IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received July 19, 2020, accepted August 20, 2020, date of publication August 25, 2020, date of current version September 8, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3019245

Automating Configuration of Convolutional Neural Network Hyperparameters Using Genetic Algorithm

FRANKLIN JOHNSON[1], ALVARO VALDERRAMA[2], CARLOS VALLE[1], BRODERICK CRAWFORD[3], RICARDO SOTO[3], AND RICARDO ÑANCULEF[4], (Member, IEEE)

[1]Department of Computer Science and Informatics, Universidad de Playa Ancha, Valparaíso 2360002, Chile
[2]Department of Informatics, Universidad Técnica Federico Santa María, Valparaíso 2390123, Chile
[3]Department of Informatics, Pontificia Universidad Catolica de Valparaíso, Valparaíso 2374631, Chile
[4]Department of Informatics, Universidad Técnica Federico Santa María, Santiago 8320000, Chile

Corresponding author: Franklin Johnson (franklin.johnson@upla.cl)

The work of Franklin Johnson was supported by Comisión Nacional de Investigación Científica y Tecnológica (CONICYT)/Fondo Nacional de Desarrollo Científico y Tecnológico (FONDECYT)/Iniciación 11180524. The work of Broderick Crawford was supported by CONICYT/FONDECYT/Regular 1171243 and Ricardo Soto was supported by CONICYT/FONDECYT/Regular 1190129.

ABSTRACT In recent years, Convolutional Neural Networks (CNN) have been widely used for real-world applications in the field of computer vision. Their class-leading performance, however, depends heavily on the architecture used for a given problem. In most cases, the architectures are manually optimized by the researchers, a time-consuming process hard to achieve without prior knowledge of CNN. In this paper, we propose a new genetic algorithm for the optimization of the CNN architecture for a given image classification problem. This algorithm extends and refines existing research in the field, by allowing depth exploration, introducing a novel sequential crossover operator, using an incremental selective pressure schedule over evolution (favoring higher diversity in early generations) and by evaluating individual performances over the validation set with early stopping. The technique is validated in three image classification dataset, namely, CIFAR10, MNIST and Caltech256 datasets, which are widely used benchmarks for image classification algorithms. We evaluate the performance and total execution time over these datasets, and compare our results with those achieved by the best genetic methods published so far. In all cases, we achieve better results in terms of test accuracy, consistently over different datasets, while remaining in the same orders of magnitude of execution time of the fastest approaches.

INDEX TERMS Artificial neural network, genetic algorithm, image classification.

I. INTRODUCTION

Machine learning and, in particular, deep learning have provided state of the art results in varied fields. Image classification is one of the current problems of computer vision where deep learning is widely used. This task, where an algorithm must classify an image into one or several categories without human intervention, is important to several other problems such as object detection or image segmentation and several developments in automation, e.g. automated driving. To tackle this problem, the most widely used architectures are deep CNN [12], [24], which have shown higher performances than traditional models. Since their inception, CNN have improved significantly the state of the art in computer vision in general [26], [33], even attaining human-level performances in some problems [18].

These network's accuracies, however, are strongly related to their structure and design. In fact, some researchers have reported significant performance improvement by finding good choices of these hyperparameters [23]. In the simplest of cases, the list of hyperparameters to optimize includes the number of layers, filter sizes and filter numbers for each layer. Even with only these attributes to tune, the problem is very complex, and there is still no gold standard for finding the best architecture for a given task. In most of the cases, it is done by each researcher by brute force, or based on structures validated in previous work [37].

The associate editor coordinating the review of this manuscript and approving it for publication was Jagdish Chand Bansal.

Literature has proved the usefulness of metaheuristics in solving complex combinatorial problems [8], [13], [31], that would be either exceedingly time-consuming to resolve by traditional optimization methods or are simply impossible to solve directly. Population-based algorithms, such as genetic algorithms [2], perform particularly well on optimization problems with large search spaces and pathological objective functions [4]. There is current research on the use of metaheuristics to optimize CNN's hyperparameters where a metaheuristic is used to tune different properties of the networks, such as kernel size or sequence of convolutional blocks in a particular problem. Sun *et al.* [39] use genetic algorithm to optimize the sequence of *skip blocks* and MaxPool layers, including depth, however, the kernel size of the convolutions is fixed. Other researches have used more simple CNN architectures [3], [30] and genetic algorithms to optimize not only the structure of the network, but the learning parameters as well, which entails comparisons between trained and untrained networks during the process. Most current developments [3], [29], [30] use traditional crossover based on genetic algorithms operators, destroying the structure of the parent networks as offspring population is created, ignoring the nature of the underlying Machine Learning (ML) problem.

In this paper, we propose a method to use metaheuristics that aims to address the concerns described above, in particular the optimization of training hyperparameters and the inheritance of the network structure during crossover. We extend existing research in the subject, by using genetic algorithms to find kernel sizes and filter number for each layer of our network, as well as the optimal number of layers for a given problem. The performance of each network is drastically influenced by these hyperparameters, which define the architecture of the convolutional section of the network and its trainable parameter number. We also introduce a crossover operator that takes into account the structure of each individual, by preserving the filter number and filter sizes of the parents in a sequential manner, allowing parents to inherit better traits to the following generations.

The following sections are organized as follows. We first introduce the structure of the optimization problem in section II as a consequence of the constraints of ML. Then, in section III a short introduction to the architecture of CNN is presented, discussing how this translates into additional constraints to the problem. In section IV, we discuss key concepts of metaheuristics and the particularities of genetic algorithms. Then, in section V, we present an overview of similar works and developments of this type of problem, followed by section VI, where we describe the method we propose for solving the problem. In section VII, we present our experiments and results. Finally, section VIII presents our conclusions and future work.

II. PROBLEM STATEMENT AND BACKGROUND

ML has recently produced many successful tools for modelling and prediction in several fields [9], [27]. Unlike other approaches, ML avoids following strict and preformulated models and rather uses data-driven algorithms that are efficient at choosing which model best represents the relationship between the response and its predictors.

Several learning paradigms have been researched, including, *supervised learning*, where the algorithm learns from labeled data [22]; *unsupervised learning*, where we have access to data without labels and the algorithm has to deduce structures inherent to the data [10]; or *reinforcement learning* where an algorithm is trained to perform a task without prior knowledge of the utility of different possible actions [40].

In this paper, we deal with a task of supervised learning, that can be stated formally as follows. Let $\mathcal{X}$ be the feature space, that is, the space from where the inputs are drawn (generaly a subset of $\mathbb{R}^n$), and $\mathcal{Y}$ the output space, meaning the space from where our target variable is drawn. If we denote by $\mathcal{S}$ the space of all possible sets of pairs $(x, y) \in \mathcal{X} \times \mathcal{Y}$ drawn from an unknown joint distribution $P(x, y)$; and by $\mathcal{H}$ the hypothesis space, *i.e.* the space of all allowed functional solutions $f : \mathcal{X} \longrightarrow \mathcal{Y}$; then a learning algorithm $\mathcal{A}$ is represented by the following mapping:

$$\begin{aligned} \mathcal{A} : \mathcal{S} &\longrightarrow \mathcal{H} \\ S_M &\longmapsto f, \end{aligned}$$

where S_M denotes a data sample and f is called a hypothesis. The aim of the learning algorithm is to find $\mathcal{A}(S_M) = f$ such that an error functional is minimized:

$$\mathcal{A}(S_M) = \underset{f \in \mathcal{H}}{\operatorname{argmin}} \sum_{(x,y) \in S_t} \ell(f(x), y). \tag{1}$$

Here, $\ell : \mathcal{Y} \times \mathcal{Y} \longrightarrow \mathbb{R}_0^+$ is called the *loss function*, a positive function that measures the differences between the predicted and ground-truth targets.

As the example data is finite, a large or complex enough model can eventually "memorize" every data pair available. This is known as overfitting and is a common problem encountered by many ML algorithms. In this case, the selected hypothesis f has near-zero error over the data, but its performance over future data is considerably lower [16]. We say the model is unable to generalize. It is therefore essential to estimate the generalization capacity of the hypothesis chosen by our algorithm. In ML this is often achieved by splitting the original dataset into two smaller sets, one called training set, used by the algorithm to select the hypothesis, and one called validation set, over which the chosen hypothesis is evaluated *a posteriori*. We denote them by S_t and S_v respectively.

