Mathematics in Engineering, 5(1): 1–21.
DOI: 10.3934/mine.2023012
Received: 24 March 2021
Revised: 19 October 2021
Accepted: 13 January 2022
Published: 11 February 2022

http://www.aimspress.com/journal/MinE

Research article

Neural architecture search via standard machine learning methodologies†

Giorgia Franchini[1,2,*], Valeria Ruggiero[1], Federica Porta[2] and Luca Zanni[2]

[1] Department of Mathematics and Computer Science, University of Ferrara, Via Machiavelli, 30, 44121 Ferrara (FE), Italy

[2] Department of Physics, Informatics and Mathematics, University of Modena and Reggio Emilia, Via Campi 213/B, 41121 Modena (MO), Italy

† **This contribution is part of the Special Issue:** Mathematical aspects of machine learning
Guest Editors: Ernesto De Vito; Lorenzo Rosasco; Silvia Villa
Link: www.aimspress.com/mine/article/6026/special-articles

* **Correspondence:** Email: giorgia.franchini@unimore.it.

Abstract: In the context of deep learning, the more expensive computational phase is the full training of the learning methodology. Indeed, its effectiveness depends on the choice of proper values for the so-called hyperparameters, namely the parameters that are not trained during the learning process, and such a selection typically requires an extensive numerical investigation with the execution of a significant number of experimental trials. The aim of the paper is to investigate how to choose the hyperparameters related to both the architecture of a Convolutional Neural Network (CNN), such as the number of filters and the kernel size at each convolutional layer, and the optimisation algorithm employed to train the CNN itself, such as the steplength, the mini-batch size and the potential adoption of variance reduction techniques. The main contribution of the paper consists in introducing an automatic Machine Learning technique to set these hyperparameters in such a way that a measure of the CNN performance can be optimised. In particular, given a set of values for the hyperparameters, we propose a low-cost strategy to predict the performance of the corresponding CNN, based on its behavior after only few steps of the training process. To achieve this goal, we generate a dataset whose input samples are provided by a limited number of hyperparameter configurations together with the corresponding CNN measures of performance obtained with only few steps of the CNN training process, while the label of each input sample is the performance corresponding to a complete training of the CNN. Such dataset is used as training set for a Support Vector Machines for Regression and/or Random Forest techniques to predict the performance of the considered learning methodology, given its performance at the initial iterations of its learning process. Furthermore, by a probabilistic exploration of the hyperparameter space, we are able to find, at a quite low cost, the setting of a CNN hyperparameters which provides the optimal performance. The results

of an extensive numerical experimentation, carried out on CNNs, together with the use of our performance predictor with NAS-Bench-101, highlight how the proposed methodology for the hyperparameter setting appears very promising.

Keywords: convolutional neural network; neural architecture search; automated machine learning; support vector machines for regression; ensemble methods; NAS-Bench dataset

1. Introduction

Machine Learning (ML) and Deep Neural Networks (DNN) are pervasive in the domain of surveillance, health-care, autonomous machinery, and vehicles. Examples can be found in EU-founded H2020 projects on automotive and smart-cities, such as CLASS (https://class-project.eu/partners) and PRYSTINE (https://www.ecsel.eu/projects/prystine), which are relying on ML and DNN technologies for solving several tasks, such as object-detection for obstacle avoidance and emergency-braking, vehicle-to-vehicle tracking, monitoring and data-fusion. However, the process of designing an efficient and accurate ML methodology for a specific task is power and time consuming and traditionally requires direct human intervention. In this paper we focus on the investigation of the most expensive computational phase of a ML methodology: the training phase. Since the same ML methodology may lead to substantially different performances depending on the setting of the so-called hyperparameters on which it is based, it is crucial to reduce the number of possible hyperparameter configurations to test in order to not burden the training phase. We recall that by hyperparameters of a learning methodologies we refer to those parameters that are not trained during the learning process but they are set a-priori as input data. In the literature there are different strategies to approach the problem of setting the hyperparameters. There exist static rules, i.e. rules that do not depend on the training phase, and dynamic rules, which only operate under certain conditions connected to the course of the training phase. Besides these strategies, another possible approach for the hyperparameter setting is based on Automated Machine Learning (AutoML) and Neural Architecture Search (NAS) techniques (see, for example, [12, 13, 17] and references therein). AutoML and NAS try to tune the topologies of the ML methodologies itself mainly focusing on the performance improvement. For example, in [2, 11] the final performance of partially trained DNNs can be predicted on the basis of simple regression models or probabilistic models that extrapolate the performance from the first part of a learning curve.

In this paper, we propose a low-cost AutoML technique based on a strategy able to predict the performance of a learning methodology starting on its behavior after only few steps of the training process. This prediction is obtained by training a Support Vector Machines for Regression (SVR) [9] and a Random Forest (RF) [8] learning technique on a suitable dataset whose input samples are given by a limited number of hyperparameter configurations together with the corresponding ML methodology performance obtained in only few steps of the training process, while the label of each input is provided by the performance related to the full training of the methodology itself (hereafter also called *the final performance* of the methodology). The approach we propose, in brief, aims to do meta-learning, i.e., learn by learn (see for example [16,23] and references therein). This means that we exploit one learning technique to model another learning technique. The benefit lies in the fact that the supervisor of the

methodology is a technique known in the literature and very robust, for example SVR and RF, while the supervised methodology is more unstable with respect to its hyperparameters, as for example a Deep Learning (DL) methodology. For this reason, even if we have to take care of the best setting of the SVR and RF hyperparameters, this task is enormously easier than the same one in the DL framework. Although the method we suggest is suitable for any ML methodology, from now on we will focus on Convolutional Neural Networks (CNNs) and the setting of their hyperparameters.

A further goal of this paper is to use the developed prediction method in the context of CNN Hyperparameters Optimisation (HO). The most intuitive approaches to tackle the HO problem are grid search and random search [6] techniques. Both methods require to drive many Artificial Neural Networks (ANNs) to convergence, making the methods very expensive from the computational point of view. As a consequence Sequential Model-Based Optimisation (SMBO) techniques have been proposed [5, 18, 21, 22] with the aim to minimise the number of ANNs driven to convergence by a proper identification of a sequence of hyperparameter settings which guarantees a corresponding sequence of increasing final performances. More recently, new HO methods based on Reinforcement Learning (RL) have emerged [3, 10, 25, 26]. The goal for most of them was to find the ANN architectures that are likely to yield an optimised performance. Thus, they were seeking the appropriate architectural hyperparameters, as the number of layers or the structure of each layer. In these papers many other hyperparameters, such as the learning rate of the optimisation algorithm and regularisation parameters, are manually chosen; therefore, these last hyperparameters are not the subject of the optimisation that we are describing. While SMBO and RL strategies aim to drive to numerical convergence as few ANNs as possible, they do not care to reduce the computational cost of the single ANN training.

The before mentioned prediction phase of our approach can be exploited to overcome this drawback. In particular, we suggest a HO phase which benefits of the prediction one, allowing us to bring to numerical convergence a lower number of CNNs with respect to that of SMBO and RL. The HO phase of our strategy exploits a probabilistic exploration of the hyperparameters space and can provide the optimal setting of hyperparameters themselves. Finally, for our HO phase, we can consider not only the hyperparameters connected to the CNN architecture, but also the hyperparameters related to the optimisation algorithm employed to train the CNN. To conclude, the proposed approach aims to combine these two aspects: the prediction of the final performance of a CNN at a low cost, and the automatic setting of its hyperparameters.

