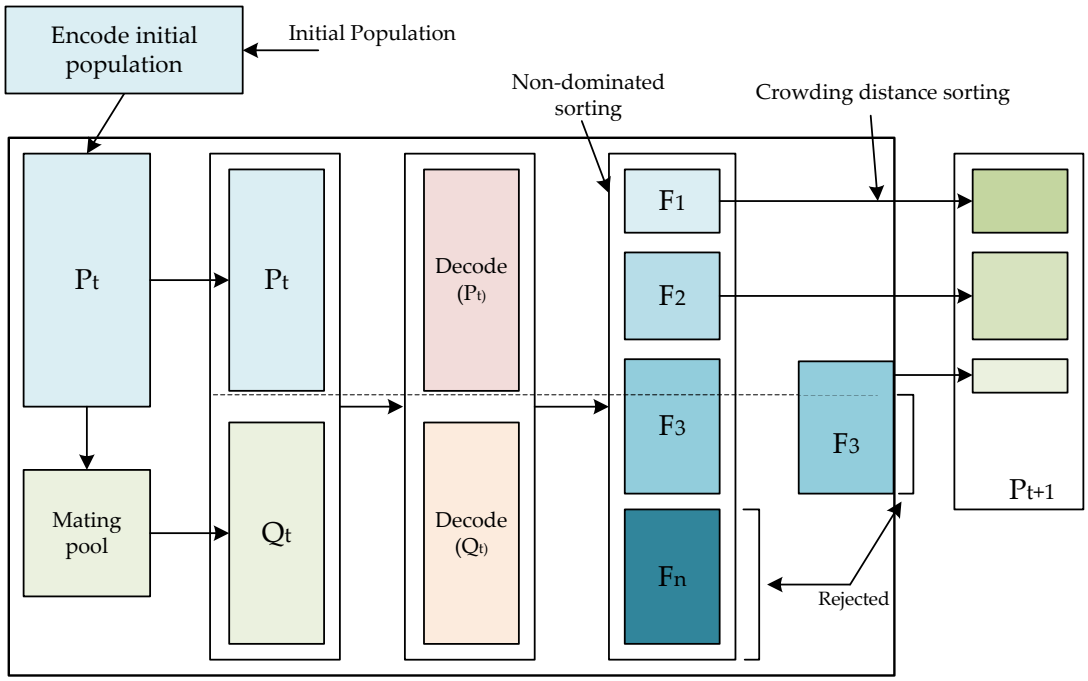


Figure 12: NSGA-II general scheme with encoding and decoding steps. NSGA-II generates an initial population Pt and after mating a population of descendants Qt is obtained. Qt is decoded for evaluation, and therefore for optimization considering non-dominance. Finally, to complete the population crowding distance criterion is applied to reorder solutions going into the next generation.

5.3.1 Benchmark datasets

CIFAR-10 and CIFAR-100 datasets were used for empirical evaluation of the proposed CGP-NAS approach. Both benchmarks have been widely used within the neural architecture search arena. Each dataset has 60,000 images that were divided in two subsets. One with 50,000 as the training set and one with 10,000 as the testing set. This configuration is only used for final best evolved models in order to obtain their final accuracy. For solutions evaluation during the evolutionary searching, the training subset was randomly divided in 45,000 images for training and 5,000 images for validation of every individual solution during evolution.

5.3.2 Experimental settings

Table 4 shows NSGA-II configuration as well as the parameters values used for CGP configuration. For training and validation of every solution within the pop-

Table 4: CGP-NAS parameters

Parameters	Value
CGP grid size $N_r \times N_c$	$\mathbf{10 \times 20}$
Chromosome length	**1005**
Mutation probability	$Pm = \frac{1}{n}$, where n is the decision vector dimension.
Crossover probability	$Pc = \mathbf{0.9}$, distribution index for simulated binary crossover $Dsc = \mathbf{20}$.
Population size	**20**
Number of Generations	**10** for CIFAR 10 Dataset **20** for CIFAR 100 Dataset

ulation during the evolutionary searching process the following configuration was set: Stochastic Gradient Descent (SDG) was used as optimizer and cosine annealing learning rate schedule. The initial learning rate was set to 0.025, while the momentum was set to 0.9 and the weight decay to 0.0005. A batch size of 128 was defined. In addition, 25 training epochs were established.

