PBIL for optimizing inception module in convolutional neural networks

PEDRO GARCÍA-VICTORIA[*], *Department of Computer Science and Artificial Intelligence ETSII University of Seville, Avda. Reina Mercedes s/n, 41012 Seville, Spain.*

MIGUEL A. GUTIÉRREZ-NARANJO[**], *Department of Computer Science and Artificial Intelligence ETSII University of Seville, Avda. Reina Mercedes s/n, 41012 Seville, Spain.*

MIGUEL CÁRDENAS-MONTES[†], *Department of Basic Research, Centro de Investigaciones Energéticas Medioambientales y Tecnológicas (CIEMAT), Avda. Complutense 40, 28040 Madrid, Spain.*

ROBERTO A. VASCO-CAROFILIS[††], *Group for Vision and Intelligent Systems, Electrical and Systems Engineering and Automation Department, University of León, Campus de Vegazana, s/n 24071 León, Spain.*

Abstract

Inception module is one of the most used variants in convolutional neural networks. It has a large portfolio of success cases in computer vision. In the past years, diverse inception flavours, differing in the number of branches, the size and the number of the kernels, have appeared in the scientific literature. They are proposed based on the expertise of the practitioners without any optimization process. In this work, an implementation of population-based incremental learning is proposed for automatic optimization of the hyperparameters of the inception module. This hyperparameters optimization undertakes classification of the MNIST database of handwritten digit images. This problem is widely used as a benchmark in classification, and therefore, the learned best configurations for the Inception module will be of wide use in the deep learning community. In order to reduce the carbon footprint of the optimization process, policies for reducing the redundant evaluations have been undertaken. As a consequence of this work, an evaluation of configurations of the inception module and a mechanism for optimizing hyperparameters in deep learning architectures are stated.

Keywords: deep learning, convolutional neural networks, optimization, PBIL, Gray coding, Hamiltonian path

1 Introduction

Computer vision is one of the areas with a larger portfolio of successful applications in deep learning. Part of this success stems from the use of relatively simple convolutional structures which are in turn

[*]E-mail: pedrogarciavictoria@gmail.com
[**]E-mail: magutier@us.es
[†]E-mail: miguel.cardenas@ciemat.es
[††]E-mail: rvasc@unileon.es

https://doi.org/10.1093/jigpal/jzac022

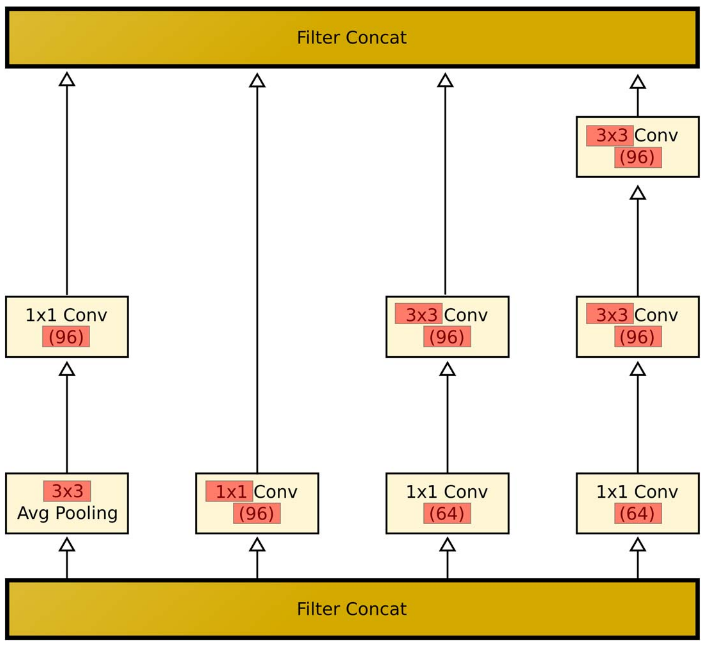

FIGURE 1. Schema of Inception-v4 module. Red rectangles indicate the elements of the configuration of this module that are being manipulated by the PBIL algorithm. This schema can be replicated up to 3 times for conforming the final architecture.

repeated with small variations, until very deep architectures are built. One of the most used structures is the inception module [19].

From the first implementation, inception module has been altered by new proposals, which maintain the essential of the module: several parallel branches of convolutional blocks, including stack of blocks, with different kernel sizes and a `maxpooling` or `average pooling` layers (see [18] for a review). In all the cases, the choice of the kernel sizes and the number of kernels are inherited from the reference publications and in few times alternative sizes are evaluated.

In this work, an implementation of Population-based Incremental Learning (PBIL) [2, 3] is used to evolve the Inception-v4 module. The final objective is to evaluate the suitability of the kernel configurations in this module and alternatively to propose other high-quality configurations. The classification of the MNIST database of handwritten digit images is used as a benchmark for this purpose [13]. The choice of this classification benchmark comes from its wide diffusion in the deep learning community. Thus, the use of MNIST as benchmark allows explore the suitable configurations found by the evolutionary algorithm for the inception module; moreover, the use of a well-known dataset as MNIST will help to an easy dissemination among the community.

