OPTIMISATION OF DEEP LEARNING HYPER-PARAMETERS USING A GENETIC ALGORITHM: A COMPARATIVE STUDY

A PREPRINT

Leonardo Capozzi
Faculdade de Engenharia
Universidade do Porto
Porto, Portugal
up201503708@fe.up.pt

Tiago Gonçalves
Faculdade de Engenharia
Universidade do Porto
Porto, Portugal
up201607753@fe.up.pt

September 4, 2023

ABSTRACT

Deep learning algorithms have been challenging human performance in several tasks. Currently, most of the methods to design the architectures of these models and to select the training hyper-parameters are still based on trial-and-error strategies. However, practitioners recognise that there is a need for tools and frameworks that can achieve high-performing models almost automatically. We addressed this challenge using a meta-heuristics approach. We implemented a genetic algorithm with a variable length chromosome with three different benchmark data sets. In this comparative study, we vary the architectures of the convolutional and the fully-connected layers and the learning rate. The best models achieve accuracy values of 98.73%, 90.81% and 54.71% on MNIST, Fashion-MNIST and CIFAR-10, respectively.

Keywords Deep learning · Meta-heuristics · Genetic algorithms · Image classification · Hyper-parameter optimisation

1 Introduction

Conventional machine learning algorithms are not able to process data in its raw form. They require careful phases of data engineering and rely on domain expertise to achieve feature extractors capable of transforming data into a suitable representation that the algorithm (*e.g.*, a classifier) may use to detect patterns in the input [1]. On the other hand, deep learning algorithms can automatically extract the representations suitable to their learning task; this process is called *representation learning* and allows deep learning algorithms to be fed with raw data [1]. Also, the increased availability of computational power and the possibility of using graphics processing unit (GPU) cards to train and deploy such algorithms in less time allowed practitioners to implement models with complex and deep architectures and to use bigger data sets to train and validate such models [2]. This context created the perfect conditions to achieve novel algorithms for different applications (e.g., image and video analysis [3], natural language understanding [4] or game playing [5]) while challenging human performance in several tasks [6]. Despite their high performances and ability to deal with raw data, these complex models require a manual selection of training parameters (commonly known as *hyper-parameters*) to converge properly. This is usually done in *trial-and-error* basis, meaning that the process may be time-consuming and tedious. To overcome this challenge, the field of automated machine learning (Auto-ML) has been created to develop tools and frameworks to achieve high-performing models almost without the need of human intervention [7, 8, 9].

Following the main ideas of [10], we performed a comparative study, using a variable-length chromosome genetic algorithm to optimise the hyper-parameters of deep learning models on three benchmark data sets (MNIST, Fashion-MNIST and CIFAR-10). The main goal of this work is to study how different data sources impact the optimal deep neural network architecture [1].

[1] Code is available at `https://github.com/TiagoFilipeSousaGoncalves/optimizing-dl-parameters-pdeec`

The remainder of the paper is organised as follows: section 2 gives an overview of the fundamentals of deep learning and genetic algorithms, necessary to understand the experimental approach; section 3 shows the methodologies and data sets that were used to address this challenge; section 4 presents the results and provides a brief discussion; and section 5 concludes the paper with the main contributions and proposes future work approaches.

2 Background

2.1 Deep Learning for Image Classification

2.1.1 Convolutional Neural Networks

To address the topic of image classification in deep learning it is mandatory to explain the concept of convolutional neural networks (CNNs). This type of deep networks has been extensively used in computer vision applications with high effectiveness [11]. A typical CNN block is composed of:

- Convolutional layers, which use kernels of different sizes and apply the convolution operation to the image to obtain feature maps. This operation can also be applied to the intermediate feature maps to generate deeper feature maps [12]. Besides accelerating the training process [13, 14], the utilisation of the convolution operation has several advantages such as the learning of features that are invariant to the location of the object or the reduced number of network parameters due to the weight sharing mechanism in the same feature map [15].
- Pooling layers, which generally follows a convolutional layer and it is used to reduce the dimensions of the feature maps and the number of network parameters [12]. Moreover, similarly to the convolutional layers, the pooling layers are also translation-invariant since their computations take neighbouring pixels into account [12]. The most common approaches are the maximum and the average pooling [16].
- Fully-connected (or dense) layers, which have the same structure of a multi-layer perceptron (MLP) [17] and require their inputs to be one-dimension (flattened) feature vectors. For this reason, these layers usually appear after the last pooling and contain most of the parameters of the CNN [12]. Also, the final output of a CNN (*e.g.*, the predicted class of an image) is given by the last fully-connected layer [18].

2.1.2 Training Strategies

The use of deep architectures allows the learning of more abstract information, however, the increase of complexity (*i.e.*, the increase in the number of parameters of the models) may also lead to over-fitting. To overcome this issue, one must employ a proper regularisation strategy [12]. Some of the most common regularisation approaches are:

- Dropout, which was introduced by [19] and consists of the deactivation of a variable number of feature detectors with a given probability, during training. This technique leads to the enhancement of the generalisation ability of the network while preventing complex co-adaptations on the training data.
- Data augmentation, which consists of the generation of new data samples without the need for extra steps of labelling [20]. For instance, in deep learning for image classification, this is done by applying transformations (*e.g.*, translations, rotations, horizontal flips, vertical flips, colour manipulations) to the images of the training set and using these transformed images during the training of the model.
- Transfer learning and fine-tuning, which is the initialisation of the model with pre-trained parameters instead of randomly set parameters. This process is also called transfer learning since it can be perceived as initialising the training phase with some prior knowledge [21]. Please note that these pre-trained parameters result from the training of state-of-the-art architecture on a benchmark data set (*e.g.*, ImageNet). Also, the training speed will increase since the network will now perform fine-tuning of the pre-trained weights to the task, instead of learning everything from scratch. A very common practice is to build image classification models using well-documented architectures (*e.g.*, VGG [22], ResNet [23]) as backbones of the models that we intend to implement to solve our specific tasks.

Please note that these methods are not mutually exclusive and can be combined to increase the robustness of the model and its performance.