One way overfitting can be controlled is by searching over smaller hypothesis spaces, but this will in turn reduce the ability of the algorithm to find increasingly complex relationships in the data, a trade off difficult to balance without further assumptions over P. More formally, we define the risk R of a given hypothesis $f \in \mathcal{H}$ as the expected value of it's loss over

every possible pair x, y drawn following P, that is:

$$R(f) = \mathbb{E}[\ell(f(x), y)] = \int_{\mathcal{X}\times\mathcal{Y}} \ell(f(x), y) dP(x, y).$$

We denote by f^* the hypothesis that minimizes R, also called the Bayes Classifier in the context of classification problems. Evidently, to find f^* is impossible, since P is unknown, and we would need infinite data to find the best fitting hypothesis over all possible $\mathcal{X} \longrightarrow \mathcal{Y}$ functions. Moreover, our hypothesis space is only a subset of the space of all possible functions from $\mathcal{X}$ to $\mathcal{Y}$, and therefore it is possible that $f^* \notin \mathcal{H}$. We may then search for the best hypothesis in our class, let:

$$f_{\mathcal{H}} = \underset{f\in\mathcal{H}}{\operatorname{argmin}}\, R(f).$$

However, in practice it is impossible to find $f_{\mathcal{H}}$ as well, considering we only have access to a finite amount of examples drawn from P and cannot therefore compute the real risk. We therefore must find instead an "Empirical Risk Minimizer", denoted by $\hat{f}_{S_t}$, that is the hypothesis that minimizes the following empirical risk estimator:

$$\hat{R}(f) = \frac{1}{|S_t|} \sum_{(x,y)\in S_t} \ell(f(x), y)),$$

which acts as a proxy or estimator of the real risk.

This means that while the original purpose of the ML algorithm $\mathcal{A}$ should be to minimize the risk functional R, it can only minimize the empirical risk, and therefore it is unlikely that the chosen hypothesis will be truly optimal regardless of the algorithm and the data, since the found hypothesis f will often have a higher risk than f^*. This difference, $\hat{R}(f) - R(f^*)$, is usually named empirical excess risk. Even more, for some hypothesis spaces (such as the case of Artificial Neural Networks for instance), the minimization of the empirical risk can only be done in an iterative stochastic process, so in most cases we are not even able to find $\hat{f}_{S_t}$, and instead, we are only able to find a function $\tilde{f}_{S_t}$ that approximates $\hat{f}_{S_t}$. Therefore our empirical excess risk will be $R(\tilde{f}_{S_t}) - R(f^*)$, denoted ER. We can decompose the excess risk as follows:

$$\begin{aligned} ER &= \hat{R}(\tilde{f}_{S_t}) - R(f^*) \\ &= \hat{R}(\tilde{f}_{S_t}) - \hat{R}(\hat{f}_{S_t}) + \hat{R}(\hat{f}_{S_t}) - R(f^*) \\ &= \hat{R}(\tilde{f}_{S_t}) - \hat{R}(\hat{f}_{S_t}) \\ &\quad + \hat{R}(\hat{f}_{S_t}) - R(f_{\mathcal{H}}) + R(f_{\mathcal{H}}) - R(f^*). \end{aligned} \tag{2}$$

Here we name:

- $\hat{R}(\tilde{f}_{S_t}) - \hat{R}(\hat{f}_{S_t})$ the optimization error, that is the difference between the best possible solution given the data and the solution found by the optimizer,
- $\hat{R}(\hat{f}_{S_t}) - R(f_{\mathcal{H}})$, the estimation error, that is the difference between the best possible solution given the data and the best theoretical solution in the hypothesis space, and finally
- $R(f_{\mathcal{H}}) - R(f^*)$ the approximation error, that is the excess risk that arises from restricting the functional space from where our hypothesis is drawn.

Notice that as the capacity of the hypothesis space $\mathcal{H}$ increases, the approximation error is expected to decrease (simply consider the extreme case where $\mathcal{H} = \{f \,|\, f : \mathcal{X} \longrightarrow \mathcal{Y}\}$), but this greater freedom also tends to increase the estimation error, since given a larger space from where to minimize the empirical risk, the optimizer tends to choose a hypothesis that memorizes every single point in the finite data rather than inferring the underling relationship.

Usually, learning algorithms depend on hyperparameters that change the behaviour and performance of the algorithm, by altering the optimization process or modifying the hypothesis space. The latter corresponds to our case, where changes to the explored hyperparameters represent changes to the hypothesis space where our algorithm draws its hypothesis.

Let Λ be the space of all possible hyperparameters for a given problem, and, for $\lambda \in \Lambda$, let $\mathcal{A}_\lambda$ be the learning algorithm and $\mathcal{H}_\lambda$ the hypothesis space associated to the hyperparameters λ. If we denote by S_t our training set, then, given $\lambda \in \Lambda$, the learning algorithm $\mathcal{A}_\lambda$ selects the hypothesis $f^\lambda_{S_t}$ that minimizes the following:

$$f^\lambda_{S_t} := \mathcal{A}_\lambda(S_t) = \underset{f\in\mathcal{H}_\lambda}{\operatorname{argmin}} \sum_{(x,y)\in S_t} \ell(f(x), y). \tag{3}$$

Now, using the same notation, we can formulate the hyperparameter optimization problem for a given algorithm and dataset as follows:

$$\min_{\lambda\in\Lambda} \sum_{(x,y)\in S_v} \ell\left(f^\lambda_{S_t}(x), y\right), \tag{4}$$

where:

- $f^\lambda_{S_t}$ is the hypothesis chosen from the train set and the hyperparameters, that is $f^\lambda_{S_t} := \mathcal{A}_\lambda(S_t)$, as stated in (3),
- S_v is the validation set.

Here, the total loss over the validation set acts as a proxy for the risk. Considering that in our case, the hyperparameters considered only affect the hypothesis space rather than the optimization algorithm, we can consider that our task is to find a value of the hyperparameters that yields the best trade off between the estimation and the approximation error, considering the optimization error an additional noise of stochastic nature for our setup.

Considering this optimization problem, we must overcome two main drawbacks. The first one is the lack of a known relationship between the hyperparameters λ and the objective function, meaning there is no *a priori* way of knowing in which direction a small change of λ will affect the objective function. The second problem is the fact that most (if not all) [28] the currently used training algorithms for neural networks have a strong stochastic component. This means that not only we know no relationship between λ and our objective function, but also that observed local relationships can be the result of the random elements involved in the training process.

From this section we can see that the hyperparameter optimization problem defined by Eq. 4 in the context of

supervised learning poses a non trivial trade off, as implied by the risk decomposition presented in Eq. 2. To tackle this problem while considering the two drawbacks presented in the last paragraphs, we opted to use metaheuristics, in particular genetic algorithms (GA) which handle well this type of difficulties.

The following section presents a more detailed discussion of the hyperparameters involved in CNN. These hyperparameters modify the behavior of the learning algorithm and the hypothesis spaces involved, which makes a good choice of them paramount to finding an algorithm with good generalization capacity. Moreover, our proposal is tailored specifically for this family of learning algorithms and therefore the relationship between the hyperparameters and the structure of the underlying CNN is essential to our proposal.

III. CONVOLUTIONAL NEURAL NETWORKS

In ML, Artificial Neural Networks (ANNs) refer to a family of learning algorithms and their corresponding hypothesis space. Initially inspired in the structure of natural neural networks found in the cerebral cortex of animals, today there is a large variety of different algorithms and approaches to develop and train ANNs architectures specialized in different tasks [1].

Particularly, in the field of image recognition, CNN show promising results [26], in some cases attaining or even surpassing human-level performance [18]. CNN achieve these results in part thanks to the particular connectivity pattern between its neurons, inspired originally in the organization of the animal visual cortex [19], where different patterns (*e.g.* vertical lines, horizontal lines) activate different neural pathways. CNN replicate this behavior by using convolutional filters, or kernels. A kernel can be understood as a moving frame, that sweeps the image, weighting each pixel according to the weights of the kernel. By using fixed weights, the kernel extract the same output when a pattern is repeated in the image.

In the discrete case, the convolution between two real valued functions f and g, defined over the integers $\mathbb{Z}$, is denoted by $f * g$ and is defined as follows:

$$\begin{aligned}(f * g) : \mathbb{Z} &\longrightarrow \mathbb{R} \\ n &\longmapsto \sum_{m=-\infty}^{\infty} f(m)g(n-m).\end{aligned} \tag{5}$$

Definition (5) is naturally extended to the 2-dimensional case, where each f and g map $\mathbb{Z}^2$ to $\mathbb{R}$ by:

$$\begin{aligned}(f * g) : \mathbb{Z}^2 &\longrightarrow \mathbb{R} \\ (i,j) &\longmapsto \sum_{(p,q)\in\mathbb{Z}^2} f(p,q)g(i-p,j-q).\end{aligned} \tag{6}$$

While this definition involves an infinite sum, in practice this is not the case. Suppose we define g to be our convolutional kernel, then g has a finite support, that is the subset of $\mathbb{Z}^2$ over which g is non-zero is finite, and therefore the convolution will be determined only by the terms of the sum where g is non-zero. The same applies for the image, where its support is given by the dimensions, and therefore it is finite. If we represent our image by f, we can compute the output of a single filter defined by the kernel g by:

$$y(i,j) = \sigma\left[(f * g)(i,j) + b_0\right], \tag{7}$$

where σ is called the activation function (a real, usually non-linear, function) and b_0 is called the bias, a trainable constant. The output filter y is also 2-dimensional, and generally the input image f is *padded*, that is extended by 0, in such a way as to preserve the original dimensions as the image is filtered. In the case the input image f has multiple channels, for instance $k \in \mathbb{N}$ channels denoted by $f_1, \ldots, f_k$, k kernels $g_1, \ldots, g_k$ are defined and their corresponding outputs added as follows:

$$y(i,j) = \sigma\left(\sum_{l=1}^{k}(f_l * g_l)(i,j)\right) \tag{8}$$

After processing the whole image, we have a new image, called a feature map. Now each pixel is the combined output of the original image's pixels trough the kernel operation. The weights of the kernel operation or convolution, are learned by the network during the training process. This approach allows CNN filters to identify repeated patterns independently of their position inside the image, while also reducing the number of parameters when compared to fully connected architecture, helping to avoid overfiting.