PBIL is a population-based evolutionary algorithm, in which the probability presence of certain information in genes is evolved. In the current problem, this information includes the three kernel sizes, the filter sizes, the max-pooling size in the Inception-v4 module, as well as the number of consecutive Inception-v4 modules (see Section 2.4 and Figure 1 for further details).

Hyperparameters under the PBIL evolution process are binary coded. In order to avoid Hamming cliffs, Gray coding is used to encode the hyperparameters to be optimized (see Section 2.2). With

Gray coding, adjacent numbers differ only in one in Hamming distance. Thus, Gray coding aims at removing barriers in evolutionary process.

Green artificial intelligence pledges for reducing the carbon footprint of their algorithms [1]. The scientific literature alerts that the training of certain deep architectures involves the emissions of the equivalent five times the lifetime of an average car including its manufacturing [17]. Our proposal is aware of AI carbon footprint, and for this reason, policies for minimizing the unwanted evaluations of individuals are implemented.

Inside the inception rationale, the branches are configured with different kernel sizes. The different sizes of the receptive fields in convolutional neural networks (CNNs) aim to capture information at different scales: the larger the receptive field, the more generic are the features extracted from the images; whereas in the opposite sense, the smaller the kernel sizes, the more local are the features extracted.

For this reason, it is reasonable to avoid the evaluation of individuals with duplicated kernel sizes. For avoiding the evaluation of these individuals with repeated kernel sizes in different branches, policies have been implemented in association with the Gray coding (see Section 2.2).

This proposal is inspired by previous efforts for improving the performance of the forecasting of the ^{222}Rn time series at Canfranc Underground Laboratory (LSC) and air quality. In the past, diverse machine learning algorithms have been used for this purpose, including multilayer perceptron, CNNs and recurrent neural networks [5–7, 15, 16].

In [16], an implementation of using STL decomposition and CNN for improving the forecasting capacity is presented with promising results, but penalized by a larger number of hyperparameters to select based on the practitioners expertise. In order to select the most suitable hyperparameters set, an optimization process based on PBIL was proposed in [20]. Due to the positive results achieved, in the current work we proposed to adapt the methodology to the optimization of the inception module.

The paper is organized as follows: Section 2 gives a brief description of the techniques used in this paper. In Section 3, the results are shown and analysed. Finally, Section 4 contains the conclusions of this work.

2 Methodology

2.1 Population-based incremental learning

PBIL is an optimization method which combines genetic algorithms with competitive learning [2, 3]. It belongs to the so-called estimation of distribution algorithms. Instead of making evolve individuals like in genetic algorithm, the probability distribution of information appearance in the genes is made evolve. The specific operators of PBIL operate over these probability distributions.

In our proposal, the individuals represent binary sequences of a fixed length and they are randomly initialized with a probability 0.5. This is the usual way of initialization for PBIL. The binary sequence is formed by the concatenation of the binary codification of every hyperparameter under optimization.

After their evaluation, the most suitable individuals are selected for updating the probability distribution representing the information under optimization. Then, based on the update probability distribution, a new generation of individuals is created, and again evaluated. The cycle is repeated, and after some generations, the population converges to a set of high-quality individuals.

The fitness function is defined as the sparse categorical crossentropy of a single execution over the validation set of the CNN defined with the hyperparameters codified by the individual. The evaluation of the individual, and hence of the CNN, is performed with a single epoch. Although this seems to be prone to underfit, the deep architecture, later it is demonstrated as an appropriate strategy for saving time processing, at the same time that it does not critically penalize the performance of the network.

In this work, PBIL is configured as follows: population size of 10 individuals evolving during 20 generations, and the mutation probability is 0.05. When mutation is applied, the probability vector is shifted by an amount of 0.1 in a randomly chosen direction. The three best and the three worst individuals are selected for updating the probability distribution. The updated probability distribution approaches the configuration of the best individuals, at the same time that it recedes from the worse ones. Thus, the individuals of the next generation inherit more likely high-performance configurations.

The choice of the population size and the number of generations stems from the computational intensity of the problems. Larger values of these parameters made the evolutionary strategy unfeasible with computational resources available. The mutation probability and the amount of shifted information have been established taking into account two information sources; on the one hand, the previous work with PBIL [20], and on the other hand, a restricted greedy search around the previous best parameters. Finally, the choice of the three worst and best individuals for updating the probability distribution comes from the previous choice of the population size with only 10 individuals.

2.2 Gray coding

Gray coding is a type of binary coding which is usually used to avoid Hamming cliffs. A Hamming cliff is formed when two numerically adjacent values have bit representations that are far apart by using the Hamming distance. For example, number 3 and 4 differ in binary representation in three bits: 0011 and 0100, having a Hamming distance of 3; or for 15 and 16, corresponding the binary representations 01111 and 10000, which have a Hamming distance of 5.