2.2 Optimisation of Hyper-parameters in Deep Learning

Despite their high-performances, deep learning algorithms may present several issues related to the proper choice of the set of hyper-parameters that leads to the best performance [24]. For the sake of clarity, we consider hyper-parameters all

the variables that we can manually define before the beginning of the training of the algorithm such as, but not limited to, model architecture (*e.g.*, type of layers, type of activation functions), the loss function, the parameter optimisation algorithm (also known as the *optimiser*), the learning rate, the dropout rate and the number of epochs. There are several strategies of hyper-parameter optimisation, namely:

- Grid-search, which consists of performing a complete search over the subset of parameters. This approach may not be appropriate when we have high-dimensional search spaces due to time constraints or when we are dealing with continuous variables since it could force us to specify some boundaries [25].
- Random-search, which works similarly as grid-search with the difference that the search is performed randomly over the subset of parameters. This approach may be preferred to handle high-dimensional search spaces and it is appropriate for continuous variables [25].
- Bayesian optimisation, which consists of building a probabilistic model based on the evaluation of the function and any available prior information, and uses that model to select the subsequent configurations of hyper-parameters to evaluate [26].
- Reinforcement learning, which consists of the utilisation of learning agents to learn the best optimisation procedures under some policy constraints [27].
- Heuristics and meta-heuristics, which consists of the application of heuristics and/or meta-heuristics approaches. One of the main concerns here is a proper choice for the representation/encoding of the solution [28].

This work focus on the studying of a genetic algorithm, which falls under the scope of heuristics and meta-heuristics.

2.3 Genetic Algorithms

Inspired by the biological mechanisms and by Darwin's theory of evolution by natural selection, genetic algorithms rely on the fact that the fittest individuals (*i.e.*, solutions) will prosper and will produce offspring. Also, this strategy assumes that the best characteristics (*i.e.*, genes) are passed on the next generations. The main intuition behind this method is that we can assure progress along with the generations and be sure that the best solutions will prosper through the execution of the algorithm.

Regarding the implementation of a standard genetic algorithm, it generally works as follows:

1. We start by defining the solution representation and select a proper encoding to the genes and the chromosomes;
2. After defining the number of generations and the size of the population, we generate random solutions and evaluate them using a fitness function;
3. We select fittest individuals and apply a crossover operation to produce offspring;
4. We may also induce some random mutations to the individuals of the next generations to increase variability.

3 Methodology

3.1 Solution Representation

Equation 1 presents our solution representation which consists of an array composed by:

1. Convolutional Layers Block, which is a tensor composed of the convolutional layers that will be part of the architecture of the model. Each convolutional layer is represented as a tensor of convolutional layer's parameters: the number of convolutional filters, the size of the convolutional kernel, the type of activation function, the dropout rate and the type of pooling function.
2. Fully-Connected Layers Block, which is a tensor composed of the fully-connected layers that will be part of the architecture of the model. Each fully-connected layer is represented as a tensor of fully-connected layer's parameters: the number of output neurons, the type of activation function and the dropout rate.
3. Learning Rate: the learning rate that is used to train the model.

$$\text{Solution} = [\text{Convolutional Layers Block} \quad \text{Fully-Connected Layers Block} \quad \text{Learning Rate}] \tag{1}$$

3.2 Parameters to Optimise

Table 1 represents the type of parameters of the convolutional layers that we intended to optimise along with the values considered. Table 2 represents the type of parameters of the fully-connected layers that we intended to optimise along with the values considered. Please note that "ReLU" stands for "Rectified Linear Unit", "Tanh" stands for "hyperbolic tangent", "Max" stands for "Maximum Pooling" and "Avg" stands for "Average Pooling". Due to time constraints, we selected *adaptive moment estimation* (Adam) [29] as the only network parameter optimisation algorithm, however, with variable learning rates ($v \in \{0.001, 0.0001, 0.00001\}$).

Table 1: Names and possible values for the convolutional layers.

Name	Possible Values
Number of Convolutional Filters	$v \in \{8, 16, 32, 64, 128, 256, 512\}$
Size of the Convolutional Kernel	$v \in \{1, 3, 5, 7, 9\}$
Type of Activation Function	Identity, ReLU, Tanh
Range of Dropout Probability	$v \in [0, 1]$
Type of Pooling Function	Identity, Max, Avg

Table 2: Names and possible values for the fully-connected layers.

Name	Possible Values
Number of Output Neurons	$v \in [1, 100]$
Type of Activation Function	Identity, ReLU, Tanh
Range of Dropout Probability	$v \in [0, 1]$

3.3 Data

To perform this comparative study we use three benchmark data sets: MNIST, Fashion-MNIST and CIFAR-10. Below, we briefly present the characteristics of each data set.

3.3.1 MNIST

Proposed by LeCun *et al.* [11], the MNIST data set is composed of 28×28 grey-scale images of $70,000$ handwritten digits from 10 categories. The training set has $60,000$ images and the test set has $10,000$ images. Figure 1 shows some images taken from the data set.

3.3.2 Fashion-MNIST

Proposed by Xiao, Rasul and Vollgraf [30], the Fashion-MNIST data set was created "to serve as a direct drop-in replacement for the original MNIST data set for benchmarking machine learning algorithms, as it shares the same image size, data format and the structure of training and testing splits". This data set is composed of 28×28 grey-scale images of $70,000$ fashion products from 10 categories, with $7,000$ images per category. The training set has $60,000$ images and the test set has $10,000$ images. Figure 2 shows some images taken from the data set.

3.3.3 CIFAR-10

Collected by Krizhevsky, Nair and Hinton [31], the CIFAR-10 data set is composed of 32×32 colour images of $60,000$ variable classes from 10 categories. The training set has $50,000$ images and the test set has $10,000$ images. Figure 3 shows some images taken from the data set.

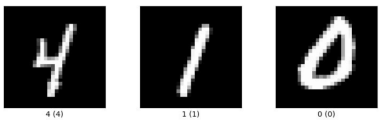


Figure 1: Images from the MNIST data set.

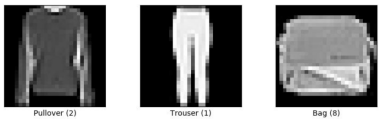


Figure 2: Images from the Fashion-MNIST data set.

Figure 3: Images from the CIFAR-10 data set.