This paper, except for the introduction and conclusions, is essentially structured in two sections. In the first section, we describe the method we are proposing. In the second section, we analyze a specific problem and we carry out an exhaustive number of numerical experiments, aimed to evaluate the effectiveness of the developed approach. Furthermore, to provide some comparison with other NAS solutions, we report some results obtained by running the proposed approach on the Nas-Bench-101 dataset [24].

2. The proposed method

The performance of the ANNs and more generally of the machine learning methodologies have shown great potential, especially in recent years. These technologies are applied in many fields: from medicine to autonomous driving, from the study of time series in industrial processes to climate

forecasts of rare events. Despite the vast state of the art on the subject, designing a new machine learning methodology, especially in DL field, remains a very difficult task. Great part of the difficulty lies in designing the structure of the method by properly setting its hyperparameters such as: steplength, mini-batch size, type of optimiser algorithm, type of layer, number of neurons per layer, dropout rate and so on. The various combinations of these hyperparameters can make the difference between a mediocre performance and a performance comparable to that of state of the art methodologies. The search for the optimal configuration is done by optimising a suitable metric that quantifies the performance of the network. In the case of classification problems, this measure can be the accuracy of the method, defined as the percentage of well classified cases by the methodology with respect to the total cases; in the case of regression, the discrepancy between the ground truth and the predicted target could be used to evaluate the performance. We remark that in this paper we use the term *performance* to indicate both accuracy and discrepancy since the difference between them is clear from the context. This process of performance optimisation is made especially hard because it is particularly expensive from a computational point of view.

Starting from the observations just made, we have developed an off-line method to predict the performance of a new machine learning methodology. In particular, the proposed method allows to predict the behavior of a CNN corresponding to an hyperparameter configuration by avoiding excessive computational costs. Indeed, once the CNNs corresponding to a very few number of hyperparameter settings have been driven to numerical convergence, the suggested method is able to infer the final performance of a new CNN by taking into account its performance only in the first steps of the training process. To realise this predictor we based ourselves on two methodologies at the same time: the SVR and the RF ones. These methodologies can be particularly useful to predict the performance of a CNN from the earliest epochs and to look for the optimal configuration of the hyperparameters. The algorithm we propose may be of particular interest for a fairly quality assessment of a new learning methodology.

Before describing and motivating the proposed method, we summarise the main steps of its structure. In the first phase, a convenient dataset of examples will be created starting from some specific hyperparameter settings. In the second phase this dataset will be employed to simultaneously train an SVR and a RF methodologies; the combination of their results will be able to predict the final performance of a CNN, related to a particular hyperparameter configuration, by only using the performance provided in the early epochs. In the third phase, this predictive tool is the key to optimise the hyperparameters of a network, by exploring the hyperparameters space with a suitable algorithm which dynamically updates a probability distribution of its elements.

2.1. First phase: generation of a dataset of examples

The first step of the method is to create a dataset of examples. We need a dataset where, for some fixed hyperparameter configurations, the performance provided by the related learning methodology in the initial iterations is matched with the performance obtained when the numerical convergence is reached. The dataset must be as meaningful and complete as possible with respect to the hyperparameters space we have chosen, compatibly with reasonable time and hardware capabilities. In the space of all the possible hyperparameter configurations, the percentage of samples to include in our dataset is driven by the available resources, i.e., it may be subject to time or hardware constraints. Once the samples that will form the dataset are chosen, namely once some particular hyperparameter

configurations have been fixed, the corresponding CNNs are trained until numerical convergence, by collecting all the features that identify the examples together with their final performances. In particular, given the training set, the validation set and the testing set of a particular problem on which the CNN is applied, the training phase for the CNN dataset construction is carried out on the training set connected to the problem, while the related final performance is evaluated using the testing set. Moreover, during the training phase of each CNN, we collect at each epoch also some metrics measured on the training set and the performance on the validation set.

The dataset creation phase is the most expensive phase of the suggested method. However, we observe that once the hyperparameter configurations that constitute the dataset has been fixed, the corresponding networks can be trained independently of each other. The high degree of parallelisation of this phase makes it particularly suitable for node infrastructures.

2.2. Second phase: performance predictor

As already noted, given a particular hyperparameter configuration, the main aim of the proposed approach is to predict the behavior of the corresponding CNN at convergence based only on the performance, measured on the validation set, relative to the initial training phase. To achieve this ambitious goal we have combined two methodologies well known in the literature: the SVR and the RF techniques.

2.2.1. Role of the SVR and the RF methodologies in the proposed method

The Support Vector Machines (SVM) methodology can be included in the supervised learning, where the data are labelled. In case of binary classification problems, the basic idea of the methodology is to devise a classifier, named SVM classifier, given by a separating hyperplane (or separating hypersurface) through the selection of the most significant examples of the dataset, called support vectors. The optimal separating hyperplane provided by SVM is then used to classify new examples. In its simplest formulation, SVM defines a linear classifier but it can be generalised to obtain non-linear classifiers by exploiting the so-called *kernel trick*. By mapping the examples into a suitable high dimensional feature space and by looking for the SVM linear classifier in that space, the user defines a non-linear decision function in the original input space. To work in the new feature space, only the scalar product between examples in the feature space has to be defined and this can be expressed by suitable kernel functions. Furthermore there is a version called SVR which, using similar principles, constitutes a methodology for regression.

One of the most important features of SVM, both in classification and regression, is that the solution is represented by using only the examples belonging to a subset of the original training set and this can lead to a significant cost reduction from a computational point of view.

On the other hand, RF is an ensemble learning methodology for classification, regression and other tasks that works as follows: it constructs a multitude of decision trees at the training time, then it finds the class that is the mode or mean (respectively for classification or regression) of the classes: this is the output of the methodology. The main principle behind this model is that a group of weak learners, the single decision trees, come together to form a strong learner, producing an increase of the performance of the model. The training procedure for RF uses the general technique of bootstrap aggregating, or bagging, i.e., given a training set $X = x_1, \ldots, x_n$ with responses $Y = y_1, \ldots, y_n$, a random sample of

the training set is selected (with replacement) for B times to train B decision trees, obtaining B *simple* classification trees; the *final* decision can be made by taking the mode or the average value of all the individual trees. This bootstrapping procedure leads to better model performance because the variance of the model is decreased without increasing the bias. In this way, while the predictions of a single tree are highly sensitive to noise in its training set, the average of all the trees is not, as long as they are not correlated.