A large Hamming distance is a barrier for the evolution of the individuals in the evolutionary algorithm. The change of one unit—involving diverse bits—in a parameter under optimization requires a large amount of simultaneous modification of the individual binary coding, but not in Gray coding. This degrades the performance of evolutionary algorithms with binary codification.

In order to avoid the Hamming cliffs, the inception module configuration is Gray-coded.

2.3 Convolutional neural networks

CNNs are specialized neural networks with special emphasis in image processing [11, 12]; although, nowadays, they are also employed in time series analysis and forecasting [8, 15, 16, 21].

The CNN consists of a sequence of convolutional layers, the output of which is connected only to local regions in the input. These layers alternate convolutional, nonlinear and pooling-based layers which allow extracting the relevant features of the class of objects, independently of their placement in the data example. The CNN allows the model to learn filters that are able to recognize specific patterns in the time series, and therefore, they can capture richer information from the series. It also embodies three features which provide advantages over the multilayer perceptron: sparse interactions, parameter sharing and equivariance to translation [11].

Although CNN are frequently associated with image or audio classification -2D grid examples- or video sequence -3D grid examples-, it can also be applied to time series analysis -1D grid examples-. When processing time series, instead of a set of images, the series has to be divided in overlapping contiguous time windows. These windows constitute the examples, where the CNN aims at finding patterns. At the same time, the application to time series modelling requires the application of 1D convolutional operators, whose weights are optimized during the training process.

2.4 Gray-code of hyperparameters of inception module

In Figure 1, the schema of the Inception-v4 module is depicted. In this schema, the elements that are handled by the evolutionary algorithm, and therefore could be altered, are pointed with red rectangles. As it can be observed, in all the elements the number of filters can be modified by PBIL.

In our approach, most of the kernel sizes are optimized by PBIL although some kernels with size 1×1 are kept frozen. Behind this decision is the own nature of 1×1 convolutional operation. It allows to reduce the computational intensity by shrinking the number of channels of the tensor of data, at the same time that capturing image features at a very local scale.

Some constraints are implemented in the evolutionary algorithm. For instance, in the right-hand branch the kernels sizes of the two convolutional layers have the same configuration. Besides, except for the left-hand branch for which only the average pooling size is evolving, for the other three branches the kernel sizes are forced to be different. Thus, configurations with equal kernels sizes in any of these three branches are excluded. This aims at capturing features at different scales. Asymmetrical configurations of the kernels sizes, such as 1×7 used in some inception modules (see [19]) are not considered.

An additional hyperparameter controls the number of inception blocks that are stacked in the final architecture. This hyperparameter is also handled by PBIL, and it can take values from 1 to 3.

As it has been mentioned, the three right convolutional branches of the inception module are forced to have different kernels sizes. They are identified by a Gray-coded 3-tuple. To reduce the carbon footprint of the tuples with repeated kernels sizes, instead of to strongly penalize their fitness after their evaluation, a codification that avoids its generation is implemented.

The range of the possible kernels sizes is restricted to {3,5,7,9,11}. By avoiding the repetitions the search space is drastically reduced. Since 5 different kernels sizes are allowed, if repetitions are allowed, $5^3 = 125$ 3-tuples should be considered. If no repeated sizes are allowed, only $5 \times 4 \times 3 = 60$ 3-tuples are considered.

The key point for avoiding Hamming cliffs by Gray coding is to order the codes in such a way that two consecutive codes are at Hamming distance one. In this paper, the same idea is considered for ordering the 3-tuples of sizes of the inception blocks. The process is guided by the following principle: *If C_1 and C_2 are two consecutive gray codings, then their associated 3-tuple of sizes T_1 and T_2 are also at Hamming distance 1.*

In order to reach this target, an abstract general graph G is constructed. The nodes of the graph are the 60 possible 3-tuples of sizes and there is an edge between two nodes if the corresponding 3-tuples are at Hamming distance 1 (see Figure 2). Any possible Hamiltonian path in this graph provides an ordering of the 60 3-tuples satisfying that two consecutive tuples are at Hamming distance 1. In particular, the ordering shown in Table 2 satisfies such condition.

Finally, both sequential orderings, the six-bits Gray codings and the Hamiltonian path of 3-tuples satisfy that all the pairs of consecutive elements are at Hamming distance one. In order to compute the fitness function of the evolutionary process, each six-bit Gray coding has associated a 3-tuple and hence an inception neural network. In this paper, 64 six-bits encodings and 60 3-tuples are considered. Our proposal is to map the three first six-bits encodings to the first tuple of the Hamiltonian path and to map the three last six-bits encoding to the last 3-tuple (i.e. we consider that '000000', '000001' and '000011' are encodings of the tuple '(11,5,7)' and '100011', '100001' and

TABLE 1. Since 60 tuples must be encoded in binary form, at least six bits are necessary to reach $2^6 = 64$ encondings. This table shows such 64 encodings ordered according to the Gray algorithm to avoid Hamming cliffs. Let us note that for each n, the n-th and the $n+1$-th codings are at Hamming distance 1.