On the other hand, training and testing data were preprocessed by a 4 pixel-mean subtraction padding on each size and randomly cropped by a 32×32 patch or its horizontally flipped image. CGP-NAS was implemented in python using the pymoo [50] and pytorch libraries and was run on Ubuntu 18.04 machine with an Intel i7-7700 4.2Ghz CPU, 16GB RAM and an NVIDIA GeForce GTX TITAN X GPU card. After an error during an individual training or evaluation, it was assigned as incorrect so that the evolutionary algorithm discards it from the search.

Once the algorithm executes the predefined number of generations, a solution from the achieved Pareto front with the smaller classification error is chosen. This solution is retrained for 500 epochs under the same scheme using the complete dataset in addition the cutout preprocessing technique is added.

5.4 Preliminary Results analysis

CGP-NAS was executed 10 independent times on each dataset. Tables 5 and 6 show the achieved experimental results in terms of classification error, the number of learnable weight parameters, MAdds, GPU-days, and the used GPU module. The best solution classification error found from the 10 independent runs is reported, as well as the average and standard deviation. We classify the results in human-design CNNs and automatically designed CNNs based on single-objective or multi-objective approaches. The "-" symbol means that the corresponding results were not reported

in these proposals. On the CIFAR-10 dataset, CGP-NAS performs better than the human-design reported models in terms of classification error, while the number of parameters is competitive with the other proposals. For the single-objective approaches, CGP-NAS outperforms CGP-CNN[7], Large-Scale Evolution[51], and Genetic-CNN[4]. However, AE-CNN[37] overcomes the proposed approach in this metric, yet CGP-NAS outperforms AE-CNN in terms of GPU-days: 1.4 days versus 27 reported for AE-CNN. Finally, compared to other multi-objective proposals, NSGANet obtains a CNN architecture with lower classification error, yet CGP-NAS obtains an improved architecture complexity (MAdds). CGP-NAS outperforms all other approaches in GPU-days metric. For CIFAR-100 dataset, 10 independent runs were also executed. CGP-NAS achieved results are shown in Table 6. CGP-NAS was able to evolve CNN architectures with competitive performance in terms of classification error. An important reduction in terms of GPU-days is appreciated, 2.1 GPU-days for CGP-NAS while 10.9 and 27 GPU-days were reported for CGP-CNN(ResSet) and NSGANetV1 respectively.

Table 5: Comparison on CIFAR-10 dataset: Classification error rate, the number of parameters and Multiply-adds (MAdds) are expressed in millions (1×10^6), GPU-days and GPU Hardware.

Model	Error rate %	Params	MAdds	GPU-Days	Hardware
Human Design					
DenseNet ($k = 12$) (Huang et al., 2017)	5.24	1.0	-	-	
ResNet ($depth = 101$) (He et al., 2016)	6.43	1.7	-	-	
ResNet ($depth = 1202$) (He et al., 2016)	7.93	10.2	-	-	
VGG (Simonyan et al., 2014)	6.66	20.04	-	-	
Single Objective Approaches					
CGP-CNN (ConvSet) (Suganuma et al., 2020)	5.92	1.50	-	8	Nvidia 1080 Ti
CGP-CNN (ResSet) (Suganuma et al., 2020)	5.01	3.52	-	14.7	Nvidia 1080 Ti
Large-Scale Evolution (Real et al., 2017)	5.4	5.4	-	2750	-
AE-CNN (Sun et al., 2020)	4.3	2.0	-	27	Nvidia 1080 Ti
Genetic-CNN (Xie et al., 2017)	7.1	-	-	17	-
Multi-Objective Approaches					
NSGANet (Lu et al., 2019)	3.85	3.3	1290	8	Nvidia 1080 Ti
MOCNN (Wang et al., 2020)	4.49	-	-	24	Nvidia 1080 Ti
MOGIG-Net (Xue et al., 2021)	4.67	0.2	-	14	-
CGP-NAS	**4.86 (5.42 ± 0.46)**	**1.40**	**388.71**	**1.4**	**Nvidia Titan X**