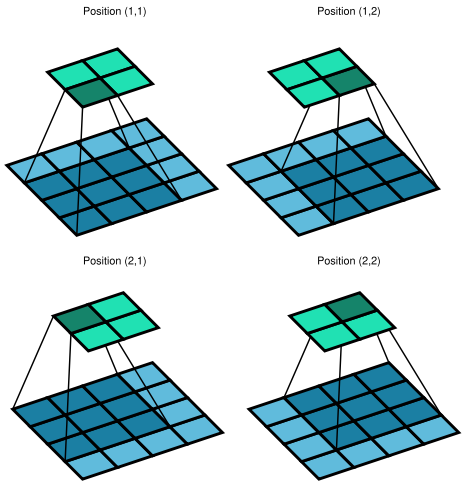

FIGURE 1. Example of the computation of a convolutional filter over an image. Since there is no padding, the filtered image on the top looses dimentionality. The labels "Position (i,j)" denote the position of the corresponding value in the filtered image. The filter in position (i,j) outputs the value of the operation $y(i,j)$ defined in Eq. 8. This value then corresponds to the pixel of coordinates i,j in the filtered image.

To then construct a layer of a CNN, several convolutional filters operate over the image, and the resulting filters are stacked. The kernels are initialized with random weights, and the network adjusts them during training, learning to extract different patterns relevant to the problem at hand. The diversity of kernels allows, in theory, the network to learn

different patterns over the image simultaneously. To construct a full CNN, several layers are then concatenated, applying the kernels of one layer over the output of the preceding layer, training all of them at once. By applying successive layers of kernels, the network can learn to extract higher-level feature maps by refining patterns sequentially. For instance, one could imagine a network that identifies rectangles to use the first layer to detect the presence of edges, the second one to identify where edges met and the last one to identify the presence of 4 joined edges.

However, for the task of object recognition or image classification, feature extraction and pattern recognition is often insufficient. There are 2 main types of layers commonly used to complement convolutional layers. Usually, one or more fully connected layers are added at the end of the network [23]. These allow combining the features extracted from the images in a way that correctly predicts the target. However, dense layers have many parameters when their inputs have a high dimension. Given that the smaller images used in image recognition are usually 32×32 RGB pixels [7] (totaling 3072 values), it is important to reduce their dimensionality before connecting the dense layers. This is why Max Pooling layers are commonly added to the networks [35], extracting the relevant information of its input layer, while at the same time reducing its dimensionality.

For a fixed CNN structure, our hypothesis space $\mathcal{H}$ is defined by all the possible combinations of real values for each parameter of the model. That is, weights and biases for each kernel in every convolutional layer, as well as biases and weights for the dense layers. Our learning algorithm has then to find, ideally, the best combination of all those values, for a given problem, with a finite number of examples. The number of parameters to set in some large networks can reach hundreds of millions [36], so finding even a good combination of values is a complex task.

The usual training process starts by initializing the weights of the whole network randomly [14]. Then, iterative methods based on stochastic gradient descent are used to adjust the parameters, aiming to minimize the total error of the network over the train set. Given the loss function, the gradient for a given train set and parameters can be computed and used to update the weights, following one of several optimization algorithms proposed in the literature [5], [21], [42]. At each iteration, the weights are updated, and, with these values, the error over the validation set can be computed and evaluated.

The convolutional layers, the max pooling, and dense layers are the basic building blocks for CNN. We focus on optimizing the structure of these series of blocks, particularly the following hyperparameters.

- Depth: The depth of the network is defined as the number of layers of the network. This has serious repercussions in the ability of the algorithm to correctly learn the task, since a too shallow network is unable to learn the complex patterns that represent the categories of images and a too deep network has too many parameters and overfits easily.
- Number of filters: It is also necessary to decide for each convolutional layer, the number of filters or neurons. This hyperparameter is crucial to allow the transmission of information to the deeper layers, as a too-small number of filters can lead to significant information loss in that particular layer. Again we must account for overfitting when augmenting this value.
- Kernel size: Also for each convolutional layer, we must decide the size of the convolution or kernel. For consistency reasons, all the kernels in a layer are the same size, however, the optimal size is dependent on the problem and the structure of the rest of the network. This parameter is crucial to allow the network to learn the patterns in their corresponding scales.

We intentionally left out several other possible hyperparameters, such as the number of dense layers, number of neurons in each dense layer or number of max-pooling layers. We consider that these hyperparameters can be left to a default value and the convolutional section of the network will allow it to learn the inherent features. Also, these hyperparameters relate strongly to the dimensionality of the ensuing hypothesis spaces, and therefore need to be treated delicately in order to avoid overfitting.

Even ignoring the aforementioned hyperparameters, our optimization problem is complex, considering not only a large search space but also the fact that for different values of the depth hyperparameter, the number of hyperparameters to optimize varies. As described in section II, the choice of hyperparameters influences both the behaviour of the learning algorithm and the hypothesis space it explores, altering the generalization capacity of the algorithm in non trivial ways. We must also take into account that CNN are complex stochastic learning algorithms that take considerable time to train, making an exhaustive search in practice impossible, and introducing an important stochastic nature to the performances achieved by each network. These characteristics of the problem make metaheuristics appealing to search for CNN structures which maximize the generalization capacity induced by the choice of hyperparameters. We describe these techniques in the following section.

IV. METAHEURISTICS

Metaheuristics constitute an active field of research [11] that also interacts successfully with many other fields of research. In particular, several approaches where metaheuristics are used to optimize the different components of neural networks have been proposed [32]. In particular, genetic algorithms have been used in cooperation with neural networks in several works. For instance, the authors in [6] use a genetic algorithm to improve the convergence speed and error rate of a three layer artificial neural network, while in [41] the authors present a method that utilizes a genetic algorithm to extract a sub-network from a CNN with negligible loss of accuracy. In the following we will introduce a few concepts related to metaheuristics in general and the combinatorial

optimization problem, to follow with a brief description of general genetic algorithms.

In combinatorial optimization, solving algorithms must be able to move efficiently through the solution space of a given problem. If the algorithms are able to find the best possible solution, that is, to explore the entire search space, a global optimum is obtained. However, in most cases this is not possible, and instead, good solutions are found by exploring only a part of the search space, and a local optimum is obtained, which in some cases may be good enough to solve the problem [25].

In more detail, a combinatorial optimization problem is a problem where the search space, that is, the set of all feasible solutions to the problem, is discrete in every case. In more formal terms, a combinatorial optimization problem can be defined by a tuple (I, h, m, g), where:

- I denotes the set of all possible instances of the problem, usually defined by the features of the particular problem.
- h denotes a map $h : I \longrightarrow \mathcal{P}(\mathcal{C})$ from the set of all possible instances to the power set (that is the set of all subsets) of the set of feasible solutions $\mathcal{C}$. In other words, given $x \in I$, $h(x)$ denotes the set of all possible solutions to the instance x of the problem, meaning all solutions that verify the restrictions imposed by the instance x and the general definition of the problem.
- m on the other hand, denotes a map $m : I \times \mathcal{C} \longrightarrow \mathbb{R}^+$. The value $m(x, y)$, where $x \in I$ and $y \in h(x)$, denotes the quality of the feasible solution y for the instance x of the problem.
- Finally, g denotes an objective, that is $g \in \{\min, \max\}$

With the previous definition, an instance x of the combinatorial optimization problem (I, h, m, g) where $x \in I$ is resolved if we found an optimal solution, that is a feasible solution $y \in h(x)$ that verifies:

$$m(x, y) = g(\{m(x, y')|y' \in h(x)\}).$$

This formal, general definition, hides several complications that deeply affect the design of methods and algorithms for the resolution of combinatorial optimization problems. We can for instance, consider that for some problems, the feasibility function h is not explicitly defined, but rather implicitly defined by a set of restrictions that can be difficult to asses on their own. Moreover, the measure of the quality of a solution m, usually called objective function, may be computationally expensive to calculate for a feasible solution, which again limits the ability to explore the search space. This is the case for our problem, where in order to evaluate m for a proposed solution, a CNN has to be trained over dozens of thousands of examples, and its value is the result of another optimization problem whose instance comes from the feasible solution of the original problem.