[’000000’, ’000001’, ’000011’, ’000010’, ’000110’, ’000111’, ’000101’, ’000100’, ’001100’,
’001101’, ’001111’, ’001110’, ’001010’, ’001011’, ’001001’, ’001000’, ’011000’, ’011001’,
’011011’, ’011010’, ’011110’, ’011111’, ’011101’, ’011100’, ’010100’, ’010101’, ’010111’,
’010110’, ’010010’, ’010011’, ’010001’, ’010000’, ’110000’, ’110001’, ’110011’, ’110010’,
’110110’, ’110111’, ’110101’, ’110100’,’111100’, ’111101’, ’111111’, ’111110’, ’111010’,
’111011’, ’111001’, ’111000’, ’101000’, ’101001’,’101011’, ’101010’, ’101110’, ’101111’,
’101101’, ’101100’, ’100100’, ’100101’, ’100111’, ’100110’, ’100010’, ’100011’, ’100001’,
’100000’]

TABLE 2. Ordering of the 60 3-tuples of kernel sizes obtained as a Hamiltonian path. Let us remark that each pair of consecutive 3-tuples are at Hamming distance 1.

[’(11, 5, 7)’,	’(11, 9, 7)’,	’(11, 9, 5)’,	’(11, 9, 3)’,	’(11, 7, 3)’,	’(11, 7, 9)’,	’(11, 7, 5)’,
’(11, 3, 5)’,	’(11, 3, 9)’,	’(11, 5, 9)’,	’(11, 5, 3)’,	’(9, 5, 3)’,	’(9, 11, 3)’,	’(9, 11, 7)’,
’(9, 11, 5)’,	’(9, 7, 5)’,	’(9, 7, 11)’,	’(9, 7, 3)’,	’(5, 7, 3)’,	’(5, 11, 3)’,	’(7, 11, 3)’,
’(7, 11, 9)’,	’(7, 11, 5)’,	’(7, 9, 5)’,	’(7, 9, 11)’,	’(7, 9, 3)’,	’(7, 5, 3)’,	’(7, 5, 11)’,
’(9, 5, 11)’,	’(9, 5, 7)’,	’(9, 3, 7)’,	’(11, 3, 7)’,	’(5, 3, 7)’,	’(5, 11, 7)’,	’(5, 11, 9)’,
’(5, 7, 9)’,	’(5, 7, 11)’,	’(5, 9, 11)’,	’(5, 9, 3)’,	’(5, 9, 7)’,	’(3, 9, 7)’,	’(3, 11, 7)’,
’(3, 11, 9)’,	’(3, 11, 5)’,	’(3, 9, 5)’,	’(3, 7, 5)’,	’(3, 7, 9)’,	’(3, 7, 11)’,	’(3, 9, 11)’,
’(3, 5, 11)’,	’(3, 5, 7)’,	’(3, 5, 9)’,	’(7, 5, 9)’,	’(7, 3, 9)’,	’(7, 3, 11)’,	’(7, 3, 5)’,
’(9, 3, 5)’,	’(9, 3, 11)’,	’(5, 3, 11)’,	’(5, 3, 9)’]			

’100000’ are encodings of the tuple ’(5,3,9)’ (see Tables 1 and 2 and Figure 2). The remaining 58 six-bit coding are bijectively mapped onto the 58 3-tuple in natural order.

Once decided the codification of the kernel sizes, the remaining hyperparameters must be also encoded. Each individual encodes all the hyperparameters needed to describe an inception neural network. Beyond the kernel sizes, the remaining hyperparameters to be encoded are the following (bX represents the branch X in the Inception-A module scheme—from left to right—and lY stands for the layer Y in the corresponding branch—from bottom to top—(see Figure 1):

- Branch 1: `b1_l1_pool_size`, with allowed values from 2 to 5 and `b1_l2_filters`, with allowed values the pairs from 32 to 256.
- Branch 2: `b2_l1_filters`, with allowed values the pairs from 32 to 256 and `b2_l1_kernel` with allowed values the odds from 3 to 11.
- Branch 3: `b3_l1_filters`, with allowed values the pairs from 32 to 256, `b3_l2_filters`, with allowed values the pairs from 32 to 256 and `b3_l2_kernel` with allowed values the odds from 3 to 11.
- Branch 4: `b4_l1_filters`, `b4_l2_filters` and `b4_l3_filters`, with allowed values the pairs from 32 to 256; and `b4_l2_kernel` and `b4_l3_kernel` with the same value from odds from 3 to 11.
- Number of inception modules concatenated: `num_inception_modules`, with allowed values `[1,2,3]`.