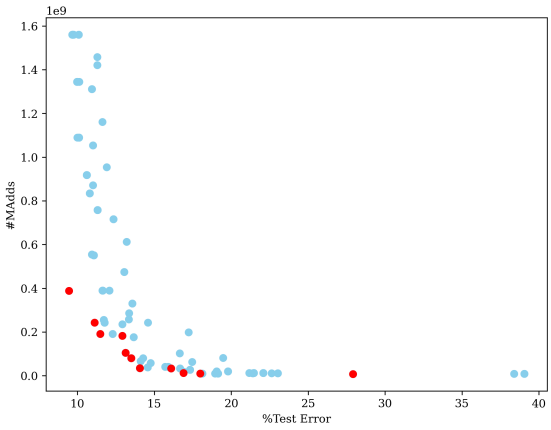
For the CIFAR-100 dataset, CGP-NAS outperforms human-design architectures; while in comparison to the single-objective approaches, CGP-NAS outperforms both CGP-CNN and Genetic-CNN approaches. Considering Large-scale Evolution and AE-CNN, the results are competitive, yet in terms of GPU-days, CGP-NAS overcomes both approaches while requiring only 2.1 GPU-days against 2750 and 36 GPU-days respectively. Comparison against multi-objective approaches, our proposal outperforms MOGIG-Net[44] and NSGANetV1[5] in the classification error, while achieving a very similar performance in MAdds. Finally, CGP-NAS shows

superior performance on this dataset in terms of GPU-days.

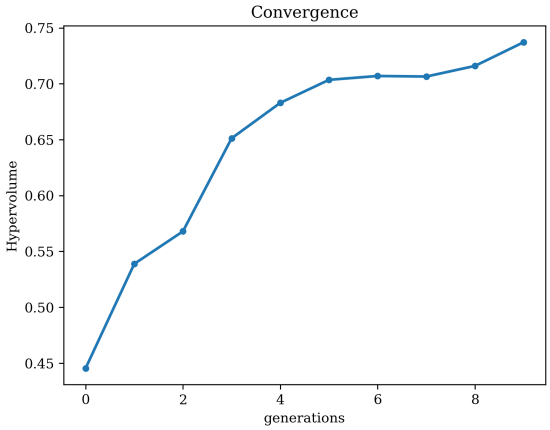
Table 6: Comparison on CIFAR-100 dataset: Classification error rate, the number of parameters and Multiply-adds (MAdds) are expressed in millions (1×10^6), GPU-days and GPU Hardware.

Model	Error rate %	Params	MAdds	GPU-Days	GPU hardware
Human Design					
DenseNet ($k = 12$) (Huang et al., 2017)	24.42	1.0	-	-	
ResNet ($depth = 101$) (He et al., 2016)	25.16	1.7	-	-	
ResNet ($depth = 1202$) (He et al., 2016)	27.82	10.2	-	-	
VGG (Simonyan et al., 2014)	28.05	20.04	-	-	
Single Objective Approaches					
CGP-CNN(ConvSet) (Suganuma et al., 2020)	26.7	2.04	-	13	Nvidia 1080Ti
CGP-CNN(ResSet) (Suganuma et al., 2020)	25.1	3.43	-	10.9	Nvidia 1080Ti
Large-Scale Evolution (Real et al., 2017)	23.0	40.4	-	2750	-
AE-CNN (Sun et al., 2020)	20.85	5.4	-	36	Nvidia 1080 Ti
Genetic-CNN (Xie et al., 2017)	29.03	-	-	17	-
Multi-Objective Approaches					
NSGANetV1 (Lu et al., 2020)	25.17	0.2	1290	27	Nvidia 2080 Ti
MOGIG-Net (Xue et al., 2021)	24.71	0.7	-	14	-
CGP-NAS	**24.23 (26.41 ± 1.41)**	**5.43**	**1581**	**2.1**	**Nvidia Titan X**