The aforementioned challenges have promoted the use and design of heuristics to solve these problems. Heuristics are procedures that show empirically good results on some families of problems or in instances of them. In order to be successful, a heuristic has to balance two different strategies: exploration, meaning the ability to evaluate feasible solutions on diverse parts of the search space, and exploitation, that is to find better solutions in the neighborhood of an already known solution. These strategies are hard to balance in general, and in some simple heuristics, a balance can be found suitable for an instance of the problem while for another instance the approach may prove to be widely imbalanced. These imbalances will either restrict the heuristic to a small region of the search space, forcing it to converge to poor local optima, or in the other cases, force the heuristic to randomly search large sections of the search space without significantly improving the results.

Metaheuristics are high-level procedures used to determine or design an heuristic able to achieve a sufficiently good solution to an optimization problem. Metaheuristics are efficient in complex problems that require large amounts of data and large amounts of processing. Metaheuristics achieve this because they do not guarantee a global optimum, but are able to find a good enough solution to solve the problem while consuming less time and using fewer processing resources.

Metaheuristics are particularly useful when there is no exact method of resolution or too many resources are required for an exact method, and also when the optimal solution is not needed and a good one is sufficient. In particular, bio-inspired metaheuristics draw their strategies from naturally occurring phenomena, such as genetic algorithms, ant colony optimization, algorithms based on artificial bee colonies, fireflies, particles swarm optimization, among others [11].

In particular, genetic algorithms (GA) are adaptive methods based on the evolution of biological species [15]. Genetic algorithms begin with a set of random solutions, called individuals. An encoding representation is defined, which will code each individual into a "chromosome". The representation is important, as it establishes how a chromosome represents a solution to the problem, and will modify the behaviour of the algorithm, which deals with the chromosomes rather than the actual solutions. Once the set of initial solutions is complete, an objective function (evaluation function) associated to the problem is applied to each individual. In this way, the best solutions can be obtained and the individuals of the population can then be reproduced. In this process, different operators are applied for crossover and mutation. The crossover operator implements mechanisms to mix the parent's chromosomes to obtain children chromosomes. Mutation mechanisms are also implemented, which introduce random changes in the chromosomes of the children. Finally, the best solutions according to the objective function are obtained.

V. RELATED WORK

A few recent works have explored the use of metaheuristics to learn the CNN structures for different image recognition problems with good results. In [39] the authors use genetic algorithms to optimize the depth, sequence and the number of kernels of each layer of the networks for the CIFAR10 and

CIFAR100 datasets. They use a rather recent development: the use of skip connections and residual blocks [17], which we did not include on our design. This allows them to achieve very high accuracy with the optimized architectures, by taking advantage of the deeper architectures allowed by the use of residual blocks. Moreover, they use a fixed kernel size, a hyperparameter that could be a factor to optimize. The authors achieve impressive results in both datasets, surpassing 95% test accuracy over the CIFAR10. In another direction, both [34] for CIFAR10 and [3] for the MNIST datasets, use genetic algorithms to optimize almost all possible hyperparameters of a CNN. Not only they optimize the structure of the network, but they also optimize simultaneously the training parameters of the networks. This includes the optimizer algorithm, the learning rate and even the total epochs of training (the number of iterations of the optimizer algorithm), among others. This could entail the comparison of individuals on the basis of their training rather than on the quality of their architecture. In contrast, we opted to only optimize hyperparameters associated to the CNN structure and train all our networks equally. We use the same algorithm and a convergence criteria to stop training, rather than a fixed number of epochs.

In [29], the authors work with the more demanding Caltech-256 dataset. For that task, they optimize the kernel size and kernel number maintaining a fixed depth reporting positive results. We allowed the exploration of depth, so that the method finds the best value for a given problem. The authors used the usual binary crossover operator of the original genetic algorithm. This implies that the values of the hyperparameters survive to the next generation, but not necessarily successful sequences of consecutive values. This is also the case for [3], where the authors apply the same binary crossover method. In this paper we propose novel type of crossover, combining some binary and some sequential crossover, acknowledging that the success or failure of the different architectures considered could be related to the sequence of consecutive hyperparameters rather than only to their values.

The success of recent research in this problem shows that using metaheuristics is an interesting approach to optimize the structures of CNN for a given problem. In the next section we describe our particular approach to this problem and the reasoning behind our choices.

VI. PROPOSED APPROACH

In this section, we present the genetic algorithm we propose to optimize the structure of a CNN. In the next subsections, we describe how we construct the individuals, including their fixed and optimized parts. We later discuss our encoding for chromosomes and their relevant values, and introduce the operators of our genetic algorithm. We also present our fitness functions and selection methods, and finally give an overview of the genetic algorithm hyperparameters and the CNN training process. An overview of the proposed algorithm can be found in the Algorithm 1, where max_gen correspond to

Algorithm 1 Proposed Genetic Algorithm

Input: (max_gen, cross_prob, mutation_prob, max_pop, conv_parameters)
Output: elite

```
generation ← 0
population ← Generate initial population from Input parameters.
while generation < max_gen do
    for individual in population do
        individual.accuracy ← obtain validation accuracy value from training network defined by individual's cromosome and conv_parameters
    end for
    Calculate each individual fitness from population
    elite ← Get individual with best fitness from population
    children_list ← Empty list
    next_pop ← Empty list
    for individuals in population do
        if Random([0,1]) ≤cross_prob then
            Choose parent1 and parent2 from population using proportional roulette wheel selection.
            if generation/max_gen < Random([0,1]) then
                child1, child2 ← Cross parent1 and parent2 sequentially
            else
                child1, child2 ← Cross parent1 and parent2 with binary list
            end if
            Add child1 and child2 to children_list
        end if
        if generation < max_gen/2 then
            next_pop ← apply mutation to children_list with probability mutation_prob
        else
            survivors ← Select a fraction of survivors from population, according to fitness
            Add survivors, children_list and elite to next_pop
            next_pop ← apply mutation to next_pop with probability mutation_prob
        end if
        Fill next_pop with random individuals or delete random individuals until max_pop total individuals
        if elite not in next_pop then
            Replace random individual from next_pop with elite
        end if
        population ← next_population
        generation ← generation +1
    end for
end while
Train population and return the elite
```

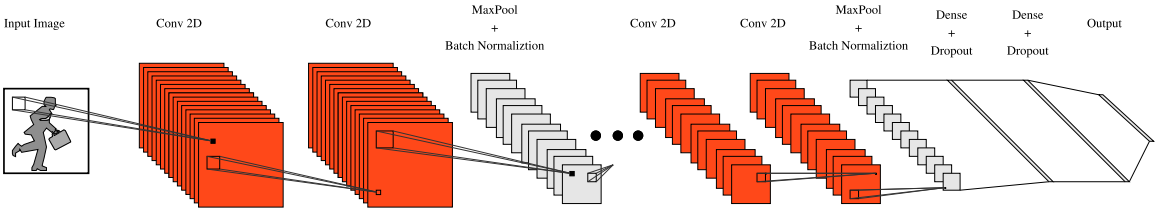

FIGURE 2. Example of a CNN structure. In our case, the red layers hyperparameters are optimized while gray layers have fixed hyperparameters.

the maximum number of generations; cross_prob and mutation_prob corresponds to the base probabilities for crossover and mutation respectively. Finally conv_parameters encompasses the definition of the CNN template and training procedure.

A. NETWORK TEMPLATE

Now we describe the network template for building the individuals. Each network has a convolutional section, composed of a sequence of convolutional, max pooling and batch normalization [20] layers. The convolutional layers are arranged in pairs, each pair followed by max pooling, dropout [38] (of only 10%) and a batch normalization layer. After the convolutional section, a fully connected section, composed of two dense layers followed each by a dropout layer (in this case of 50%), complete the network, as shown in Fig. 2. As the authors show in [38], adding dropout in both the convolutional and fully connected layers of a network can improve its performance by controlling the overfitting phenomenon. We choose the mentioned 10% and 50% after a small tuning process with a generic CNN that yielded good results for these values.

The convolutional layers allow the network to learn patterns from data, while the batch normalization layers helps the training process as shown in literature. The max pooling layers allow the network to reduce the dimensionality of the representation. After the convolutional section, the fully connected layers are able to learn the relationship between the extracted features and the category of each image. Dropout layers help to prevent overfitting during the training process.

It is worth mentioning that we use padding for the convolutional layers. That means we add zeros to the input to preserve the dimensions of the input and output of each layer. We choose this method to help to define easily the minimum and maximum number of layers (avoiding dimensional problems), as well as to allow deeper constructions in low resolution datasets (without padding images loose resolution after the convolution, as in Fig. 1). Therefore, in the convolutional section, the dimensions of the images can only be modified by the max-pooling layers. In our case, if the input image of the max pool layer has $d \times d$ pixels, the output image will be of resolution $d' \times d'$ where $d' = \lfloor \frac{d}{2} \rfloor$.