2.2.2. The prediction step

After the first step aimed to generate a suitable dataset, an SVR and a RF methodologies are trained on this generated dataset of hyperparameter configuration. As input features of the SVR technique, in addition to the hyperparameters, there is also the CNN performance at the early epochs, while the final CNN performance (measured on the testing set) represents the label of the sample. In particular, the CNN performance includes 4 values for each early epoch: the loss value on the training and validation sets, the Mean Absolute Error (MAE) value on the training and validation sets. At this point, the trained SVR methodology is able to predict the final performance of a CNN related to a particular hyperparameter configuration, which is not in the training set, given its performance (on the training and validation sets) in the initial phase of the learning process. In a similar way, we can obtain by the RF methodology a prediction of the final performance of a specific CNN from its behavior at the initial phase of the learning process, without using the testing set. Therefore, having both the results of SVR and RF available, we can use such results separately or in a proper combination. As mentioned before the ensemble methods lead to better model performance due to a variance reduction. Driven by this observation, in the proposed method, the final CNN predicted performance is the arithmetic mean between the prediction provided by SVR and that provided by RF; hereafter we call this approach HYBRID. The developed method can be particularly convenient from a computational point of view. This is because at the pure cost of creating the dataset, it generates a tool capable of predicting the performance of a new methodology based only on its initial training phase. An advantage of the proposed approach is its great scalability. The dataset that must be created does not depend neither on the complexity of the problem nor on the time taken by the single training (which can often be variable).

2.3. Third phase: from predictor to hyperparameter optimisation

In order to estimate the optimal hyperparameter configuration, we use Algorithm 2.1, which is an extension of [14], to perform the hyperparameters space exploration. In the following, we detail the main steps of Algorithm 2.1.

Each investigated hyperparameter is described by the letter i associated with the vector V_i which contains all its possible configurations. Each element of the V_i vectors is associated with a probability. These probabilities are stored in the P_i vectors and their initial values are set as proportional to the final performance. More in details, we consider each V_i separately; for each $a_j^i \in V_i$ we find all the dataset elements that have feature a_j^i and for these elements we sum all the related performance. At this point for each vector V_i we have a cumulative vector, of the same size, with the performances sum. We normalized these cumulative vectors to have a probability vectors. We summarized the main steps of this computation in Algorithm 2.2. In the normalization step, it is fundamental to pay

attention to the two different cases: if we use accuracies as performance metric, we can consider a probability proportional directly to the accuracy, but if the discrepancy is measured, we have to use the difference between the maximum discrepancy and the actual one. This initialisation procedure mimics the setting of a genetic algorithm (GA) [15]. Indeed, as in GA, each individual in the population has a probability proportional to its fitness to reproduce; also in our case, each characteristic have a probability proportional to its performance of being chosen to generate a new configuration of hyperparameters.

Algorithm 2.1 The hyperparameter space exploration algorithm

1: $t = 0$
2: **while** $t \leq maxiter$ and $convergence == false$ **do**
3: Given the vectors V_i, set the associated probabilities P_i as specified in Algorithm 2.2 with the actual dataset.
4: Generate a realisation of the random state $x^{(t)}$ according with probabilities.
5: Training CNN($x^{(t)}$) for some epochs.
6: Use SVR and RF to predict final performance $r^{(t)}$.
7: Create a new element of the dataset with configuration $x^{(t)}$ concatenated with the performances in the earliest epochs and final performance $r^{(t)}$.
8: $t = t + 1$
9: **end while**

Algorithm 2.2 Probability allocation algorithm

1: Initialize each vector V_i with possible values of the hyperparameters chosen. Set $v_i = \#V_i$
2: Initialize each vector P_i of associated probabilities with the same size of V_i.
3: **for** $i \in \{1, \ldots, n_{param}\}$ **do**
4: **for** $j \in \{1, \ldots, v_i\}$ **do**
5: Find the set D^i_j of all the dataset elements that have the feature a^i_j,
6: Sum all the performances of the elements in D^i_j and store the sum in P^j_i.
7: **end for**
8: Normalize the vector P_i to obtain a probability vector.
9: **end for**

At each step of the exploration Algorithm 2.1, a value is chosen for each hyperparameter; this choice is made according to the probability distribution. At this point, the characteristics of the new CNN, which will be trained only in the very first epochs, have been fixed. Therefore, based on the vector $x^{(t)}$ containing the characteristics that define the CNN and the performances in the earliest epochs, the SVR and RF methodologies predict the performance of the CNN at the end of the learning process. This final performance, is considered as a *fitness*, denoted as $r^{(t)}$. At this point, the new element constituted of both $x^{(t)}$ and the performances in the earliest epochs, is a new element of the dataset with final performance $r^{(t)}$ computed with the SVR and/or the RF predictors. Now we have a dataset with an additional element; we compute again all the vectors P_i as at the beginning. Obviously, each update must always keep probability values in the vectors P_i, i.e., all values must remain in the [0, 1] range and the entries of each vector must add up to 1. In this way during the hyperparameters space exploration

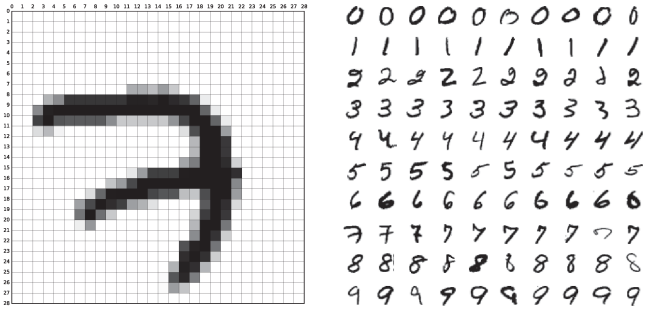
the probability vectors are dynamically modified; the exploration stops, with respect to an element of the hyperparameters space, when one value of this hyperparameter reaches a probability higher than a certain threshold s_i. The whole process is stopped when all the elements of the hyperparameters space are fixed or a maximum number of iterations is achieved. Finally, the 10 best configurations, according to the prediction technique, are trained to convergence and the hyperparameters configuration that leads to the lowest discrepancy is chosen.

3. Numerical experiments

In order to evaluate the behavior of the proposed method, we report the results of a set of numerical experiments in the imaging framework. In the first application we use all the thee phases of our approach and we discuss the obtained results. The second application concerns a comparison of the proposed performance predictor with NAS solutions in the context of NAS-Bench dataset.

3.1. An artifact removal application

We consider a CNN for removing noise from an image. Many medical diagnostic tools (to mention just one example) have the tendency to create artifacts on the image during image acquisition [4]. Such artifacts frequently hinder the reading and the understanding of the image itself. Generally, filters are added after the image acquisition to remove the artifacts, but this process can take a long time for two reasons. Firstly, these filters often depend on a set of hyperparameters whose setup requires several tests and secondly, they are slow in the inference phase. Being global filters on the image, they have to consider the image in its entirety, making the method hard to parallelise. Hence the idea is to generate a CNN for this purpose, which learns how to remove specific types of artifacts from an image. Once trained, such a network would be able to clean the image from the acquired artifacts in a short time, also thanks to the possibility of parallelisation of convolutional filters. In particular, the experiment we consider reproduces a very frequent artifact in the field of image acquisition for medical diagnostics: the presence of horizontal stripes. The original and not corrupted images are taken from the MNIST (Modified National Institute of Standards and Technology database) database. The MNIST database is a collection of handwritten digits commonly used to test various classification methodologies on images (see Figure 1); in our case we use images without labels, because our interest is not their classification, but the denoising of corrupted images generated from them.



(**a**) MNIST sample belonging to the digit '7'. (**b**) 100 samples from the MNIST training set.

Figure 1. MNIST database.