The overall performance of CGP-NAS according to the achieved results shows that it can find CNN's architectures with a competitive trade-off between the classification error and the complexity (Madds). In addition, the time required to perform the search is significantly reduced in comparison to other state-of-the-art approaches. The reason for the GPU-days reduction is related to the evolutionary search since CGP-NAS pursues the architecture complexity in terms of MAdds as an objective to minimize. Thus, the complexity of the evolved CNN architectures is reduced. It is also expected that the evaluation time will be reduced, and as a consequence, the overall search time decreases. This happens more efficiently due to the real-based representation used in CGP-NAS because it accelerates the convergence in the early stages of the evolutionary search [46] in combination with NSGA-II, a well-established and high performing MOEA. Moreover, CGP-NAS good performance can also be attributed to the base of CGP representation. The mesh used in this experiment was set to 10x20. This does not mean that all solutions will have a fixed size because the phenotypes obtained by CGP are of variable size. Therefore, the freedom to search for less complex solutions is more feasible likely since it does not have a fixed size. Figures 13a and 13b show the Pareto front and the convergence curve for the Hypervolume metric (average for all executions) on the CIFAR-10 dataset. It is observed that the evolutionary process pursues the best possible trade-off between both objectives, classification error and architecture complexity. On average, the hypervolume metric has an ascending value and this is indicative of overall good distribution, and a well-distributed solution in the objective function space is a good indicator.



(a) CGP-NAS achieved Pareto front for CIFAR-10 dataset.

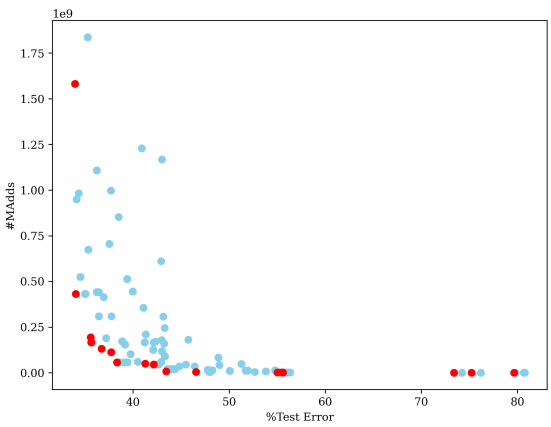


(b) Hypervolume convergence curve achieved for CGP-NAS on CIFAR-10.

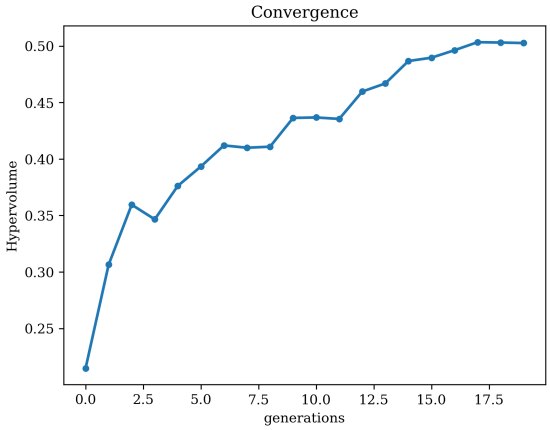
Figure 13: CGP-NAS achieved (a) Pareto front (red dots) and (b) convergence curve for the Hypervolume metric. A set of moderately distributed solutions is obtained while they are dispersed to get as close as possible to the Pareto front.

On the other hand, for the CIFAR-100 dataset, Figures 14a and 14b show the evolved solutions through the generations and the final Pareto fronts and the convergence curve for the hypervolume metric. It is observed that the Pareto front has in general a good solutions dispersion throughout the objective function space. In terms of hypervolume, the obtained curve shows a good performance through the generations. This behavior confirms that using real-based vectors for solution representation can indeed attain similar or even better performance results in the NAS. A general remark for CGP-NAS when targeting CIFAR-100 data set is its significant improvement in terms of GPU-days that are required to evolve competitive solutions. CGP-NAS configuration due to the computational constraints was set for a small population size with a reduced number of generations. Yet, CGP-NAS achieved competitive results and significantly overcame those in terms of the GPU-days. It is possible to confirm that NSGA-II as a NAS searching engine, with its real-based solution encoding as well as specialized operators, helps to improve the convergence process, while coherence in those evolved solutions is maintained due to CGP flexibility.

Figures 15a and 15b shows the best CNN architecture found for both datasets. The best-evolved solution shares many elemental blocks with those found in ResNet networks. For example, using ResBlocks that allow shortcut connections, increases the number of channels according to the architecture deep, or using pooling blocks for downsampling, these features are widely present in CNN's developed by experts.



(a) CGP-NAS achieved Pareto front for CIFAR-100 dataset.



(b) Hypervolume convergence curve achieved by CGP-NAS on CIFAR-100.