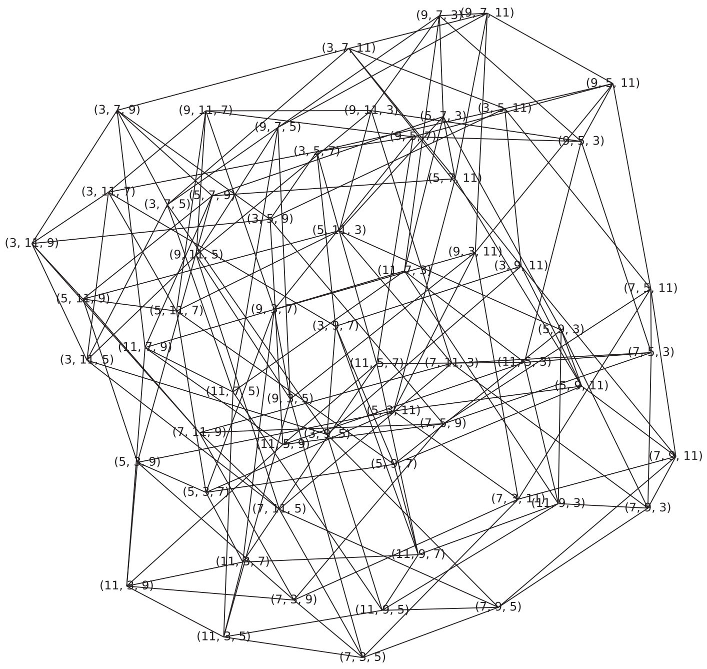

FIGURE 2. Graph with 60 nodes, where the nodes are labelled with the 3-tuples of kernel sizes (without repeated sizes). There is an edge between two nodes if and only if the corresponding 3-tuples are at Hamming distance 1. Any Hamiltonian path in this graph provides a sequence of the 60 3-tuples where two consecutive ones are at Hamming distance 1 (see Table 2).

As pointed above, each individual of the population consists on the concatenation of the Gray coding of these hyperparameters. For example, if the following hyperparameters are chosen:

b1_l1_pool_size : 3,	b1_l2_filters : 116
b2_l1_filters : 118,	b2_l1_kernel : 7
b3_l1_filters : 228,	b3_l2_filters : 210
b3_l2_kernel : 5,	b4_l1_filters : 120
b4_l2_filters : 160,	b4_l2_kernel : 3
b4_l3_filters : 184,	b4_l3_kernel : 3
num_inception_modules : 3	

then, the concatenation of the Gray codings of the sequence [3, 116, 118, 228, 210, 120, 160, 184, 37, 3] is the corresponding individual, namely

1010011101001101100101101011101110001001111000011100100110111110

Let us note that kernel sizes are not present in the sequence in an explicit way. They are encoded as the index of the 3-tuple $(7, 5, 3)$ in the Hamiltonian path considered. Particularly, the former individual is best individual produced by the evolutionary process.

In order to decode the individuals, they are split into hyperarameters and each of them is decoded by using the Gray decoding algorithm.

2.5 Statistics

In order to ascertain if the proposed forecasting methods applied to the test set improve the prediction, two different types of tests can be applied: parametric and non-parametric. The difference between both relies on the assumption that data are normally distributed for parametric tests, whereas nonexplicit conditions are assumed in non-parametric tests. For this reason, the latter is recommended when the statistical model of data is unknown [9, 10].

The Kruskal–Wallis test is a non-parametric test used to compare three or more groups of sample data. For this test, the null hypothesis assumes that the samples are from identical populations. The procedure when using multiple comparison to test whether the null hypothesis is rejected implies the use of a post-hoc test to determine which sample makes the difference. The most typical post-hoc test is the Wilcoxon signed-rank test.

The Wilcoxon signed-rank test belongs to the non-parametric category. For this test, the null hypothesis assumes that the samples are from identical populations, whereas the alternative hypothesis states that the samples come from different populations. It is a pairwise test that aims to detect significant differences between two sample means. If necessary, the Bonferroni correction can be applied to control the family-wise error rate (FWER). FWER is the cumulative error when more than one pairwise comparison (e.g. more than one Wilcoxon signed-rank test) is performed.

3 Results and Discussion

3.1 Evolving inception

In this paper, several hyperparameters of the Inception-A architecture are optimized using the PBIL algorithm. As mentioned above, every set of hyperparameters identifies a concrete neural network with this architecture.

The first layer is a simple 2D convolutional layer that plays the role of the stem module of the Inception network. The stem module refers to the first operations performed before Inception-A blocks. After the stem module, a number of Inception-A blocks are concatenated based on the hyperparameters represented by an individual. The output of the last block is flattened and connected to a *dropout-dense-dropout-dense* group of layers. The output is a dense layer with *softmax* as the activation function. In Figure 3, the network architecture proposed is detailed. The hyperparameters of the proposed network (not including Inception-A block ones) are the same for every individual evaluated (see Section 2.4).