3.4 Genetic Algorithm of Variable Chromosome Length

Based on the work of [10], we implemented an evolutionary genetic algorithm with variable chromosome length to achieve the deep learning architecture that fits better a certain data set. Concerning our work, one may identify two blocks with dynamic length property: the convolutional layers block and the fully-connected layers block. For the sake of clarity, we consider relevant to explain the adjustable parameters of the algorithm: f_{max} (maximum number of phases), g_{max} (number of generations), p_{max} (number of individuals of the population), i (initial chromosome length), e (number of train epochs), s (number of automatically selected individuals) and m (mutation rate).

We start by defining f_{max}, g_{max}, p_{max}, i, e, s and m. In the first phase f_0, first generation g_0, we generate $p = p_{max}$ random solutions with chromosome length $c_{len} = i$. We then train all the solutions on the training set for e epochs and save their accuracy values. After obtaining all these values, the solutions are sorted according to their accuracy values and the best s solutions are automatically selected for the next generation g_i. The solutions are then selected according to a given probability, dependent on their accuracy values (*i.e.*, solutions with high accuracy values have high probabilities of being selected). After the selection, the solutions go through the operation of crossover. The solutions may also suffer gene mutations at a given rate of m. This processed is repeated until $g_i = g_{max}$. In the end of the last generation, we move on to the nest phase f_i and the chromosome length is updated, $c_{len} += 1$. In the phases $f_i > f_0$, the solutions are initialised with the weights of the best solution of the previous phase, through a transfer learning operation. The main idea here is to assure that the initialisation of the solution of the next phases is not random. This way we assure that the solutions have already some prior knowledge. One of the main reasons for this procedure is to assure that e will not have a significant impact on the training of the solutions with longer chromosomes. Even if the solutions have different architectures, we transfer the maximum number of parameters from the reference solution. Since the search space size of the solutions increases with length of the chromosome (see Figure 4), we also add an

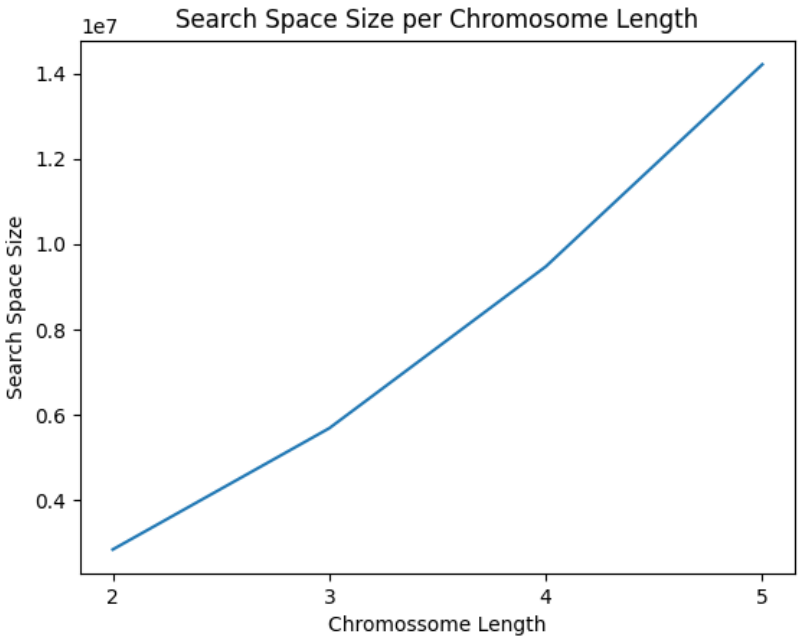


Figure 4: Variation of the solution's search space size with the increase of the length of the chromosome.

early-stopping parameter to the algorithm which stops the search if the fitness of the best model achieved so far does not increase from one phase to the next one. The complete structure of the algorithm is presented below.

Algorithm 1: Evolutionary Genetic Algorithm with Variable Chromosome Length.

Result: The fittest solution.
Initialise input shape;
Initialise number of labels;
Initialise $f_{max} = 5$;
Initialise $g_{max} = 100$;
Initialise $p_{max} = 40$;
Initialise $i = 2$;
Initialise $e = 5$;
Initialise $s = 4$;
Initialise $m = 0.2$;
Initialise $f = 0$;
while $f < f_{max}$ **do**
  $g = 0$;
  while $g < g_{max}$ **do**
    if $g = 0$ **then** Generate $p = p_{max}$ random solutions;
    else Select the s best solutions automatically;
    Select the remaining $p = p_{max} - s$ solutions, according to a given probability;
    Perform the crossover operation;
    Perform the mutation operation at the rate m;
    for p in range of p_{max} **do**
    if $f > 0$ **then** Perform the transfer learning operation;
    Train the solution for e epochs;
    Compute fitness of the all solutions (*i.e.*, accuracy);
    $g+ = 1$
  end
  $f+ = 1$
end

Table 3: Accuracy (%) and loss values obtained with the best model for the test set of MNIST, Fashion-MNIST and CIFAR-10.

Data set	Accuracy (%)	Loss
MNIST	98.73	0.01878
Fashion-MNIST	90.81	0.11558
CIFAR-10	54.71	1.46837

When the algorithm reaches convergence, we train the fittest solution for 30 epochs on the training set and test the model on the test set of each data set. Section 4 presents the results per data set.

3.4.1 Fitness Function

To calculate the best individuals of a given generation we need to compute a fitness value for each individual. The initial fitness function that was implemented can be written as:

$$\mathcal{F}(s) = (1/loss(s)) * accuracy(s) \tag{2}$$

where s is a solution, $loss(s)$ is the loss value of solution s and $accuracy(s)$ is the classification accuracy of solution s. Maximising this function leads to solutions that have a large accuracy and a low loss value. One of the downsides of this function is that small variations in the loss can lead to very large changes in the overall value of the fitness, which often resulted in the selection of individuals that did not have the best accuracy but had a low loss. To overcome this issue we used the following fitness function:

$$\mathcal{F}(s) = accuracy(s) \tag{3}$$

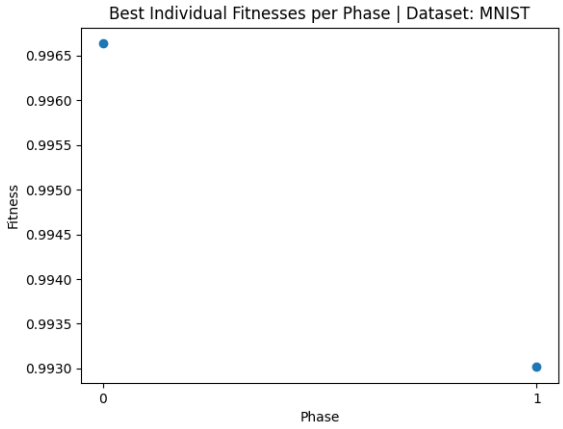
which does not include the loss value and focuses only on the accuracy of the model. This is the fitness function used in this work.