The images of the database are in grayscale (0-255); after a normalization in the range $[0, 1]$, we

center each image in a 28×28 pixel box. Since we need a database with images having horizontal stripes, we have to pre-process the MNIST database by adding horizontal stripe artifacts. We chose to add artifacts by using a simple cycle that adds random horizontal stripes in the original images.

In Figure 2 we show some examples of true and corresponding corrupted images. At this point we use images with stripes as input and images without stripes as labels for the CNN. We detail now the main features of the three phases of the approach described in Section 2 for facing this imaging problem.

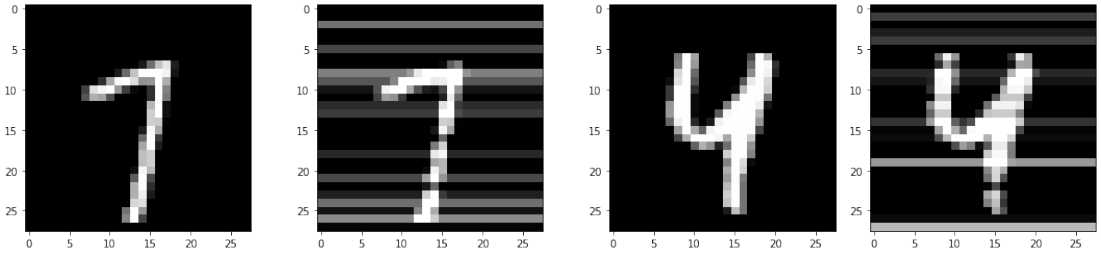


Figure 2. Some samples from MNIST database, without and with stripes.

First phase: generation of a dataset of examples. The first step of the method must be the choice of the CNN hyperparameters whose optimal configuration we want to find. In particular, as hyperparameters to analyse, we select the mini-batch size, the steplength, the optimiser type, the number of filters and the kernels size of a convolutional layer. In Description 3.1 we report the set of values which the selected hyperparameters can assume. In particular, V_3 is the set of the optimisation algoritms within which to choose to train a CNN, i.e., the Stochastic Gradient Descent (SGD) method [7], the SGD method with Momentum [20] and the AdaM scheme [19].

Description 3.1 Generation of the hyperparameters space

1: Choose hyperparameters to optimise: (1) SL-steplength, (2) MB-mini-batch size, (3) OPT-optimiser, (4) NK-filters number, (5) K-kernels size
2: Choose bounds for the hyperparameters and possible configurations:

- $V_1 = \{1e-3, 1e-2, 0.5e-1, 1e-1\}$
- $V_2 = \{32, 128, 1024\}$
- $V_3 = \{SGD, Momentum, AdaM\}$
- $V_4 = \{1, 4, 8, 16\}$
- $V_5 = \{3, 5, 7\}$

We remark that we consider a CNN with 4 convolutional layers; as a consequence, each layer has 12 different configurations, because it can have $1, 4, 8, 16$ filters number and $3, 5, 7$ kernels dimension. As can be seen in Description 3.1, the entire dataset is composed of 746496 samples. We decide to sample the space of the hyperparameters estimating 0.01% of all possible configurations; hence we need to train until numerical convergence only 75 CNNs. We remark that these configurations are chosen with uniform distribution in the hyperparameters space.

As for the other CNN hyperparameters, the hidden layers of the network are convolutions with a padding which generates an output with the same size of the input and the activation function is ReLU

at each layer. In Figure 3 we show an example of one of the CNNs considered. In order to train the 75 selected CNNs, we establish to minimise the Mean Square Error (MSE) as loss function. In particular, we consider MSE between the processed image obtained from the CNN and the corresponding true image in the MNIST database. The CNNs are trained on a subset of images of the MNIST database (training set) while the related final performances are computed on another subset (testing set). Indeed, the MNIST database is already subdivided into 60000 images representing the training set and 10000 images representing the testing set. The elements of the testing set must be considered unknown examples, useful only to verify the effectiveness of the method. The images of the training set are further subdivided into a validation set, which contains 5000 images, and in the actual training with 55000 images. The validation set will be used also to implement the stopping criterion, i.e., the early stopping, as explained in the following.The optimisation algorithm for the CNN training ends when a stopping criterion is satisfied or within a maximum of 20 epochs. To avoid overfitting, we use the well known early stopping technique. As known in the literature, if the learning phase is too long, all the ML methodologies can lead to be too adherent to the data and unable to provide a general model. For this reason, as stopping criterion in the learning phase, we use the early stopping technique, on the validation set, with patient parameter set equal to 3, i.e., after 3 checks without improvement (on the validation set), we stop the learning process and the iteration corresponding to the obtained minimum loss value is considered as the solution and it enables us to process new samples.

The generated dataset consists of a table in which each row corresponds to the set of hyperparameters for the training of a network together with the obtained performances on the training and validation sets at each early epochs. As metric for evaluating the final performance, we use the average MAE between the processed images and the corresponding true ones on the testing set.

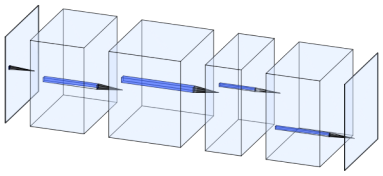


Figure 3. A possible CNN scheme.

Second phase: performance predictor. Next, as explained in Section 2, an SVR and a RF methodologies are trained on the generated dataset, by using for any example the final performance as label. We use cross validation techniques to set up SVR kernels and RF options. Even if it takes time, this preliminary step can lead to a less expensive hyperparameter optimisation process than those already known in the literature. The aim of the method is to predict the final behavior of a network, after acquiring the performance metrics only at the first epochs. A crucial point is to fix how many initial epochs to consider and how to use SVR and RF together.

As for the first issue, since we consider at most 20 epochs for each numerical convergence training, it is reasonable to use no more than 5 epochs for training the CNNs with hyperparameter configurations not belonging to the training dataset. By carrying out an extensive experimentation, we choose the number of initial epochs to take into account, by evaluating the final predicted results obtained by employing the performances at 0/1/2/3/4/5 epochs. The results of the numerical tests reported in

Section 3.1.1 suggest to fix the number of the initial epochs equal to 3; this value represents a good compromise between the training time and the predictor performance. As for the second issue, the same numerical tests also allow to evaluate the effectiveness of the predictors obtained with the SVR, the RF and the HYBRID metodologies. We observe that the HYBRID one provides the best predictions.

Third phase: from predictor to HO. We use the predictor obtained in the previous phase to explore the space of the hyperparameters, by searching the configuration with the probability of best performance. This procedure is stopped when the maximum number of iterations (set equal to 400) is reached or when the probabilities P_i for each i of the current configuration are greater than $\frac{2}{\#V_i}$. Furthermore, when the probability of one of the hyperparameters reaches this threshold, the corresponding value is fixed for the next iterations.

3.1.1. Results

During the generation of the CNN dataset (first phase), we observe that the networks have very different performances. We can classify the performance of the networks in terms of the provided reconstructed testing images. In particular, we fix three macro-categories of reconstructions: good, intermediate and very bad ones.