We optimize the hyperparameters of the convolutional section, in particular the number convolutional layers and its hyperparameters. The fully connected section and the batch normalization and maxpooling layers are fixed (in the sense we do not optimize the hyperparameters of these layers), as shown in Fig. 2. All the fixed hyperparamters and network template information are defined in the conv_parameters input to Algorithm 1.

B. CHROMOSOME REPRESENTATION

We now present the encoding used to represent the individuals, the relevant intervals of definition and their importance to the machine learning problem.

For the chromosome encoding, we must include the information of depth, the number of filters of each convolutional layer and the kernel size used in that layer. For simplicity, to represent an individual with n convolutional layers, we use a list of length $2n + 1$, with the first value being n. The next $2n$ values are pairs of number of filters in a layer and the kernel size used in the same layer. Then, if we denote by L_i the number of filters of the i-th convolutional layer and k_i its kernel size, for $i \in 1 \ldots n$, the corresponding chromosome is the following:

$$[n, L_1, k_1, L_2, k_2, \ldots, L_n, k_n].$$

To define the possible values each allele can take, we must first define the possible depths. This value depends strongly on the dataset. For datasets with lower resolutions (such as CIFAR10 with 32×32 pixels each image), we considered depths from 2 to 7 for most of our tests. It was not possible to consider deeper architectures with this configuration because of the dimensionality constraints. For datasets with larger resolutions (such as Caltech256 which we cropped to 128×128 pixels), it was possible to consider deeper architectures, up to 12 convolutional layers, with a minimum of 6. We had to impose a higher minimum for bigger images, to allow a sufficient dimensionality reduction before the dense section of the network. For the number of filters in each layer, we used 4 as a minimum with the maximum at 128, whereas for kernel sizes we allowed values between 2 and 9. These values were chosen considering the usual choices made in state of the art architectures for similar problems.

It is important to emphasize the influence of these hyperparameters over the hypothesis space. While the choice of these values doesn't directly affect the optimization process of the learning algorithm, it does delineates the hypothesis space over which the learner (in this case the chosen optimizer) can search for a solution to the classification problem. Different

values of depth, number of convolutional units or kernel sizes change the dimensions of the hypothesis space and therefore the complexity of the resulting hypothesis.

C. GENETIC ALGORITHM OPERATORS

We now introduce the main components of our genetic algorithm. During the design and testing of our algorithm, we implemented a few strategies that aimed at encouraging exploration during the first generations and exploitation towards the end, to help the algorithm converge.

Our mutation operator receives a list of individuals and gives each one a flat chance to be mutated, corresponding to the mutation rate. An individual chosen for mutation can then be mutated in two different ways, as shown in Figure 3, with a 50% chance each. The first option is modifying one of the alleles of the chromosome, other than the depth. If this is the case, one allele other than the first one is randomly chosen. Its value is then replaced by a random value inside the set of possible values for that allele in particular (it can represent the number of filters or the kernel size). The second option is a change in the number n of convolutional layers of the individual, in which case a new n is chosen randomly from the set of possible values. If this new value happens to be smaller than the older, the chromosome is simply trimmed endwise to fit the new depth. On the other hand, if the value is greater than the previous one, the chromosome is extended by random values in the relevant intervals.

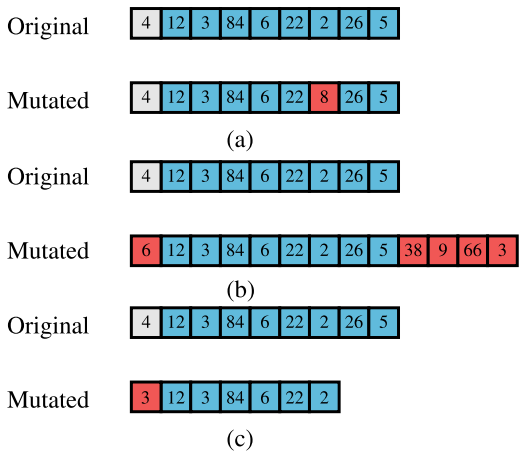

FIGURE 3. Examples of the three kinds of mutation: (a) random point, (b) layer convolutional layer number, and (c) smaller convolutional layer number. Red color respresents a new random value, blue the original value and gray squares correspond to the original number of convolutional layers (depth).

The mutation operator described above is applied to different populations depending on the current generation: during the first half of the evolution process, as seen in line 22 of Algorithm 1, the operator is applied only over the children list, to then complete the desired population with random individuals allowing for greater diversity in the early stages. During the second half of the evolution process, this operator is applied over the children, the elite and the individuals who are permitted to survive to the next population. This is shown in line 26 the Algorithm 1, and aims to preserve more information from the previous generation overall, in order to achieve an intensification of the search. The mutation rate corresponds to mutation_prob, an input of the Algorithm 1.

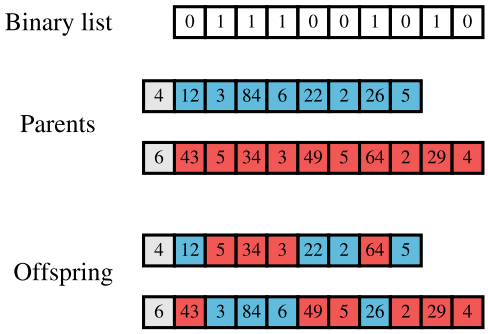

FIGURE 4. Examples of binary crossover method. In white the binary list, red and blue represent the genes of parents. The depth allele is represented in gray.

For the crossover operator, we also defined two different approaches, one based on the original binary crossover method and a sequential method. While we find the binary one is unsuited for the sequential nature of the structure to optimize, we use it as a way to augment diversity in the early generations. During evolution, each method has a probability to be used, that depends on the progress of the algorithm, fading out the binary approach in favor of the sequential one to intensify the search towards the final generations. The exact probability of using sequential crossover in generation g from a total of G generations will be $\frac{g}{G}$, and the probability of using the binary crossover $1 - \frac{g}{G}$. Our overarching strategy involves iterating over every individual, giving it a chance to mate proportional to their fitness value. If an individual is chosen for crossover, a partner is chosen randomly using the same criteria. Once both parents are determined, the crossover method is chosen. We now describe both methods in detail.

The binary crossover method first generates a random binary list with a length of twice the largest of the parents depths. Offspring's chromosomes are initialized with the depth value corresponding to one of the parents (each parent is represented in one offspring). Then, the rest of each chromosome is filled following the binary list, i.e. choosing values from one or the other parent, as shown in Fig. 4. If the length of a child exceeds that of one parent, the rest of the chromosome is filled with the alleles of the other parent. This operator, while maintaining values similar to the original chromosomes, gives radically new structures, combining these values in new random sequences. This encourages diversity and exploration in the early generations. The binary crossover operator is used in line 17 of the Algorithm 1.

In the later generations, however, we aim to intensify the search, by allowing sequential crossover to take precedence. This crossover method allows the generation of children containing sequences of successful structures, acknowledging

the sequential nature of the optimized structure. Let C_a and C_b be the parents chromosomes as follows:

$$C_a : [n_a, L_1^a, k_1^a, L_2^a, k_2^a, \ldots, L_{n_a}^a, k_{n_a}^a]$$
$$C_b : [n_b, L_1^b, k_1^b, L_2^b, k_2^b, \ldots, L_{n_b}^b, k_{n_b}^b]$$

The method first chooses randomly a crossover point, a value p between 1 and the smaller of the parent's chromosomes, that is $p \in \{1, 2, \ldots, min(n_a, n_b)\}$. Then, the resulting children's chromosomes S_1 and S_2 yielded by the method are:

$$S_1 : [n_a, L_1^b, k_1^b, \ldots, L_p^b, k_p^b, L_{p+1}^a, k_{p+1}^a, \ldots, L_{n_a}^a, k_{n_a}^a]$$
$$S_2 : [n_b, L_1^a, k_1^a, \ldots, L_p^a, k_p^a, L_{p+1}^b, k_{p+1}^b, \ldots, L_{n_b}^b, k_{n_b}^b]$$

Notice this procedure preserves the diversity of the population in term of depths without loosing successful structures, rather recombining sections of these structures. This method is also visually depicted in Fig. 5, and corresponds to the method used in the Algorithm 1, line 15.

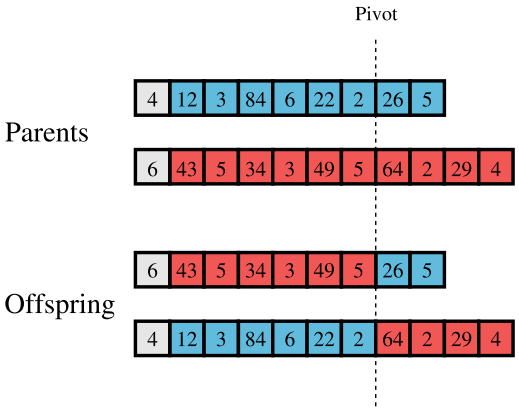

FIGURE 5. Examples of sequential crossover method. The "Pivot" is the randomly chosen value p.