4 Results and Discussion

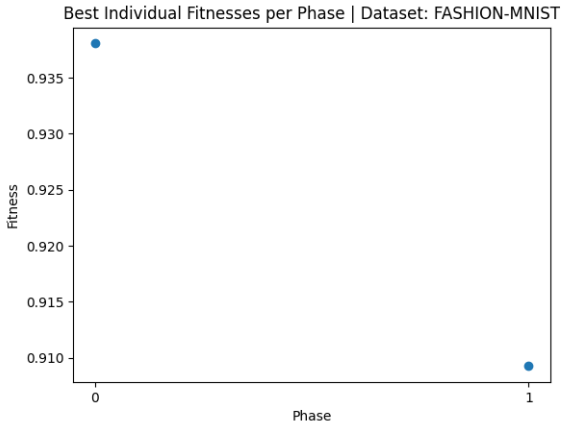
Figure 5 and Figure 6 present the best individual fitnesses per phase and the best individual fitnesses per generation per phase, respectively. The results show that, in all data sets, that from one phase to the next one (*i.e.*, when we increase by the size of the chromosome), the solutions seem to lose quality. This becomes clearer when we observe Figure 7 and Figure 8, which show the overall distribution of the fitnesses of the generated solutions and the distribution of the fitnesses of the generated solutions in each phase, respectively. While in phase 0 most of the solutions seem to have fitnesses that are closer to the best, in phase 1, the solutions are almost uniformly distributed by all fitnesses. Intuitively, increasing the size of the chromosome from one phase to another (*i.e.*, increasing the complexity of the model) would mean that the model would over-fit easier. However, we believe that this did not happen due to two possible causes: our transfer learning procedure is not benefiting the model in terms of the initialisation of its weights or the number of training epochs (e) is not enough.

Table 3 presents the accuracy and loss values achieved in all data sets. Since the complexity of the data set increases from MNIST to CIFAR-10, the results are coherent, since the best (*i.e.*, higher accuracy and lower loss) are achieved in the MNIST data set, followed by Fashion-MNIST and CIFAR-10.

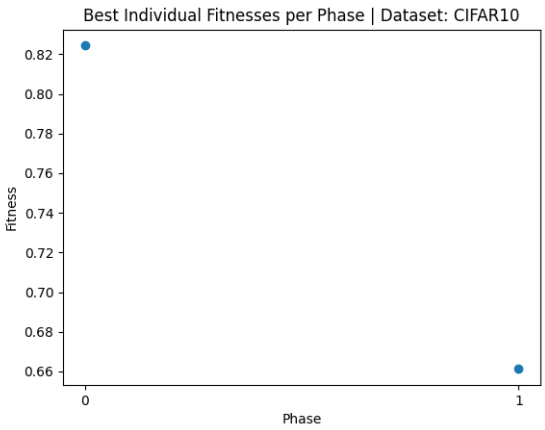
The results also suggest that the initialisation of the genetic algorithm depends on the data set. Therefore, this approach requires proper tuning of the parameters concerning the data in which we want to optimise the algorithm.



(a) MNIST

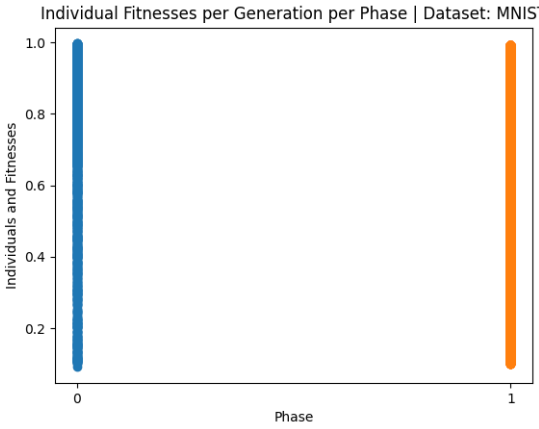


(b) Fashion-MNIST

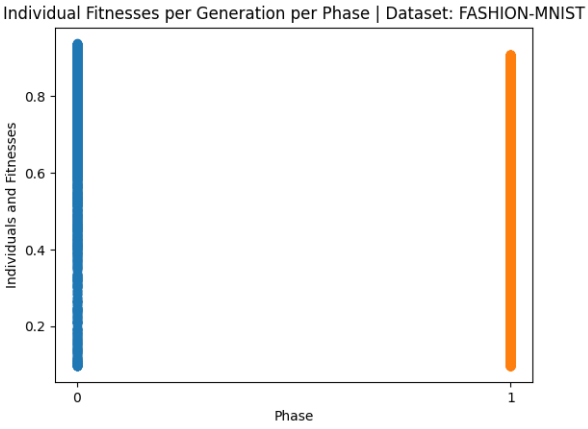


(c) CIFAR-10

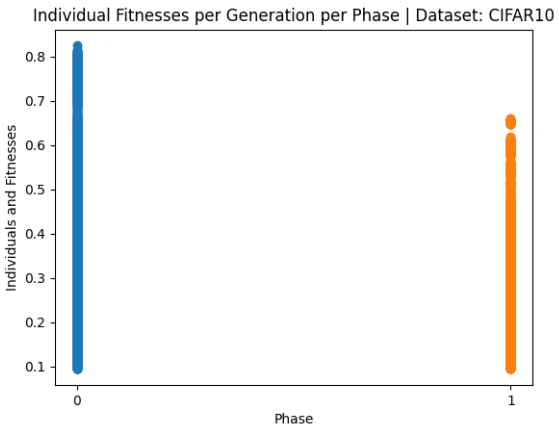
Figure 5: Best individual fitnesses per phase: a) MNIST, b) Fashion-MNIST and c) CIFAR-10 data sets.