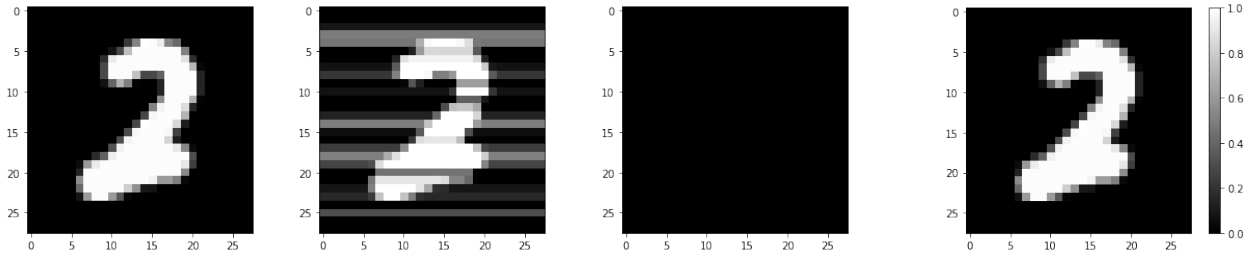
In the first case a well-configured network can lead to a MAE value around 0.008 or less, providing a reconstruction almost identical to the image without stripes. In the intermediate case, the MAE is raised by an order of magnitude; we find values around 0.08, leading to a reconstruction with some perceptible artifacts, but the image is still well recognizable. In the worst case, the MAE increases, with values greater than 0.1. In these cases, the resulting image is completely flattened and both the artifacts and the image itself disappear. We show an example for each of the three subsets. In Figures 4–6, for each configuration, we compare the original image, the input image with artifacts, the output image and the difference between the predicted and the original images.

As for the choice of the number of the initial epochs to take into account at the second phase, we plan a set of numerical experiments. In particular, for a fixed number of the initial epochs, starting from the CNN dataset generated in the first phase, we consider several training and testing sets on which the different predictors of the performance are trained and tested. Any testing set has been created by randomly selecting 10 CNN configurations of the whole generated dataset; then the corresponding 10 prediction labels, i.e., 10 values of MAE, are obtained and the MSE of these 10 values is computed. On all the generated training and testing sets, we determine the average MSE values and the corresponding standard deviations.

These results are reported in Tables 1 and 2 for 0/1/2/3/4/5 initial epochs and for the three predictors provided by the SVR, RF and HYBRID methodologies. In particular, we consider 100 different training and testing sets in Table 1 and 1000 in Table 2. The numerical results of these tables show that the HYBRID methodology seems to provide the best predictions, also showing to be the most robust method with the lowest standard deviations. Furthermore, a value equal to 3 initial epochs represents a good compromise between network training time and predictor performance.

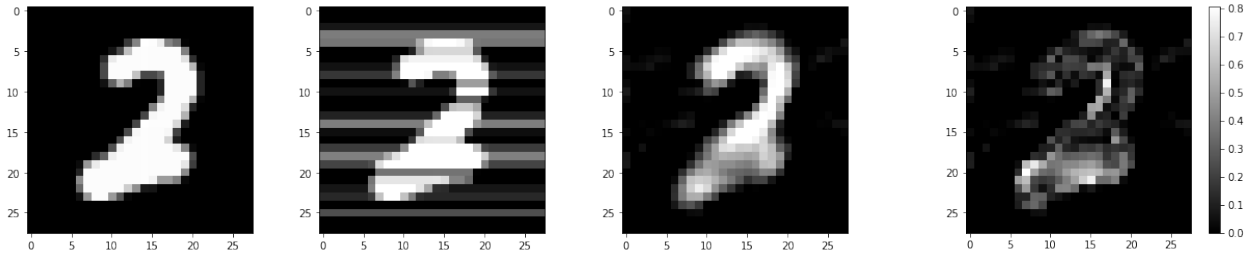
In order to evaluate the effectiveness of the predictor on a prefixed testing set, we compare the label of each network (REAL MAE) with the corresponding Predicted MAE. Table 3 shows these results for such testing elements. We observe the good behavior of the predictor, which is able to well distinguish

between the three classes of CNNs belonging to the dataset generated at the first phase.



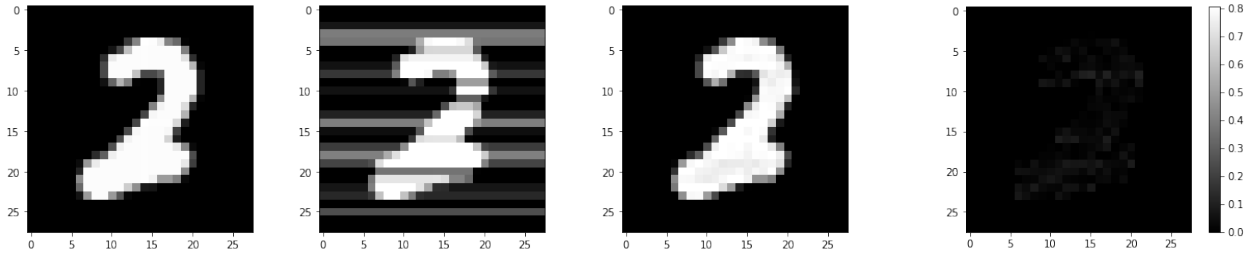
Original image | Image with artifacts | Output of the network | Difference map between the predicted and the original images

Figure 4. The very bad case.



Original image | Image with artifacts | Output of the network | Difference map between the predicted and the original images

Figure 5. The intermediate case.



Original image | Image with artifacts | Output of the network | Difference map between the predicted and the original images

Figure 6. The good case.

Once the predictor is trained, it can be employed to explore the space of hyperparameters and find the optimal configuration to obtain good reconstructed images. As already mentioned, in order to predict the performance of the current configuration, at the step 5 of the Algorithm 2.1 only 3 epochs are calculated. As a consequence, from a computational point of view, the time required by the method is equal to that required by 1200 epochs (when all the prefixed 400 iterations are performed as in our experiment). Since the single network takes at most 20 epochs, the whole search process is computationally equivalent to the total training of 60 networks. At the end of the process we take the 10 best hyperparameter configurations, based on the measure provided by the predictor, and we

drive them to numerical convergence. To summarise, the overall number of CNNs driven to numerical convergence is given by 145, by taking into account the 75 ones to construct the dataset and the 70 ones for determining the best reconstructed images. The number of CNNs to be trained to obtain an efficient configuration looks very promising compared to the 746496 possible hyperparameter configurations.

Table 1. Average MSE values and related standard deviations for 100 random testing sets, obtained by the SVM, RF and HYBRID methodologies by taking into account performances at different numbers of initial epochs (0/1/2/3/4/5).

	0	1	2	3	4	5
SVR	**0.0463**	0.0228	0.0163	0.0157	0.0155	0.0153
	±0.0088	±0.0079	±0.0067	±0.0057	±0.0057	±0.0049
RF	0.0512	0.0286	0.0170	0.0165	0.0145	0.0147
	±0.0099	±0.0084	±0.0070	±0.0070	±0.0065	±0.0056
HYBRID	0.0471	**0.0228**	**0.0149**	**0.0140**	**0.0133**	**0.0128**
	±0.0077	±0.0075	±0.0063	±0.0056	±0.0052	±0.0049

Table 2. Average MSE values and related standard deviations for 1000 random testing sets, obtained by the SVM, RF and HYBRID methodologies by taking into account performances at different numbers of initial epochs (0/1/2/3/4/5).