Figure 14: CGP-NAS achieved (a) Pareto front (red dots) and (b) convergence curve for the Hypervolume metric. A set of moderately distributed solutions is obtained while they are dispersed to get as close as possible to the Pareto front.

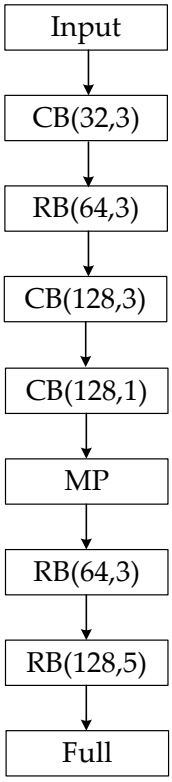
6 Final Remarks

This study proposes the development of a new NAS solutions representation and a algorithm using a multi-objective approach, which has been little investigated from the evolutionary point of view. Given this problem, the first ideas chosen were the CGP representation and the NSGA-II algorithm.

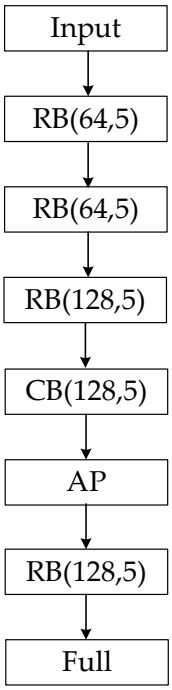
According to the achieved preliminary results, combining solutions representation such as CGP with the searching strength of NSGA-II provides competitive NAS results. Moreover, the flexibility for solution representation in evolutionary algorithms and the searching ability of their multiobjective techniques can be explored further for performance improvement.

Results obtained so far show that compact CNN architectures can be automatically designed while competitive accuracy is maintained. The results also showed that CCNs' architecture design is indeed a multiobjective optimization problem where trying to increase CNN's accuracy while reducing the number of MAdds is clearly in conflict.

Using real-based solutions encoding allows us to make a transition between the integer-based solutions representation used in CGP and a more flexible domain. Thus, it is possible to exploit the benefits of an accurate representation and the searching ability of evolutionary multiobjective algorithms without any external modification by decoding solutions for their evaluation.



(a) Best architecture found on CIFAR-10 Dataset



(b) Best architecture found on CIFAR-100 Dataset

Figure 15: The best evolved CNNs architectures for (a) CIFAR-10 and (b) CIFAR-100 datasets. Both deploy ResBlocks with internal shortcuts connections, ConBlocks and Pooling blocks. See Table 3 for notation details.

The CGP-NAS achieved results are competitive with those from previously proposed approaches. Moreover, CGP-NAS overcame CGP-CNN, MOCNN, NSGANet, and NSGANetV1 in terms of the GPU-days metric. Moreover, CGP-NAS was empirically validated using a small population size, but due to the real-based solutions representation and the NSGA-II as the searching engine, a balanced search for fast convergence was performed, thus having a reduced population and number of generations did not negatively affect finding competitive solutions.

A significant advantage for CGP-NAS is its high-level customization, as it can be adapted to different evolutionary multiobjective algorithms that work in the continuous domain. Some disadvantages of the proposed CGP-NAS are the decision vector size. In this preliminary setting, a 10×20 CGP mesh with maximum arity equals 4, represents a 1005 decision vector size, see Equation 15. This size would increase with respect to the total number of nodes within the CGP mesh. This can

be a drawback for evolutionary algorithms. A possible way to overcome this is to research co-evolution as a way to deal with large decision vectors.

In future work, new blocks will be added to the function set. A starting point could be those used by [5], where more elaborate functions such as dilated convolutions, local binary convolutions, and depth-wise-separable convolutions are proposed. These have shown good NAS performance.

On the other hand, benchmarking will be extended to the ImageNet dataset, which is being used more and more in this area. Also, other image tasks such as image inpainting and denoising will be considered. A starting point for reference would be based on [43], in which CGP was used to search for convolutional autoencoders and competitive results are reported.

References

[1] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, and B. Hodjat, "Evolving Deep Neural Networks," in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pp. 293–312, Elsevier, 2019.