(a) MNIST

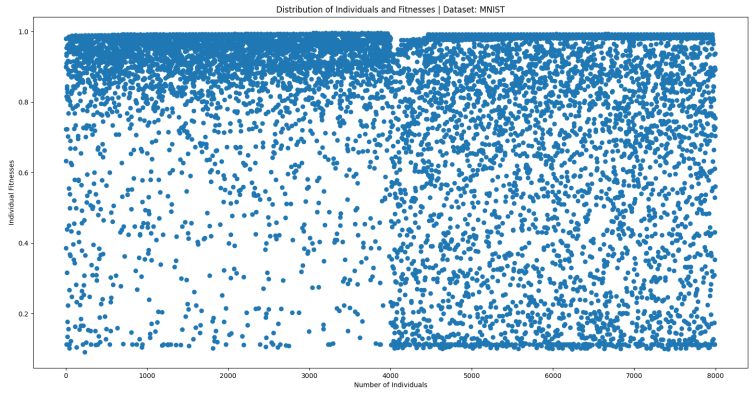


(b) Fashion-MNIST

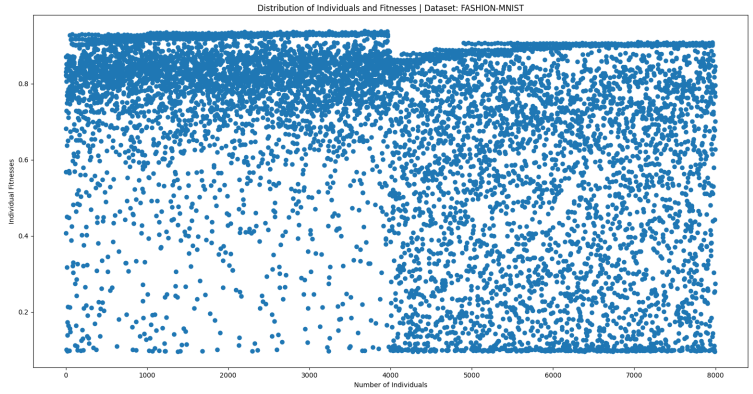


(c) CIFAR-10

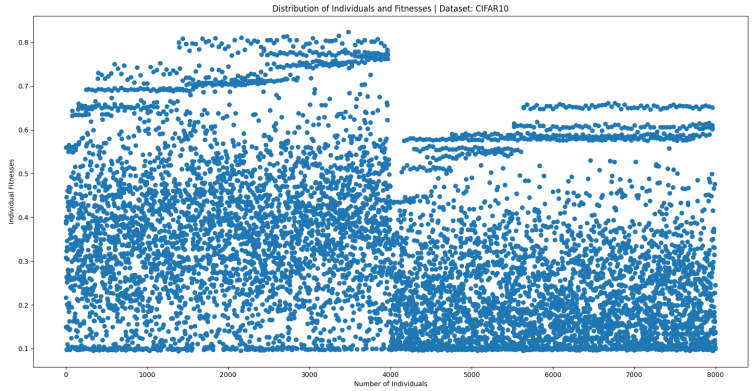
Figure 6: Individual fitnesses per generation per phase: a) MNIST, b) Fashion-MNIST and c) CIFAR-10 data sets.



(a) MNIST

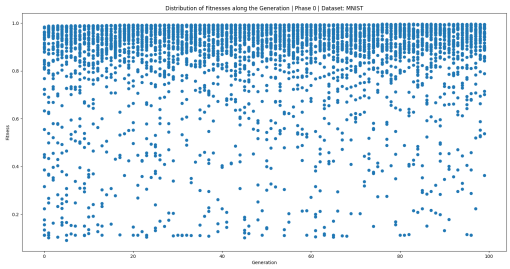


(b) Fashion-MNIST

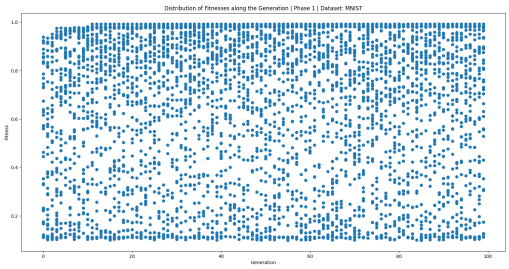


(c) CIFAR-10

Figure 7: Distribution of the fitnesses of all generated solutions: a) MNIST, b) Fashion-MNIST and c) CIFAR-10 data sets.

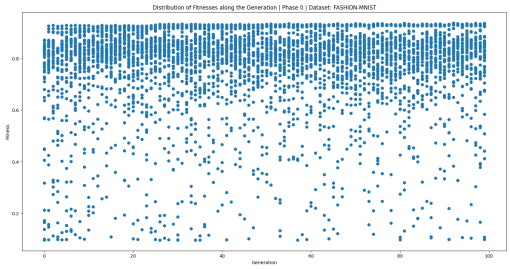


(a.1) Phase 0

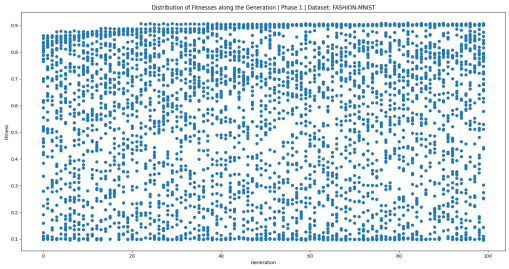


(a.2) Phase 1

(a) MNIST

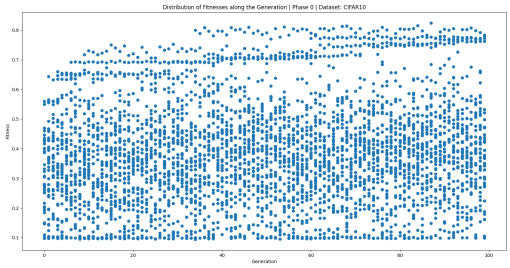


(b.1) Phase 0

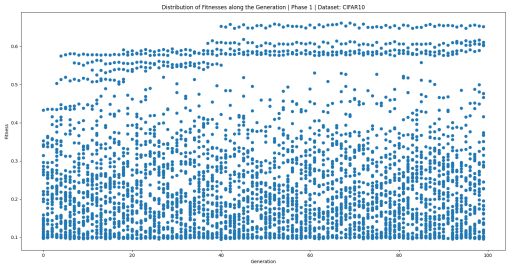


(b.2) Phase 1

(b) Fashion-MNIST



(c.1) Phase 0



(c.2) Phase 1

(c) CIFAR-10

Figure 8: Distribution of the fitnesses of the solutions per generations per phases: a) MNIST, b) Fashion-MNIST and c) CIFAR-10 data sets.