Due to the computational intensity, during the evolution a reduced data set—composed of 10^4 examples—are used as the training set. The validation and test sets are composed of 10^4 examples for the evolutionary strategy and during the production (see next section). It must be noticed that the validation and the test sets are just the same examples in the evolutionary strategy and subsequently in the production. This avoids leakages, namely that examples could be in the training set in the evolutionary process and in the test or the validations sets in the production. During the production, the most promising model is trained with a training set of $5 \cdot 10^4$ examples.

During the evolutionary process performed by PBIL, every model (i.e. an individual) is evaluated in the validation set which is a portion of the training set. Test set is separated and reserved before the process in order to avoid data leaks. As stated before, the validation loss is used as fitness value to measure the quality of an individual, whereas the accuracy of the test set—not seen before by the evolutionary process—is used as final quality criterion (Figure 4). Each box-plot presents the accuracy of all individuals of each generation. Each individual of PBIL is a configuration for the

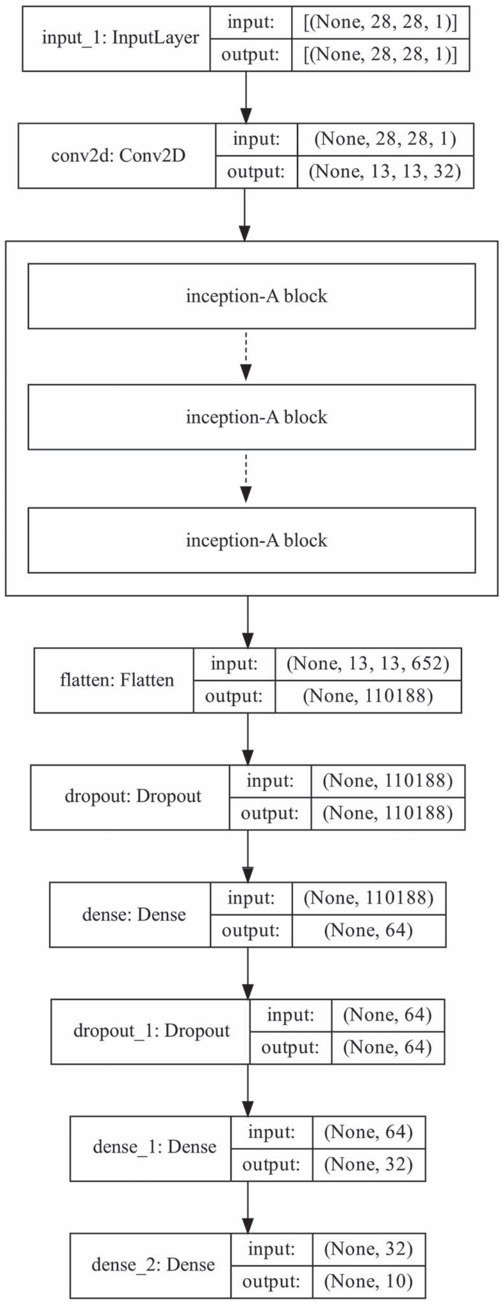

FIGURE 3. Model architecture. Stripped lines indicate that the number of blocks varies from 1 to 3.

Inception module, and the accuracy of the test set is used as the quality criterion. As it can be appreciated, the first generations contain already good individuals, while along the generations these good individuals are concentrated and new ones created.

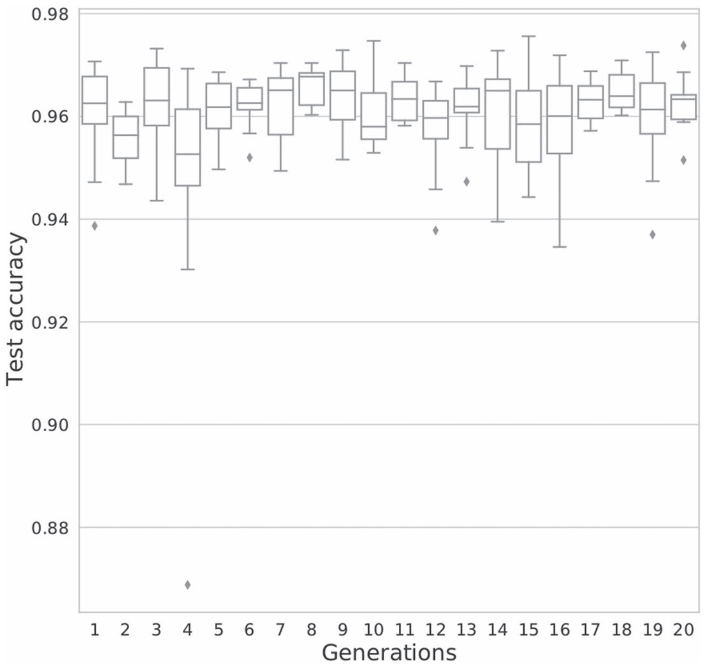

FIGURE 4. Evolution of the accuracy in test set per generation.