	0	1	2	3	4	5
SVR	0.0464	0.0227	0.0168	0.0153	0.0156	0.0160
	±0.0098	±0.0074	±0.0062	±0.0056	±0.0057	±0.0056
RF	0.0499	0.0278	0.0163	0.0157	0.0156	0.0151
	±0.0102	±0.0083	±0.0066	±0.0065	±0.0068	±0.0067
HYBRID	**0.0463**	**0.0255**	**0.0147**	**0.0135**	**0.0135**	**0.0133**
	±0.0086	±0.0070	±0.0055	±0.0052	±0.0055	±0.0055

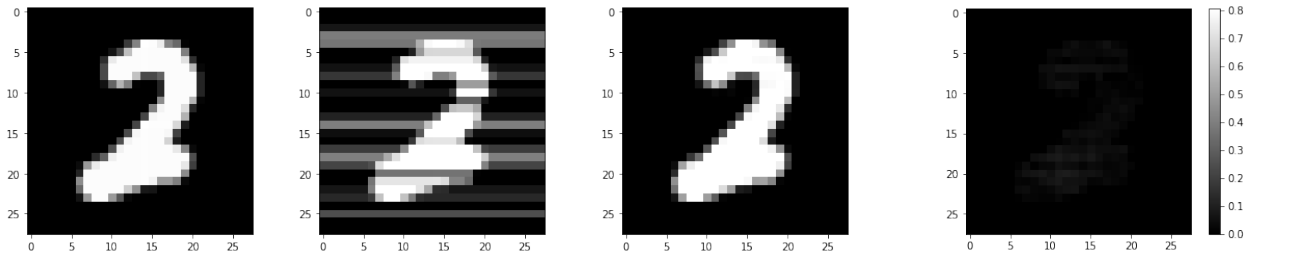
Table 3. HYBRID predictions for MAE measure on 10 samples of the dataset.

REAL MAE	Predicted MAE
0.0406	0.0450
0.1325	0.0968
0.0152	0.0139
0.0304	0.0343
0.0071	0.0152
0.1934	0.1389
0.0638	0.0937
0.0387	0.0425
0.0511	0.0415
0.0048	0.0101

In Table 4 the 10 best hyperparameter configurations are shown. We remark that the best MAE obtained at numerical convergence (REAL MAE) and the one predicted (Predicted MAE) refer to the same hyperparameter configuration and we underline that in the first phase the best CNN configuration had final MAE value equal to 0.0059, significantly improved by the last phase in which the best CNN configuration had final REAL MAE value equal to 0.0048. The Figure 7 shows one of the reconstructed images obtained from this CNN.

Table 4. Features and performance for the best 10 predicted CNNs. NK_i and K_i denotes the filters number and the kernel size respectively of the i-th layer, $i = 1, \ldots, 4$.

OPT	SL	MB	NK_1	NK_2	NK_3	NK_4	K_1	K_2	K_3	K_4	REAL MAE	Predicted MAE
2	1e-1	32	4	8	16	4	7	3	7	7	**0.0048**	**0.0063**
3	1e-3	128	8	16	8	16	7	5	7	3	0.0049	0.0064
2	5e-2	128	8	8	4	4	5	5	3	7	0.0079	0.0069
2	1e-2	32	8	8	16	4	3	5	7	7	0.0083	0.0075
2	1e-3	32	4	4	4	1	5	5	5	5	0.0221	0.0084
2	1e-1	32	8	8	16	1	5	5	3	7	0.0050	0.0086
3	1e-2	32	4	16	8	4	7	5	3	7	0.0064	0.009
3	1e-2	32	4	16	8	4	3	5	3	7	0.0055	0.0094
3	1e-1	32	8	8	4	1	3	3	7	7	0.0068	0.0102
2	1e-1	32	8	16	4	4	7	3	3	7	0.0057	0.0106



Original image | Image with artifacts | Output of the network | Difference map between the predicted and the original images

Figure 7. One of the reconstructed images obtained from the CNN related to the best hyperparameter configuration.

3.1.2. Ablation study

In this section we conduce an ablation study to highlight which features, among the ones employed, are more relevant for the performance predictor model. In particular we distinguish between the hyperparameters connected with the optimiser (type of optimisers, learning rate and minibatch size) and the other ones connected to the CNN architecture (number of kernels and kernel sizes). Moreover, we train the three predictors provided by the SVR, RF and HYBRID methodologies by taking into account or not the performance related to the first three epochs.

In Table 5, for 1000 random testing sets, we report the average MSE values and the related standard deviations obtained by the SVR, RF and HYBRID methodologies by considering different combinations of the input feature subsets. OPT refers to all the hyperparameters connected with the optimiser, ARCH refers to all the hyperparameters connected with the architecture and IP (Initial Performance) refers to the performance of the first three epochs. As can be expected, the initial performance is the most important feature. Indeed, from Table 5, we observe that IP alone achieves a good average MSE values for all the three methodologies. On the other hand, we observe that the hyperparameters connected with the optimiser are more relevant for the performance predictor than those connected with the CNN architecture. Finally, we highlight that, in general, the best results can be achieved by considering all the three subsets OPT, ARCH and IP.

Table 5. Average MSE values and related standard deviations for 1000 random testing sets, obtained by the SVR, RF and HYBRID methodologies by taking into account different combinations of the hyperparameter subsets.

OPT	ARCH	IP	SVR	RF	HYBRID
✓			0.0260	0.0408	0.0320
			±0.0109	±0.0090	±0.0094
	✓		0.0463	0.0547	0.0495
			±0.0115	±0.0120	±0.0104
		✓	0.0190	0.0145	0.0146
			±0.0085	±0.0068	±0.0063
✓		✓	0.0175	0.0145	0.0140
			±0.0085	±0.0070	±0.0066
	✓	✓	0.0177	0.0149	0.0143
			±0.0069	±0.0066	±0.0061
✓	✓		0.0464	0.0499	0.0463
			±0.0098	±0.0102	±0.0086
✓	✓	✓	0.0153	0.0157	0.0135
			±0.0056	±0.0065	±0.0052

3.2. The method applied to NAS-Bench-101

In order to perform a comparison to competing NAS solutions, we provide an additional experiment by using the NAS-Bench-101 dataset [24]. First of all we highlight the main features of this dataset and then, taking into account these peculiarities, we describe how to apply the proposed method to NAS-Bench-101 dataset.

3.2.1. Characteristics of NAS-Bench-101

NAS-Bench-101 is a public architecture dataset for NAS research. The NAS-Bench-101 search space is composed by $423k$ unique convolutional architectures (423624 to be precise). All of these architectures were trained and evaluated multiple times on the well-known CIFAR-10 image database and the results were compiled into a large dataset of over 5 million trained models. The aim is that,

using this benchmark, the researchers can evaluate the quality of a diverse range of models in milliseconds by querying the precomputed dataset.

Each of the 423634 unique modules within the search space has been trained three times for 4, 12, 36, and 108 epochs ($423K * 3 * 4 \simeq 5M$ total trained models). For any module, the following metrics are recorded in the dataset:

- training accuracy
- validation accuracy
- testing accuracy
- training time.