5 Conclusions and Future Work

Based on the approach of [10], we implemented a genetic algorithm with variable chromosome length to achieve deep learning architectures that fit better certain data sets. This comparative study used 3 benchmark data sets (MNIST, Fashion-MNIST and CIFAR-10). To the authors' knowledge, this is the first study that uses this algorithmic approach and performs a comparison between these 3 data sources. The results obtained are in accordance to what has been reported in the literature and are coherent, *i.e.*, using the same parameters to initialise the algorithm, thus, it makes sense that the best results are obtained with the simplest data set, which, in this work, is MNIST.

Further work should be devoted to an ablation study in which we remove the transfer learning procedures to evaluate the impact of this operation in the quality of the generated solutions and to add a variable number of epochs and different algorithms of optimisation as hyper-parameters to achieve better training and test results. Besides, it would also be interesting to study new approaches in terms of fitness functions (*e.g.*, include the training epochs (e)).

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[3] H. Pham, Q. Xie, Z. Dai, and Q. V. Le, "Meta pseudo labels," *arXiv preprint arXiv:2003.10580*, 2020.

[4] S. Ravi and Z. Kozareva, "Self-governing neural networks for on-device short text classification," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 887–893, 2018.

[5] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[6] S. M. McKinney, M. Sieniek, V. Godbole, J. Godwin, N. Antropova, H. Ashrafian, T. Back, M. Chesus, G. C. Corrado, A. Darzi, *et al.*, "International evaluation of an ai system for breast cancer screening," *Nature*, vol. 577, no. 7788, pp. 89–94, 2020.

[7] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2423–2432, 2018.

[8] H. Cai, J. Yang, W. Zhang, S. Han, and Y. Yu, "Path-level network transformation for efficient architecture search," in *International Conference on Machine Learning*, pp. 678–687, PMLR, 2018.

[9] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[10] X. Xiao, M. Yan, S. Basodi, C. Ji, and Y. Pan, "Efficient hyperparameter optimization in deep learning using a variable length genetic algorithm," *arXiv preprint arXiv:2006.12703*, 2020.

[11] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[12] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, and M. S. Lew, "Deep learning for visual understanding: A review," *Neurocomputing*, vol. 187, pp. 27–48, 2016.

[13] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[14] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Is object localization for free?-weakly-supervised learning with convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 685–694, 2015.

[15] M. D. Zeiler, *Hierarchical convolutional deep learning in computer vision*. PhD thesis, New York University, 2013.

[16] Y.-L. Boureau, J. Ponce, and Y. LeCun, "A theoretical analysis of feature pooling in visual recognition," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 111–118, 2010.

[17] C. Van Der Malsburg, "Frank rosenblatt: Principles of neurodynamics: Perceptrons and the theory of brain mechanisms," in *Brain theory*, pp. 245–248, Springer, 1986.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[19] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[20] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, "Autoaugment: Learning augmentation strategies from data," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 113–123, 2019.

[21] D. Erhan, A. Courville, Y. Bengio, and P. Vincent, "Why does unsupervised pre-training help deep learning?," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 201–208, JMLR Workshop and Conference Proceedings, 2010.

[22] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[24] J.-Y. Kim and S.-B. Cho, "Evolutionary optimization of hyperparameters in deep learning models," in *2019 IEEE Congress on Evolutionary Computation (CEC)*, pp. 831–837, IEEE, 2019.

[25] P. Liashchynskyi and P. Liashchynskyi, "Grid search, random search, genetic algorithm: A big comparison for nas," *arXiv preprint arXiv:1912.06059*, 2019.

[26] K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. Hoos, and K. Leyton-Brown, "Towards an empirical foundation for assessing bayesian optimization of hyperparameters," in *NIPS workshop on Bayesian Optimization in Theory and Practice*, vol. 10, p. 3, 2013.

[27] K. Li and J. Malik, "Learning to optimize," *arXiv preprint arXiv:1606.01885*, 2016.

[28] L. A. Passos, D. R. Rodrigues, and J. P. Papa, "Fine tuning deep boltzmann machines through meta-heuristic approaches," in *2018 IEEE 12th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, pp. 000419–000424, Ieee, 2018.

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[30] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[31] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

Supplementary Material

In this section, we show the results obtained with the first fitness function (Equation 2), during 4 phases. Each phase had 50 generations and each generation had 40 individuals. Table 4 presents the accuracy and loss values achieved in all data sets.

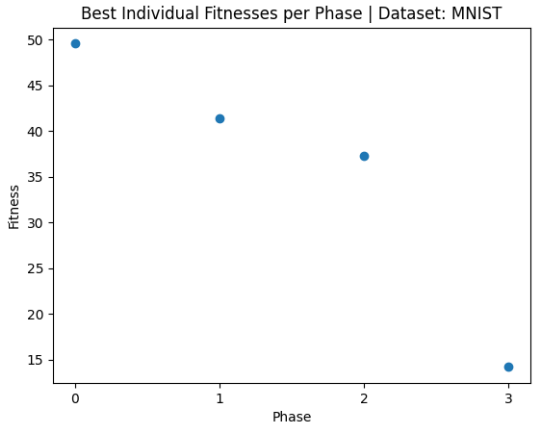
Figure 9 shows the best individual fitnesses per phase of all data sets, Figure 10 shows all the individual fitnesses per generation per phase of all data sets and Figure 11 presents the distribution of the fitnesses of all generated solutions of all data sets.

Figure 12, Figure 13, Figure 14 show the distribution of the fitnesses of the solutions per generations per phases of the MNIST, Fashion-MNIST and CIFAR-10 data sets respectively.

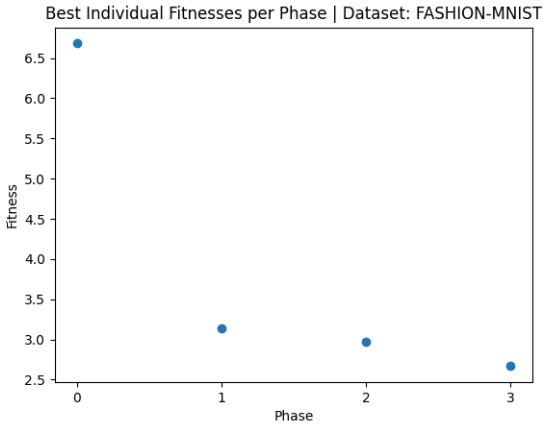
It is interesting to observe that, even with this preliminary test with a different fitness function, the best solutions were found in phase 0. Moreover, please note that the quality of the best solution does not improve in posterior phases. Once again, this means that the number of training epochs may be important as a hyper-parameter of the solution itself or, at least, should be taken into account in a new fitness function.