D. FITNESS EVALUATION AND SELECTION

To calculate the fitness of an individual we must train it completely to measure its validation accuracy, as shown in line 7 of the Algorithm 1. We describe our training process in the last paragraph of this section. During the first half of the evolution process, we calculate the fitness of the individual proportionally to its validation accuracy. If we denote by A_i the validation accuracy of an individual i of the population P, its fitness f_i can be computed by:

$$f_i = \frac{A_i}{\sum_{j \in P} A_j}.$$

As we can see, the fitness of an individual represents the fraction it contributes to the total accuracies accumulated in the whole population. While this approach easily differentiates individuals with poor performance from good models in early stages, after a few generations the differences between the best and the worst individual become smaller. Eventually, all individuals have a similar fitness value and the algorithms can no longer differentiate their performances. This is why after reaching the midpoint of evolution, we change the strategy to a rank-based one. To compute this fitness value, we must first rank all individuals according to their accuracies: the best individual will have a rank of 1, the second-best a rank of 2 and so on. Now, if we denote by r_i the rank of the individual i, we can compute its fitness by:

$$f_i = \frac{n + 1 - r_i}{\sum_{j \in P} (n + 1 - r_j)}$$

where n is the number of individuals. This is equivalent to the previous fitness if we replace the validation accuracy by $n + 1 - r_i$, a value that represents how good is the network compared to the rest of the population.

During the evolution process, we use an elitist approach, as shown in line 30 of the Algorithm 1. Also, to maintain a constant number of individuals through the evolution process, we implement two different approaches. During the first half of evolution, the next generation is initialized with the best individual, the mutated individuals, and the children. Then, the desired population size is obtained by generating new random individuals. This means that the only individual that survives unaltered does so through elitism. This is done so to allow for a greater diversity during the first generations, introducing random individuals to each new generation.

During the latter half of evolution, the elite and modified individuals (mutated and children) are also part of the next generation. However, to achieve the desired population size, individuals of the previous generation have a chance to survive to the next one, proportional to their fitness value. This allows some successful individuals from a generation to survive to the next one unchanged, intensifying the search process.

E. GENETIC ALGORITHM HYPEPARAMETERS AND NETWORK TRAINING

For assessing our proposal, we conduct experiments over a constant population of 12 individuals through 12 generations. We used a mutation rate of 0.3, which means that the expected number of mutated individuals corresponds to 30% of the population, which is achieved by giving each individual a 0.3 flat chance of mutation (regardless of their fitness). We used 0.25 as the base probability of an individual having children so that the expected number of children corresponds to 0.5 times the population (since each individual that mates spawn 2 children). These values correspond to the following inputs of the Algorithm 1: max_gen for the total number of generation, cross_prob for the base probability of having children, mutation_prob for the mutation probability and max_pop to the number of individuals.

To train a new individual we use the Adam optimizer [21] over the train set with its default parameters. We train all networks monitoring their validation accuracy, and stop training after 3 consecutive iterations where the validation accuracy does not improve. We store the best validation

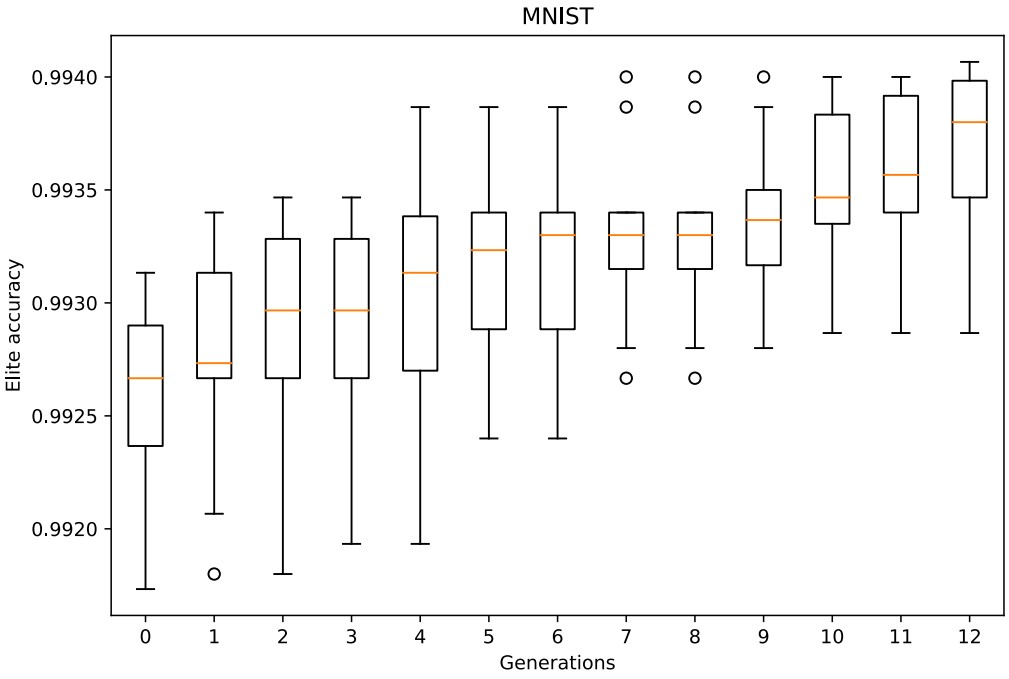

FIGURE 6. Evolution of the validation accuracy of the elites over the generations for the MNIST dataset. The results corresponds to 10 different seeds.

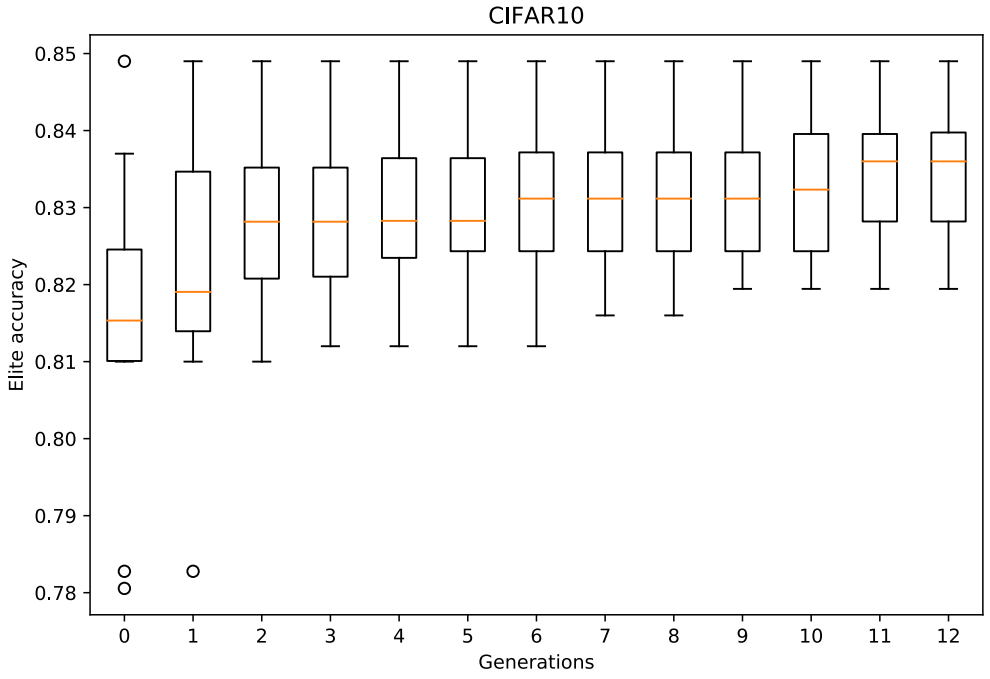

FIGURE 7. Evolution of the validation accuracy of the elites over the generations for the CIFAR10 dataset. The results corresponds to 10 different seeds.

accuracy and we use it to compute the fitness of individual solutions. This training parameters are considered on the input conv_parameters of the Algorithm 1. Also, when a trained individual survives to the next generation, we do not train it again, since we already have a measure of its quality through the original validation accuracy. It is worth noting that the splitting of the dataset into train, validation and test sets is done before the evolution process. All training and validation processes are done with the same sets of examples.

In the next section we describe the different test realized with this approach, describing the particular considerations for each dataset.