In summary, NAS-Bench-101 is a tabular dataset which maps CNN architectures to their trained and evaluated performances on CIFAR-10. Specifically, all networks share the same feedforward "skeleton". What distinguishes the different architectures is a collection of neural operations linked in an arbitrary graph-like structure. Each network is represented by a directed acyclic graph with up to 7 vertices and 9 edges. The valid operations at each vertex can be "3x3 convolution", "1x1 convolution", and "3x3 max-pooling". For further details, see [24]. Therefore, we highligh that the dataset does not take into account the optimizer, nor the hyperparameters connected to it, like mini-batch size, learning rate and optimizer type.

In order to apply the proposed method, we proceed by extracting a small subset of the dataset. In the second phase, where the performance predictor has to be trained, we use as features of any CNN only its performance at the epochs 4, 12 and 36, measured on the CIFAR10 training and validation sets, by discarding (as in the previous application) the metrics related with the testing set. With 9 input values (training accuracy, training time, validation accuracy at 4, 12, 36 epochs) for any element of CNN extracted dataset, we train the SVR, RF and HYBRID methodologies. As a consequence, given the features of a unseen CNN, we can predict its final performance on the CIFAR10 testing set (i.e., at 108 epoch).

In order to evaluate our approach with respect to the state-of-the-art in [24], in the third phase we use the search techniques described in [24], i.e., random search and evolution search (in the regularized version), by using for each CNN of NAS-Bench-101 database the performance predictor trained in the second phase instead of the final performance recorded in the database. Finally a comparison between our results and the ones obtained in [24] is presented. The following results are obtained by using the software downloadable at https://github.com/google-research/nasbench.

3.2.2. Results for NAS-Bench-101

In this subsection we present the details and the results of the three different phases.

First phase. In the first phase we extract 423 different CNN architectures from NAS-Bench-101 with their metrics measured at epochs 4, 12, 36 and 108. We highlight that, as in the application of the previous section, the size of the CNN training set is equal to 0.01% of the whole dataset. We use 9 input features, as we said before, and as label the accuracy measured on the CIFAR10 testing set at epoch 108. In this manner we build the initial dataset. Because of the availability of NAS-Bench-101 dataset, the first phase becomes quite cheap, since we can neglect the time used to train the 423 different architectures until numerical convergence.

Second phase. In the second phase we use the extracted dataset to train an SVR and a RF. As in the previous application, also the results obtained by the HYBRID method are carried out. In particular, we use 400 samples from the extracted set to train SVR and RF; then, the remaining 23 elements are considered as unseen CNNs, so that they enable us to evaluate the effectiveness of the performance predictor. For the optimization of the SVR and RF hyperparameters, the Optuna framework [1] is used. Optuna is an automatic hyperparameter optimization software framework, especially designed for machine learning. In Figure 8, we show the results obtained by the three methodologies (SVR, RF and HYBRID) for the 23 unseen elements of the extracted set, also compared with the CNN performance at epoch 108 (in red in figure). In Table 6, we report the mean error (MSE) over the 23 unseen CNNs in the case of the three methodologies.

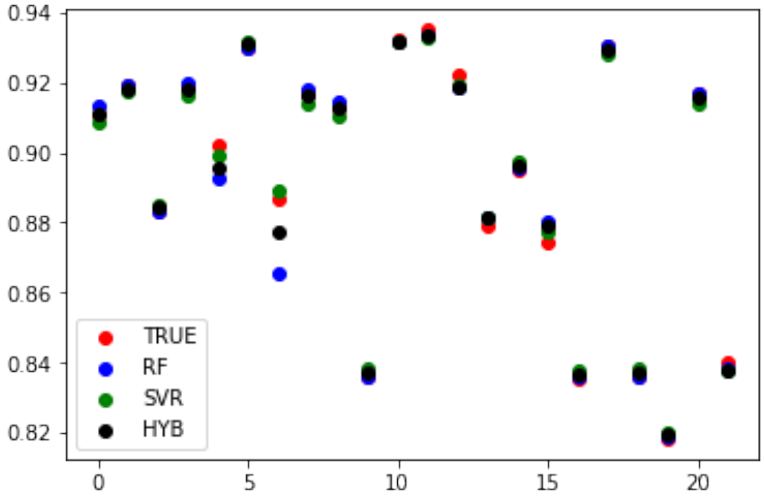


Figure 8. Predicted and real accuracy values obtained by the SVR, RF and HYBRID methodologies for the 23 CNNs used as unseen data.

Table 6. MSE values for the SVR, RF and HYBRID methodologies over the 23 CNNs used as unseen data.

	RF	SVR	HYBRID
MSE	5.6147e-06	2.6950e-05	1.8964e-06

Third phase. In the last phase, we use the three performance predictors inside two different search strategies proposed in [24]. The first strategy is a simple random search and the second an evolution search in the regularised version. In [24] the authors used a prefixed budget time T to run the two different search strategies. In particular, for any CNN involved in the current search step, the related training time at 108 epoch is subtracted until the end of the time. In order to obtain a fair comparison with our approach, we run the same methodologies with a time budget obtained by subtracting from the original value T the time needed to create the initial dataset (423 CNNs). Then for any CNN involved in the current search step, the accuracy at the 108 epoch is predicted by SVR or RF or HYBRID methodologies, and the training time at epoch 36 is subtracted from the current budget time until the

end of the time. In the Figures 9 and 10 we show the results obtained for random search and evolution search (in the regularized version). In particular, in each figure we compare the accuracy behavior at epoch 108 (measured on the CIFAR10 testing set) of the current CNN with respect to the training time for the standard method (red line) and the ones obtained by using the SVR (blue line), RF (green line) and HYBRID (black line) prediction values of the final performance. We observe that in the random case, the accuracy behavior obtained by using the HYBRID predictor in quite similar to the one retrieved by standard approach. Also in the evolution search strategy, the use of RF and HYBRID methodologies as predictors provides an accuracy behavior very similar to the one obtained with the standard technique.

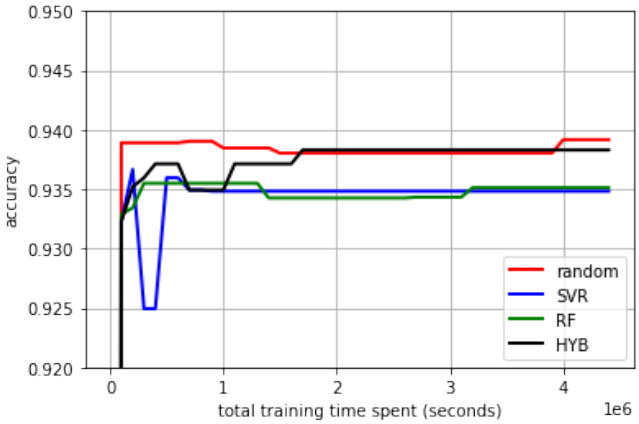


Figure 9. Random search: the red line denotes the accuracy behavior obtained by the final performance of the current CNN as in [24], while the other blue, green and black lines show the accuracy behavior with respect to the training time when the SVR or RF or HYBRID methodology is used as predictor of the final performance of the CNN.