[2] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, and W. Banzhaf, "NSGA-Net," in *Proceedings of the Genetic and Evolutionary Computation Conference*, (New York, NY, USA), pp. 419–427, ACM, jul 2019.

[3] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Evolving Deep Convolutional Neural Networks for Image Classification," *IEEE Transactions on Evolutionary Computation*, vol. 24, pp. 394–407, apr 2020.

[4] L. Xie and A. Yuille, "Genetic CNN," in *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 1388–1397, IEEE, oct 2017.

[5] Z. Lu, I. Whalen, Y. Dhebar, K. Deb, E. Goodman, W. Banzhaf, and V. N. Boddeti, "Multi-Objective Evolutionary Design of Deep Convolutional Neural Networks for Image Classification," *IEEE Transactions on Evolutionary Computation*, pp. 1–1, sep 2020.

[6] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proceedings of the Genetic and Evolutionary Computation Conference*, (New York, NY, USA), pp. 497–504, ACM, jul 2017.

[7] M. Suganuma, M. Kobayashi, S. Shirakawa, and T. Nagao, "Evolution of deep convolutional neural networks using cartesian genetic programming," mar 2020.

[8] Z. Lu, K. Deb, E. Goodman, W. Banzhaf, and V. N. Boddeti, "NSGANetV2: Evolutionary Multi-objective Surrogate-Assisted Neural Architecture Search," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12346 LNCS, pp. 35–51, Springer Science and Business Media Deutschland GmbH, aug 2020.

[9] B. Wang, B. Xue, and M. Zhang, "Particle Swarm Optimization for Evolving Deep Convolutional Neural Networks for Image Classification: Single- and Multi-Objective Approaches," in *Natural Computing Series*, pp. 155–184, Springer, 2020.

[10] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[11] Y. Ho and D. L. Pepyne, "Simple explanation of the no free lunch theorem of optimization," *Kibernetika i Sistemnyj Analiz*, vol. 38, no. 2, pp. 164–173, 2002.

[12] T. Elsken, J. H. Metzen, and F. Hutter, "Neural Architecture Search: A Survey," tech. rep., 2019.

[13] A. Eiben and J. Smith, *Introduction to Evolutionary Computing*. Natural Computing Series, Berlin, Heidelberg: Springer Berlin Heidelberg, 2015.

[14] J. Miller, P. Thomson, T. Fogarty, and I. Ntroduction, "Designing electronic circuits using evolutionary algorithms. arithmetic circuits: A case study," *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science*, 10 1999.

[15] J. F. Miller, "Cartesian Genetic Programming," in *Natural Computing Series*, vol. 43, pp. 17–34, 2011.

[16] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, 2001.

[17] C. A. C. Coello, G. B. Lamont, D. A. V. Veldhuizen, C. A. Coello Coello, G. B. Lamont, and D. A. Van Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems*. Genetic and Evolutionary Computation Series, Boston, MA: Springer US, 2007.

[18] K. Deb, *Evolutionary and Swarm Intelligence Algorithms*, vol. 779 of *Studies in Computational Intelligence*. Cham: Springer International Publishing, 2019.

[19] M. T. M. Emmerich and A. H. Deutz, "A tutorial on multiobjective optimization: fundamentals and evolutionary methods," *Natural Computing*, vol. 17, pp. 585–609, sep 2018.

[20] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," Tech. Rep. 2, 2002.

[21] C. M. Fonseca and P. J. Fleming, "Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization," 1993.

[22] J. Knowles and D. Corne, "The pareto archived evolution strategy: a new baseline algorithm for pareto multiobjective optimisation," in *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, vol. 1, pp. 98–105 Vol. 1, 1999.

[23] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach," *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, vol. 3, no. 4, pp. 257–271, 1999.

[24] E. Zitzler, M. Laumanns, and L. Thiele, "Spea2: Improving the strength pareto evolutionary algorithm," tech. rep., 2001.

[25] Q. Zhang and H. Li, "MOEA/D: A multiobjective evolutionary algorithm based on decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 11, pp. 712–731, dec 2007.

[26] Hui Li and Qingfu Zhang, "Multiobjective Optimization Problems With Complicated Pareto Sets, MOEA/D and NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 13, pp. 284–302, apr 2009.

[27] M. Sonka, V. Hlavac, and R. Boyle, *Image Processing, Analysis and Machine Vision*. Boston, MA: Springer US, 1993.