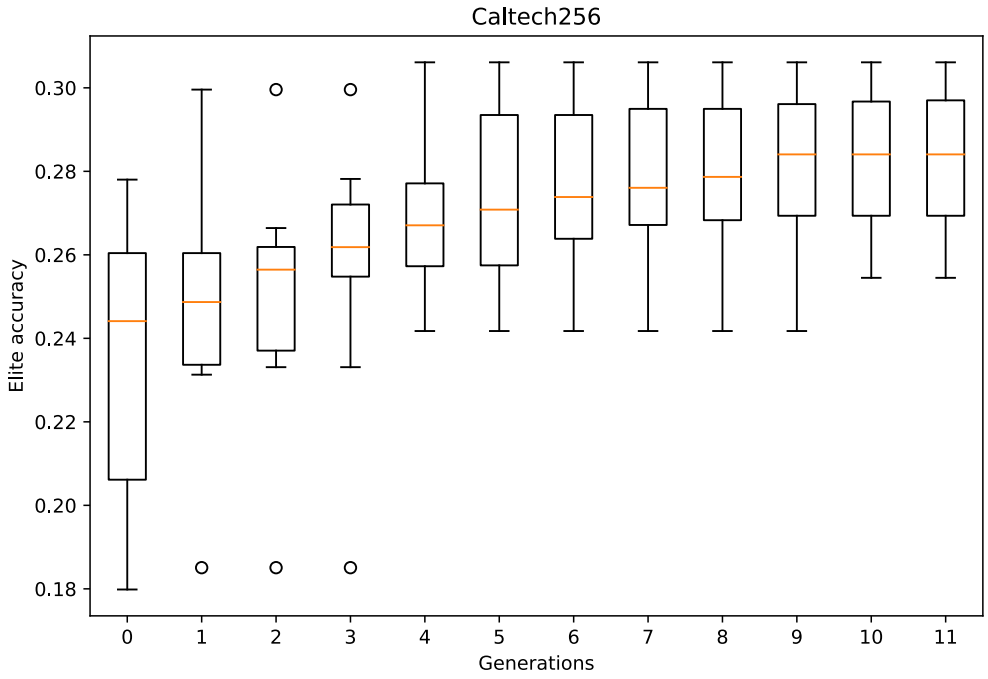

FIGURE 8. Evolution of the validation accuracy of the elites over the generations for the Caltech256 dataset. The results corresponds to 10 different seeds.

VII. RESULTS

We performed all our tests using GPU acceleration for training (Nvidia GTX 2080). We used our algorithm to find optimal structures for the datasets MNIST, CIFAR10, and CALTECH256, running each test 10 times with different seeds for the genetic algorithm. Besides our proposal, we also performed tests using the algorithms proposed by Loussaief and Abdelkrim [29], Sun *et al.* [39], and Bhandare and Kaur [3] for each dataset. For Loussaief *et al.* we were able to execute 10 different runs, while for the rest only one run was performed because of time constraints. The results in terms of performance are summarized in Table 1, while Table 2 presents the running time of each experiment. In the following paragraphs each dataset is described along with the relevant adjustment to our algorithm. A brief discussion on the performance of the implemented algorithms is presented for each dataset.

The MNIST dataset is composed of 70,000 28 × 28 pixels black and white images of handwritten digits (0-9), of which 45,000 are used for training, 15,000 for validation and 10,000 for testing set. This dataset has the smallest resolution of all three datasets. For our algorithm, we explored depths between 2 and 6, filters for each convolutional layer between 4 and 64 and kernel sizes between 2 and 9. Considering the complexity of the dataset, we used 64 neurons for the dense layers. From Table 1, we see our proposal achieves a final accuracy over the test set of 99.56% ± 0.027%, averaging 5.35 ± 0.211 hours for each experiment, as presented in Table 2. As a comparison, Loussaief *et al.*'s method achieves 99.16±0.142% with an average experiment duration of 1.59±0.126. Sun *et al.*'s algorithms yields 99.3% accuracy after 135.42 total hours, while Bhandare's algorithm achieves 99.02% accuracy after 6.4 hours. We can observe the evolution of the validation accuracies of the 10 elites of our proposal through the generations in Fig 6.

TABLE 1. Test accuracies in percentages of different algorithms. The values presented for Sun and Bhandare correspond to a single run, while for our proposal and Loussaief's 10 runs were performed.

	MNIST	CIFAR10	Caltech256
Loussaief [29]	99.16 ±0.142	77.55 ±0.615	22.15 ±2.519
Bhandare [3]	99.02	74.43	17.64
Sun [39]	99.3	82.28	*
Our Proposal	**99.56** ± 0.027	**84.85** ± 0.366	**33.3** ± 1.364

The second dataset we worked with is the CIFAR10 dataset. It is composed of 60,000 32 × 32 RGB images labeled in 10 categories, such as dog, truck, or ship. We used 45,000 images for training, leaving 9,000 for validation and 6,000 for testing. For this dataset, we allowed our algorithm to explore depths between 2 and 7, the number of filters between 4 and 128 and kernel sizes ranging from 2 to 9, with the dense layers of the networks composed of 128 neurons. From Table 1, we can see our algorithm found networks that achieved 84.85% ± 0.336% accuracy over the test set, while Table 2 shows the average experiment lasted 12.39 ± 1.981 hours. In the same dataset, Loussaief *et al.* achieves 77.55%± 0.615% accuracy in an average of 1.29 ± 0.072 hours, while Sun *et al.* reaches 82.28% accuracy in 257.4 hours and Bhandare *et al.*'s proposed algorithm reaches 74.43% accuracy in 6.81 hours. The validation accuracies of the elites trough evolution process of the 10 runs of our algorithm can be observed in Fig. 7.

Finally, the more complex dataset is the CALTECH256. Composed of 30,607 RGB images of different resolutions and aspect ratio, the CALTECH256 has 256 distinct categories

TABLE 2. Total time spent in each experiment. The average over 10 runs is reported for our proposal and Loussaief *et al.*. For the rest of the experiments only one run was performed. It must be pointed out that Sun *et al.*'s proposal proved incompatible with the Caltech256 dataset. All values are in hours.

	MNIST	CIFAR10	Caltech256
Loussaief [29]	$\mathbf{1.59} \pm 0.126$	$\mathbf{1.29} \pm 0.072$	$\mathbf{7.96} \pm 0.595$
Bhandare [3]	6.4	6.81	50.03
Sun [39]	135.42	257.4	$*$
Our Proposal	5.35 ± 0.211	12.39 ± 1.981	41.98 ± 4.94

plus a "clutter" category for images with no identifiable object. We prepared the images by first filling the smaller dimension with black pixels in order to get a square aspect ratio. Then, all images were scaled to 128×128. We also used 75%, 15% and 10% of the images for the training, validation and test set respectively, preserving the proportion of examples in each category. For this task, our algorithm explores depths between 6 and 10, with 4 to 128 filters in each convolutional layer and kernels of sizes 2 to 9. The final dense layers have 128 units. As shown in Table 1, an accuracy of $33.3 \pm 1.364\%$ was achieved by our algorithm in an average of 41.98 ± 4.94 hours. Loussaief *et al.* achieved $22.15 \pm 2.519\%$ in the same task, with an average experiment of 7.96 ± 0.595 hours. Bhandare *et al.* achieves 17.64% in 50.03 hours, while the dimensionality of the problem caused issues for Sun *et al.*'s proposal and we were unable to run a successful test because of memory problems.

From the presented comparisons, we can see that in most cases our algorithm outperforms state of the art proposals without incurring into a significant increase in the computation times required to optimize the relevant parameters. In fact the only proposal consistently faster than ours is Loussaief's but still the total computation times are in the same orders of magnitude.

VIII. CONCLUSION

This paper aimed to present and validate a novel genetic algorithm for the automatic optimization and design of CNN architectures for image classification problems. To this end, we first circumscribed the list of optimizable hyperparameters, ensuring the individuals are compared based on the performance of their underling architectures and not their training hyperparameters. A fundamental piece of our method is a novel crossover operator that considers the nature of the underlying structures, as well as a way to explore the different possible depths in an automated manner. We implemented different approaches for the crossover operation and the environmental selection over the generations as well, to allow for greater diversity in the earlier generations and intensify the search towards the latter stages.

We were able to demonstrate the quality of our approach by performing several experiments with consistently good results, outperforming state of the art algorithms over different datasets in terms of the accuracy of the final network's architectures. We conclude that the proposed algorithm is a consistently successful tool for the optimization of hyperparameters of CNN for different image classification tasks.

We consider this subject still has many interesting lines of future investigation. For instance the exploration of different of network architectures, or the optimization of the convolutional construction blocks of the networks in an automated manner instead of using fixed blocks for all networks.

REFERENCES

[1] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, Nov. 2018, Art. no. e00938.

[2] Z. Beheshti and S. M. H. Shamsuddin, "A review of population-based meta-heuristic algorithms," *Int. J. Adv. Soft Comput. Appl*, vol. 5, no. 1, pp. 1–35, 2013.

[3] A. Bhandare and D. Kaur, "Designing convolutional neural network architecture using genetic algorithms," in *Proc. Int. Conf. Artif. Intell. (ICAI)*, 2001, pp. 150–156.

[4] L. Bianchi, M. Dorigo, L. M. Gambardella, and W. J. Gutjahr, "A survey on metaheuristics for stochastic combinatorial optimization," *Natural Comput.*, vol. 8, no. 2, pp. 239–287, Jun. 2009.