Table 4: Accuracy (%) and loss values obtained with the best model for the test set of MNIST, Fashion-MNIST and CIFAR-10.

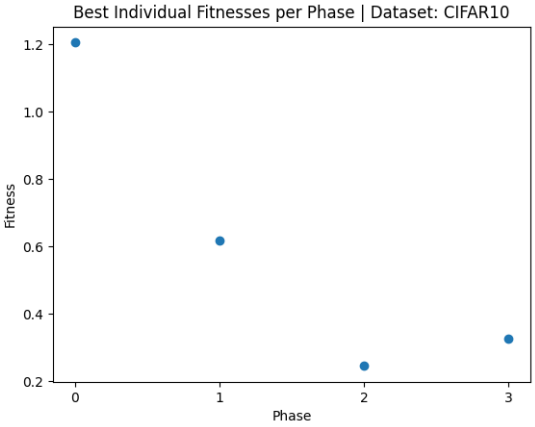
Data set	**Accuracy (%)**	**Loss**
MNIST	98.87	0.01813
Fashion-MNIST	88.76	0.38088
CIFAR-10	58.81	0.52339



(a) MNIST

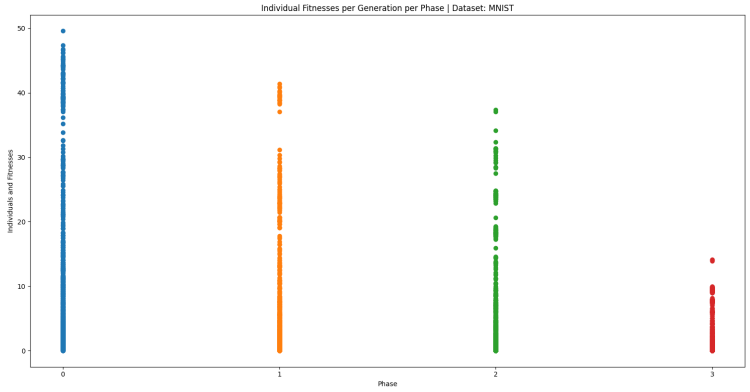


(b) Fashion-MNIST

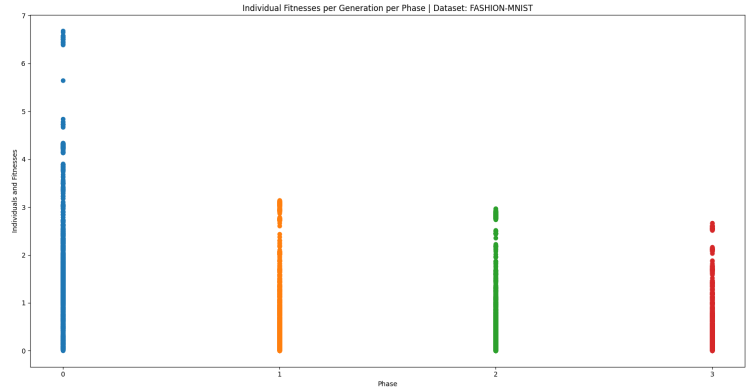


(c) CIFAR-10

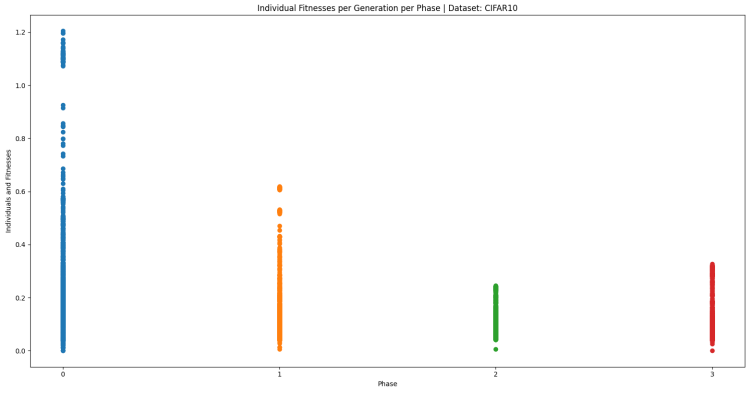
Figure 9: Best individual fitnesses per phase: a) MNIST, b) Fashion-MNIST and c) CIFAR-10 data sets.