[28] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[29] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[30] S. Russell and P. Norvig, *Artifical Intelligence: A Modern Approach (Third Edition)*. 3er ed., 2009.

[31] N. Noman, *A Shallow Introduction to Deep Neural Networks*, pp. 35–63. Singapore: Springer Singapore, 2020.

[32] H. J. Escalante, *Automated Machine Learning—A Brief Review at the End of the Early Years*, pp. 11–28. Cham: Springer International Publishing, 2021.

[33] H. Zhu and Y. Jin, *Toward Real-Time Federated Evolutionary Neural Architecture Search*, pp. 133–147. Cham: Springer International Publishing, 2021.

[34] K. O. Stanley and R. Miikkulainen, "Evolving Neural Networks through Augmenting Topologies," *Evolutionary Computation*, vol. 10, pp. 99–127, jun 2002.

[35] D. B. D'ambrosio and K. O. Stanley, *A Novel Generative Encoding for Exploiting Neural Network Sensor and Output Geometry*. 2007.

[36] S. Risi, J. Lehman, and K. O. Stanley, *Evolving the Placement and Density of Neurons in the HyperNEAT Substrate*. 2010.

[37] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Completely Automated CNN Architecture Design Based on Blocks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, pp. 1242–1254, apr 2020.

[38] W. Irwin-Harris, Y. Sun, B. Xue, and M. Zhang, "A Graph-Based Encoding for Evolutionary Convolutional Neural Network Architecture Design," in *2019 IEEE Congress on Evolutionary Computation (CEC)*, pp. 546–553, IEEE, jun 2019.

[39] A. Bakhshi, S. Chalup, and N. Noman, *Fast Evolution of CNN Architecture for Image Classification*, pp. 209–229. Singapore: Springer Singapore, 2020.

[40] Y.-H. Kim, B. Reddy, and S. Yun, "NEMO: Neuro-Evolution with Multiobjective Optimization of Deep Neural Network for Speed and Accuracy Chanwon Seo," tech. rep., 2017.

[41] M. Pelikan, D. E. Goldberg, and E. Cantu-Paz, "Boa: The bayesian optimization algorithm," pp. 525–532, Morgan Kaufmann, 1999.

[42] S. Li, Y. Sun, G. G. Yen, and M. Zhang, "Automatic Design of Convolutional Neural Network Architectures Under Resource Constraints," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, 2021.

[43] M. Suganuma, S. Shirakawa, and T. Nagao, *Designing Convolutional Neural Network Architectures Using Cartesian Genetic Programming*, pp. 185–208. Singapore: Springer Singapore, 2020.

[44] Y. Xue, P. Jiang, F. Neri, and J. Liang, "A Multi-objective evolutionary approach based on graph-in-graph for neural architecture search of convolutional neural networks," *International Journal of Neural Systems*, vol. 31, sep 2021.

[45] J. Liang, E. Meyerson, B. Hodjat, D. Fink, K. Mutch, and R. Miikkulainen, "Evolutionary neural AutoML for deep learning," in *Proceedings of the Genetic and Evolutionary Computation Conference*, (New York, NY, USA), pp. 401–409, ACM, jul 2019.

[46] J. Clegg, J. A. Walker, and J. F. Miller, "A new crossover technique for Cartesian genetic programming," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation - GECCO '07*, (New York, New York, USA), p. 1580, ACM Press, 2007.

[47] M. Pinos, V. Mrazek, and L. Sekanina, "Evolutionary Neural Architecture Search Supporting Approximate Multipliers," pp. 82–97, Springer, Cham, apr 2021.

[48] Y. Sun, H. Wang, B. Xue, Y. Jin, G. G. Yen, and M. Zhang, "Surrogate-Assisted Evolutionary Deep Learning Using an End-to-End Random Forest-Based Performance Predictor," *IEEE Transactions on Evolutionary Computation*, vol. 24, pp. 350–364, apr 2020.

[49] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, IEEE, jun 2016.

[50] J. Blank and K. Deb, "pymoo: Multi-objective optimization in python," *IEEE Access*, vol. 8, pp. 89497–89509, 2020.

[51] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, and A. Kurakin, "Large-Scale Evolution of Image Classifiers," mar 2017.