[5] L. Bottou, "Stochastic gradient learning in neural networks," *Proc. Neuro-Nımes*, vol. 91, no. 8, p. 12, 1991.

[6] Z. Chen, A. Huang, and X. Qiang, "Improved neural networks based on genetic algorithm for pulse recognition," *Comput. Biol. Chem.*, vol. 88, Oct. 2020, Art. no. 107315.

[7] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "A downsampled variant of ImageNet as an alternative to the CIFAR datasets," 2017, *arXiv:1707.08819*. [Online]. Available: http://arxiv.org/abs/1707.08819

[8] B. Crawford, R. Soto, G. Astorga, J. García, C. Castro, and F. Paredes, "Putting continuous metaheuristics to work in binary search spaces," *Complexity*, vol. 2017, pp. 1–19, Dec. 2017.

[9] J. A. Cruz and D. S. Wishart, "Applications of machine learning in cancer prediction and prognosis," *Cancer Informat.*, vol. 2, Jan. 2006, Art. no. 117693510600200.

[10] A. Dey, "Machine learning algorithms: A review," *Int. J. Comput. Sci. Inf. Technol.*, vol. 7, no. 3, pp. 1174–1179, 2016.

[11] T. Dokeroglu, E. Sevinc, T. Kucukyilmaz, and A. Cosar, "A survey on new generation Metaheuristic algorithms," *Comput. Ind. Eng.*, vol. 137, Nov. 2019, Art. no. 106040.

[12] K. Fukushima and S. Miyake, "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition," in *Competition Cooperation Neural Nets*. Berlin, Germany: Springer, pp. 267–285, 1982.

[13] M. Gendreau and J.-Y. Potvin, "Metaheuristics in combinatorial optimization," *Ann. Oper. Res.*, vol. 140, no. 1, pp. 189–213, 2005.

[14] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. 13th Int. Conf. Artif. Intell. Statist.*, 2010, pp. 249–256.

[15] D. E. Goldberg, *Genetic Algorithms*. London, U.K.: Pearson, 2006.

[16] D. M. Hawkins, "The problem of overfitting," *J. Chem. Inf. Comput. Sci.*, vol. 44, no. 1, pp. 1–12, Jan. 2004.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, abs/1512.03385, pp. 1–5, Oct. 2015.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1026–1034.

[19] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *J. Physiol.*, vol. 195, no. 1, pp. 215–243, Mar. 1968.

[20] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015, *arXiv:1502.03167*. [Online]. Available: http://arxiv.org/abs/1502.03167

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: http://arxiv.org/abs/1412.6980

[22] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," *Emerg. Artif. Intell. Appl. Comput. Eng.*, vol. 160, pp. 3–24, Dec. 2007.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[24] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, "Object recognition with gradient-based learning," in *Shape, Contour Grouping Computer Vision*. Berlin, Germany: Springer, 1999, pp. 319–345.
[25] S. Lendl and A. P. Punnen, "Combinatorial optimization with interaction costs: Complexity and solvable cases," *Discrete Optim.*, vol. 33, pp. 101–117, Aug. 2019.
[26] Y. D. Li, Z. B. Hao, and H. Lei, "Survey of convolutional neural network," *J. Comput. Appl.*, vol. 36, no. 9, pp. 2508–2515, 2565, Sep. 2016.
[27] W.-Y. Lin, Y.-H. Hu, and C.-F. Tsai, "Machine learning in financial crisis prediction: A survey," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 42, no. 4, pp. 421–436, Jul. 2012.
[28] I. Livieris and P. Pintelas, "A survey on algorithms for training artificial neural networks," Dept. Math., Univ. Patras., Patras, Greece, Tech. Rep. TR08-01, 2008.
[29] S. Loussaief and A. Abdelkrim, "Convolutional neural network hyperparameters optimization based on genetic algorithms," *Int. J. Adv. Comput. Sci. Appl.*, vol. 9, no. 10, pp. 252–266, 2018.
[30] N. M. Aszemi and P. D. D. Dominic, "Hyperparameter optimization in convolutional neural network using genetic algorithms," *Int. J. Adv. Comput. Sci. Appl.*, vol. 10, no. 6, pp. 269–278, 2019.
[31] S. Nesmachnow, "An overview of metaheuristics: Accurate and efficient methods for optimisation," *Int. J. Metaheuristics*, vol. 3, no. 4, pp. 320–347, 2014.
[32] V. K. Ojha, A. Abraham, and V. Snáel, "Metaheuristic design of feedforward neural networks: A review of two decades of research," *Eng. Appl. Artif. Intell.*, vol. 60, pp. 97–116, Apr. 2017.
[33] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural Comput.*, vol. 29, no. 9, pp. 2352–2449, Sep. 2017.
[34] A. J. Reiling, "Convolutional neural network optimization using genetic algorithms," Ph.D. dissertation, School Eng., Univ. Dayton, Dayton, OH, USA, 2017.
[35] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: http://arxiv.org/abs/1409.1556
[36] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: http://arxiv.org/abs/1409.1556
[37] L. N. Smith and N. Topin, "Deep convolutional neural network design patterns," 2016, *arXiv:1611.00847*. [Online]. Available: http://arxiv.org/abs/1611.00847
[38] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.
[39] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Automatically designing CNN architectures using genetic algorithm for image classification," 2018, *arXiv:1808.03818*. [Online]. Available: http://arxiv.org/abs/1808.03818
[40] R. S. Sutton, "Introduction: The challenge of reinforcement learning," in *Reinforcement Learning*. Boston, MA, USA: Springer, 1992, pp. 1–3.
[41] Z. Wang, F. Li, G. Shi, X. Xie, and F. Wang, "Network pruning using sparse learning and genetic algorithm," *Neurocomputing*, vol. 404, pp. 247–256, Sep. 2020.
[42] M. D. Zeiler, "ADADELTA: An adaptive learning rate method," 2012, *arXiv:1212.5701*. [Online]. Available: http://arxiv.org/abs/1212.5701

FRANKLIN JOHNSON received the B.E. degree, the master's degree in informatics, and the Ph.D. degree from the Pontificia Universidad Católica de Valparaíso, in 2006 and 2017, respectively.

He is currently the Head of the Department of Computer Sciences, Universidad de Playa Ancha. His research interests include metaheuristics, autonomous search, and combinatorial optimization.

ALVARO VALDERRAMA received the bachelor's degree in mathematical engineering from the Universidad Técnica Federico Santa María, Chile, in 2019, where he is currently pursuing the master's degree in computer science.

Since 2018, he has been an Independent Consultant for renewable energy related projects sponsored by the Ministry of Energy, Santiago, Chile. His research interests include machine learning, deep neural networks, and image classification.

CARLOS VALLE received the Ph.D. degree in computer science from the Universidad Técnica Federico Santa María, Chile, in 2014.

From 2014 to 2018, he was a Researcher with the Computer Science Department, Universidad Técnica Federico Santa María. Since 2018, he has been an Associate Professor with the Department of Computer Science and Informatics, Universidad de Playa Ancha, Chile. His research interests include machine learning, ensemble learning, deep neural networks, and pattern recognition with applications to time series.

BRODERICK CRAWFORD received the M.Sc. degree from the Universidad de Chile, in 2001, and the Ph.D. degree in informatics engineering from the Universidad Técnica Federico Santa María, Valparaíso, in 2011.

He has more than 20 years of teaching experience with universities. He has about 15 years of experience working in research teams. He is currently a Professor of computer science with the Pontificia Universidad Catolica de Valparaíso, Chile. He has published more than 300 scientific papers in different peer-reviewed international conferences and journals involving different topics, such as artificial intelligence, constraint programming, and software engineering.

RICARDO SOTO received the Ph.D. degree in computer science from the University of Nantes, France, in 2009.

He is currently a Full Professor and the Head of the Computer Science Department, Pontificia Universidad Catolica de Valparaíso, Chile. He has published more than 200 scientific papers in different international conferences and journals, some of them top ranked in computer science, operational research, and artificial intelligence. Most of these articles are based on the resolution of real-world and academic optimization problems related to industry, manufacturing, rostering, and seaports. His main research interests include metaheuristics, global optimization, and autonomous search.

RICARDO ÑANCULEF (Member, IEEE) received the Ph.D. degree in computer science from the Universidad Técnica Federico Santa María, Chile, in December 2011. His Ph.D. thesis on the use of computational geometry methods for training support vector machines.

From 2012 to 2013, he held a Postdoctoral Research position with the Intelligent Systems Laboratory, University of Bristol, U.K., working on the application of online learning methods to real time text tagging. In 2014, he was an Assistant Professor with the Universidad Técnica Federico Santa María, teaching computational statistics and machine learning. His current research interests include ensemble learning, deep learning, pattern recognition, and applications to time series and biomedical data analysis. He collaborates on these topics with research groups from the University of Bologna, Italy, and the University of Barcelona, Spain.