TABLE 3. Mean and standard deviation of accuracy for 20 independent runs on MNIST test set. The values correspond to the results of the best individuals of PBIL separated by the configuration of the number of blocks, and the execution of the overall best hyperparameters set raised from PBIL.

Hyperparameters	Accuracy
Best Inception-A 1-block individual	0.9899 ± 0.0014
Best Inception-A 2-blocks individual	0.9913 ± 0.0009
Best Inception-A 3-blocks individual	0.9920 ± 0.0011
Best hyperparameters set with early stopping	0.9922 ± 0.0008

When the evolutionary process is finished, the best individual is trained with early stopping and applying a learning rate scheduler in the same train set. In order to perform early stopping, the same validation set is used. Results presented in Table 3 are the mean accuracy and the standard deviation on the test set of 20 independent training sessions. As it can be appreciated, the best hyperparameters obtained by PBIL produce the highest accuracy, similar to those produced by original hyperparameters of Inception-A block with three concatenated blocks. This demonstrates that hyperparameters presented in [18] are already a high quality configuration for the Inception-A module. More detail about the best combination obtained is shown in tabular at the end of Section 2.4. It should be underlined that the best configuration is fully compatible with the usual Inception-A configuration, differing only in a larger number of filters.

3.2 Results comparison and statistical tests

In Table 3, the mean and standard deviation of accuracy for 20 independent runs on MNIST test set is shown. The results include 20 runs of the best hyperparameters set with early stopping; and the best results of PBIL with a number of blocks ranging from 1 to 3, and a single epoch per hyperparameter configuration.

The application of the Kruskal–Wallis test to the accuracy shown in Table 3 indicates that the differences between the medians of the best result obtained in the independent runs are significant for a confidence level of 95% ($p - value = 5 \cdot 10^{-7}$). A confidence level of 95% ($p - value$ under 0.05) is used in this analysis. This means that the differences are unlikely to have occurred by chance with a probability of 95%.

The statistical analysis using the Wilcoxon signed-rank test with Bonferroni correction of the results obtained with the 20 independent runs (Table 3) indicates that the differences between the best hyperparameter sets obtained by PBIL with early stopping and the execution during the PBIL evolution with a single epoch with configuration with 1- or 2-blocks are significant for a confidence level of 95% ($p - value$ under 0.05). The corresponding p-values are $p - value = 0.0001$ for 1-block configuration, and $p - value = 0.003$ for 2-blocks configuration. This means that the differences are unlikely to have occurred by chance with a probability of 95%. Otherwise, the comparison with 3-block configuration ($p - value = 0.35$) indicates that the differences are not significant for a confidence level of 95% ($p - value$ under 0.05).

This last comparison demonstrates that our approach with a single epoch in the evaluation of the population of PBIL does critically not penalize the performance of the CNN, at the same time that diminish the computational intensity of the evolutionary algorithm.

The mean accuracy achieved, 0.9922 ± 0.0008, with this architecture is among the best ones reported at https://paperswithcode.com/sota/image-classification-on-mnist for deep architectures in the order of 10^5 trainable parameters. Two comparisons are emphasized with the implementations reported in this repository.

- On the one hand, the one with the highest quality implementation reported on the web [4]. This implementation is a CapsNet one, with more than 1.5 million of trainable parameters and a reported accuracy of 0.9987.
- On the other hand, the one with highest quality implementation with a number of trainable parameters similar to our implementation, in the order of 10^5 trainable parameters [14]. This is also a CapsNet implementation with a reported accuracy of 0.9984.
- In comparison to our implementation, the cited implementations are trained with more than two orders of magnitudes of epochs. In our implementation, the early stopping does not progress beyond 10 epochs, with the best accuracy around 5 epochs. This underlines the low carbon footprint of our work, at the same time that achieves a competitive accuracy.

4 Conclusions

In this paper, a first attempt to optimize the architecture of the inception module has been proposed. For this purpose, PBIL as optimizer and MNIST as benchmark are used. The relevance of this task stems from the wide use of the inception module in computer vision, where it holds a large number of success cases.

Regarding the contributions of this proposal, it is aware of a low carbon footprint in the artificial intelligence area. Unacceptable deep learning architectures are no longer evaluated thanks to the codification of these architectures. This allows saving CPU cycles, and therefore, reducing their carbon footprint.

The analyses of results demonstrate that the optimized architecture of Inception-A module using only 3 blocks, and therefore, a low number of trainable parameters, achieves an excellent performance. This performance supports the use of evolutionary strategies for optimizing deep architectures while reducing the carbon footprint through the use of a single epoch during the parameters optimization. Furthermore, the best configuration is fully compatible with the usual configuration of Inception-A module.

As part of the future work, more elements of the inception architecture are intended to be handled by the evolutionary algorithm. This will allow us to propose novel architectures for this module. Other benchmarks, such as Fashion MNIST, CIFAR-100 or The Street View House Numbers are suggested for in-depth evaluation of the best hyperparameter set, and for the evolutionary algorithm proposal as a deep architecture optimizer.