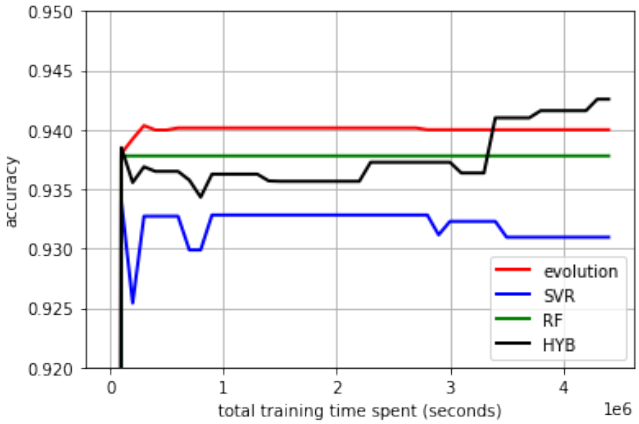


Figure 10. Evolution search (regularized): the red line denotes the accuracy behavior obtained by the final performance of the current CNN as in [24], while the other blue, green and black lines show the accuracy behavior with respect to the training time when the SVR or RF or HYBRID methodology is used as predictor of the final performance of the CNN.

4. Conclusions

To be trained, a CNN usually requires a very expensive numerical investigation in terms of time and computational cost. Particularly, the selection of proper values for the hyperparameters, defining both the architecture of the network and the optimisation algorithm exploited for the training, may be computationally very demanding. Moreover, a bad hyperparameter configuration can lead to inefficient CNNs. For these reasons, in this paper we proposed an automatic Machine Learning procedure to set such hyperparameters with the aim of optimising the network performance. To reach this goal, we develop a low-cost strategy able to predict the performance of a CNN given its hyperparameter setting and its behavior after only few steps of the training process. The effectiveness of the suggested approach has been tested on a set of numerical experiments in the framework of image reconstruction. The proposed method (first, second and third phases) was evaluated on a artifact removal application with promising results. To provide a comparison with the state-of-the-art NAS solutions, the performance predictor described in the second phase of our approach was applied in the context of NAS-Bench dataset, by obtaining very comparable results.

Acknowledgments

This work has been partially supported by the INdAM research group GNCS and by POR-FSE 2014-2020 funds of Emilia-Romagna region (Deliberazione di Giunta Regionale n. 255- 30/03/2020).

Conflict of interest

The authors declare no conflict of interest.

References

1. T. Akiba, S. Sano, T. Yanase, T. Ohta, M. Koyama, Optuna: A next-generation hyperparameter optimization framework, In: *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019, 2623–2631. http://dx.doi.org/10.1145/3292500.3330701
2. B. Baker, O. Gupta, R. Raskar, N. Naik, Accelerating neural architecture search using performance prediction, 2017, arXiv:1705.10823.
3. B. Baker, O. Gupta, N. Naik, R. Raskar, Designing neural network architectures using reinforcement learning, 2017, arXiv:1611.02167.
4. J. F. Barrett, N. Keat, Artifacts in CT: recognition and avoidance, *RadioGraphics*, **24** (2004), 1679–1691. http://dx.doi.org/10.1148/rg.246045065
5. J. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, Algorithms for hyper-parameter optimization, In: *Advances in Neural Information Processing Systems*, 2011, 2546–2554.
6. J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, *J. Mach. Learn. Res.*, **13** (2012), 281–305.

7. L. Bottou, F. E. Curtis, J. Nocedal, Optimization methods for large-scale machine learning, *SIAM Rev.*, **60** (2018), 223–311. http://dx.doi.org/10.1137/16M1080173
8. L. Breiman, Random forests, *Machine Learning*, **45** (2001), 5–32. http://dx.doi.org/10.1023/A:1010933404324
9. C. J. C. Burges, A tutorial on support vector machines for pattern recognition, *Data Mining and Knowledge Discovery*, **2** (1998), 121–167. http://dx.doi.org/10.1023/A:1009715923555
10. H. Cai, T. Chen, W. Zhang, Y. Yu, J. Wang, Efficient architecture search by network transformation, 2017, *arXiv/1707.04873*.
11. T. Domhan, J. T. Springenberg, F. Hutter, Speeding Up Automatic Hyperparameter Optimization of Deep Neural Networks by Extrapolation of Learning Curves, In: *IJCAI International Joint Conference on Artificial Intelligence*, 2015, 3460–3468.
12. T. Elsken, J. H. Metzen, F. Hutter, Neural architecture search: a survey, *J. Mach. Learn. Res.*, **20** (2019), 1997–2017.
13. T. Elsken, J.-H. Metzen, F. Hutter, Simple and efficient architecture search for convolutional neural networks, 2017, arXiv: 1711.04528.
14. G. Franchini, M. Galinier, M. Verucchi, Mise en abyme with artificial intelligence: how to predict the accuracy of NN, applied to hyper-parameter tuning, In: *INNSBDDL 2019: Recent advances in big data and deep learning*, Cham: Springer, 2020, 286–295. http://dx.doi.org/10.1007/978-3-030-16841-4_30
15. D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*, Addison Wesley Publishing Co. Inc., 1989.
16. T. Hospedales, A. Antoniou, P. Micaelli, A. Storkey, Meta-learning in neural networks: a survey, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, in press. http://dx.doi.org/10.1109/TPAMI.2021.3079209
17. F. Hutter, L. Kotthoff, J. Vanschoren, *Automatic machine learning: methods, systems, challenges*, Cham: Springer, 2019. http://dx.doi.org/10.1007/978-3-030-05318-5
18. F. Hutter, H. Hoos, K. Leyton-Brown, Sequential model-based optimization for general algorithm configuration, In: *LION 2011: Learning and Intelligent Optimization*, Berlin, Heidelberg: Springer, 2011, 507–523. http://dx.doi.org/10.1007/978-3-642-25566-3_40
19. D. P. Kingma, J. Ba, Adam: a method for stochastic optimization, 2017, arXiv:1412.6980.
20. N. Loizou, P. Richtarik, Momentum and stochastic momentum for stochastic gradient, Newton, proximal point and subspace descent methods, *Comput. Optim. Appl.*, **77** (2020), 653–710. http://dx.doi.org/10.1007/s10589-020-00220-z
21. J. Mockus, V. Tiesis, A. Zilinskas, The application of Bayesian methods for seeking the extremum, In: *Towards global optimisation*, North-Holand, 2012, 117–129.
22. B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, N. De Freitas, Taking the human out of the loop: A review of bayesian optimization, *Proc. IEEE*, **104** (2016), 148–175. http://dx.doi.org/10.1109/JPROC.2015.2494218
23. S. Thrun, L. Pratt, Learning to learn: introduction and overview, In: *Learning to learn*, Boston, MA: Springer, 1998, 3–17. http://dx.doi.org/10.1007/978-1-4615-5529-2_1

24. C. Ying, A. Klein, E. Real, E. Christiansen, K. Murphy, F. Hutter, NAS-Bench-101: Towards reproducible neural architecture search, In: *Proceedings of the 36–th International Conference on Machine Learning*, 2019, 7105–7114.

25. Z. Zhong, J. Yan, W. Wei, J. Shao, C.-L. Liu, Practical block-wise neural network architecture generation, In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, 2423–2432. http://dx.doi.org/10.1109/CVPR.2018.00257

26. B. Zoph, Q. V. Le, Neural architecture search with reinforcemente learning, 2017, arXiv:1611.01578.

AIMS Press

© 2023 the Author(s), licensee AIMS Press. This is an open access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0)