Funding

M.C.M. is co-funded by the Spanish Ministry of Economy and Competitiveness for funding support through the project 'Unidad de Excelencia María de Maeztu': CIEMAT-FÍSICA DE PARTÍCULAS through the grant MDM-2015-0509. M.A.G.N. is partially supported by the Ministry of Science and Innovation through the project PID2019-107339GB-I00 'Advances in Computational Topology and Applications'.

References

[1] Estrategia Nacional de Inteligencia Artificial. *Technical Report*. Gobierno de España, 2020.

[2] S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. *Technical Report CMU-CS-94-163*. Carnegie Mellon University, Pittsburgh, PA, 1994.

[3] S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. In *Machine Learning, Proceedings of the Twelfth International Conference on Machine Learning*, A. Prieditis, S. J. Russell eds, Tahoe City, California, USA, 9–12 July 1995, pp. 38–46. Morgan Kaufmann Publishers, 1995.

[4] A. Byerly, T. Kalganova and I. Dear. A branching and merging convolutional network with homogeneous filter capsules. *CoRR*. abs/2001.09136, 2020.

[5] M. Cárdenas-Montes. Forecast daily air-pollution time series with deep learning. In *Hybrid Artificial Intelligent Systems—14th International Conference, HAIS 2019*, León, Spain, 4–6 September 2019. Vol. 11734 of Lecture Notes in Computer Science, pp. 431–443. Springer, 2019.

[6] M. Cárdenas-Montes. Uncertainty estimation in the forecasting of the ^{222}Rn radiation level time series at the Canfranc Underground Laboratory. *Logic Journal of the IGPL*, 2020 jzaa057.

[7] M. Cárdenas-Montes and I. Méndez-Jiménez. Ensemble deep learning for forecasting 222Rn radiation level at Canfranc Underground Laboratory. In *14th International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2019)*, Seville, Spain, 13–15 May 2019. Vol. 950 of Advances in Intelligent Systems and Computing, pp. 157–167. Springer, 2019.

[8] J. C. B. Gamboa. Deep learning for time-series analysis. *CoRR*. abs/1701.01887, 2017.

[9] S. García, A. Fernández, J. Luengo and F. Herrera. A study of statistical techniques and performance measures for genetics-based machine learning: Accuracy and interpretability. *Soft Computing*, **13**, 959–977, 2009.

[10] S. García, D. Molina, M. Lozano and F. Herrera. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: A case study on the CEC'2005 special session on real parameter optimization. *Journal of Heuristics*, **15**, 617–644, 2009.

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

[12] Y. LeCun. Generalization and network design strategies. *Technical Report*. University of Toronto, 1989.

[13] Y. LeCun, C. Cortes and C. J. Burges. MNIST handwritten digit database. ATT labs [online]. http://yann.lecun.com/exdb/mnist, 2, 2010.

[14] V. Mazzia, F. Salvetti and M. Chiaberge. Efficient-capsnet: Capsule network with self-attention routing. *CoRR*. abs/2101.12491, 2021.

[15] I. Méndez-Jiménez and M. Cárdenas-Montes. Modelling and forecasting of the 222 Rn radiation level time series at the Canfranc Underground Laboratory. In *Hybrid Artificial Intelligent Systems—13th International Conference, HAIS 2018*, Oviedo, Spain, 20–22 June 2018. Vol. 10870 of Lecture Notes in Computer Science, pp. 158–170. Springer, 2018.

[16] I. Méndez-Jiménez and M. Cárdenas-Montes. Time series decomposition for improving the forecasting performance of convolutional neural networks. In *Advances in Artificial Intelligence—18th Conference of the Spanish Association for Artificial Intelligence, CAEPIA 2018*, Granada, Spain. Vol. 11160 of Lecture Notes in Computer Science, pp. 87–97. Springer, 2018.

[17] E. Strubell, A. Ganesh and A. McCallum. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 3645–3650. Association for Computational Linguistics, Florence, Italy, 2019.

[18] C. Szegedy, S. Ioffe, V. Vanhoucke and A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31. AAAI Press, 2017.

[19] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich. Going deeper with convolutions, 2014.

[20] R. A. Vasco-Carofilis, M. A. Gutiérrez-Naranjo and M. Cárdenas-Montes. PBIL for optimizing hyperparameters of convolutional neural networks and STL decomposition. In *Hybrid Artificial Intelligent Systems—15th International Conference, HAIS 2020*, Gijón, Spain, 11–13 November 2020, E. A. de la Cal, J. R. V. Flecha, H. Quintián and E. Corchado, eds. Vol. 12344 of Lecture Notes in Computer Science, pp. 147–159. Springer, 2020.

[21] Z. Wang, W. Yan and T. Oates. Time series classification from scratch with deep neural networks: A strong baseline. *CoRR*. abs/1611.06455, 2016.

Received 20 February 2021