(a) MNIST

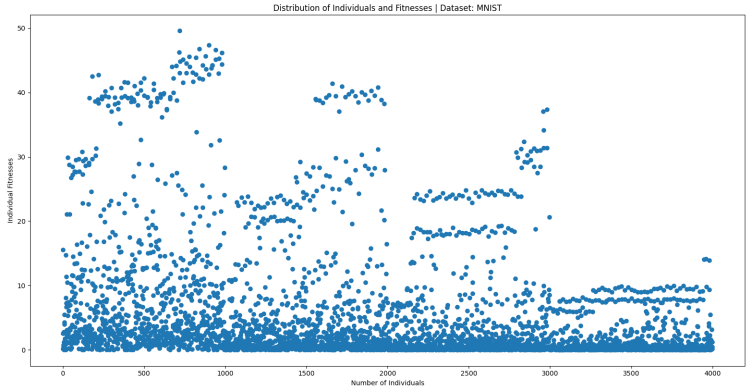


(b) Fashion-MNIST

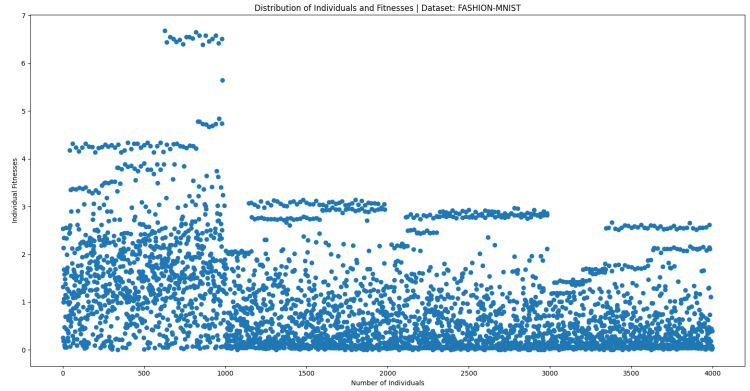


(c) CIFAR-10

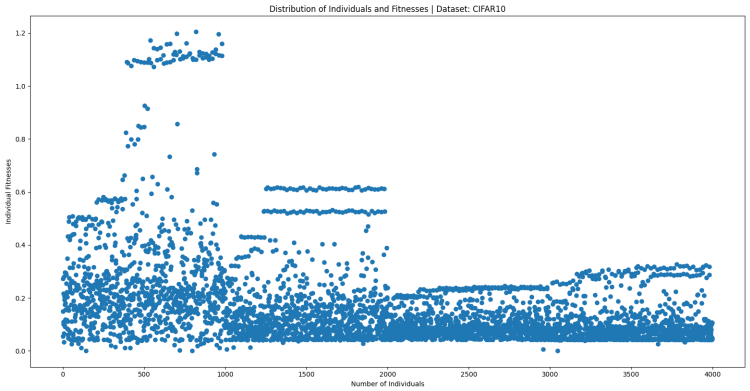
Figure 10: Individual fitnesses per generation per phase: a) MNIST, b) Fashion-MNIST and c) CIFAR-10 data sets.



(a) MNIST

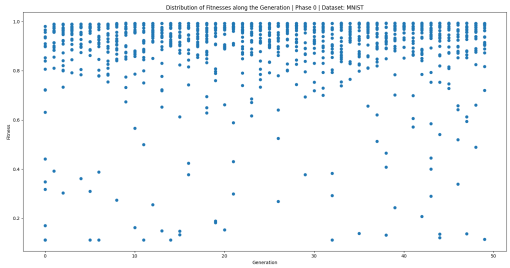


(b) Fashion-MNIST

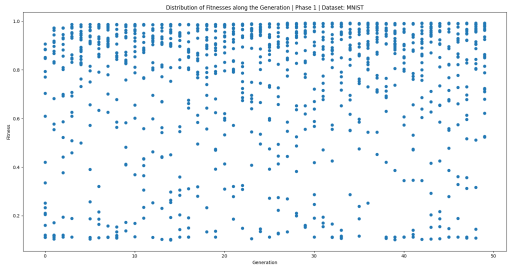


(c) CIFAR-10

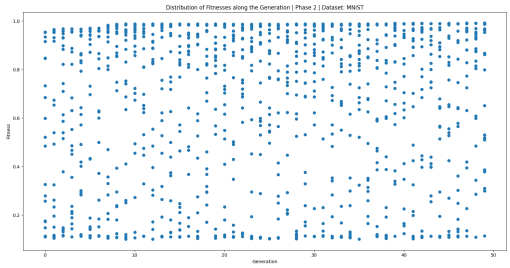
Figure 11: Distribution of the fitnesses of all generated solutions: a) MNIST, b) Fashion-MNIST and c) CIFAR-10 data sets.



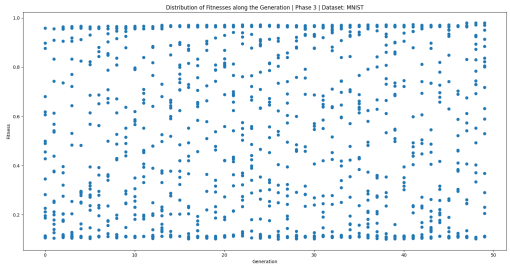
(a) Phase 0



(b) Phase 1

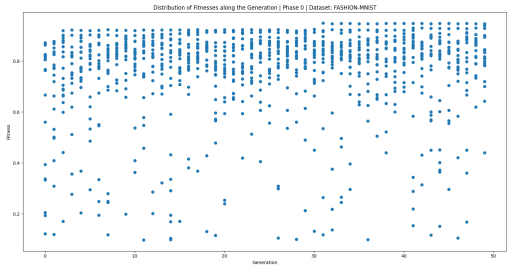


(c) Phase 2

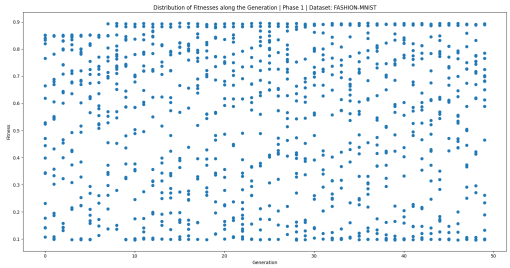


(d) Phase 3

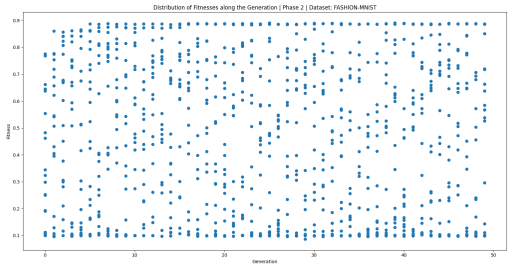
Figure 12: Distribution of the fitnesses of the solutions per generations per phases on MNIST data set.



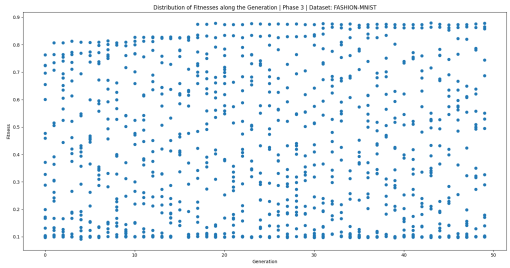
(a) Phase 0



(b) Phase 1

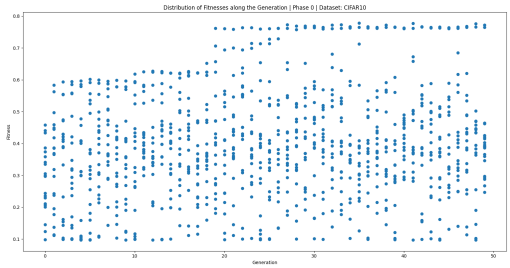


(c) Phase 2

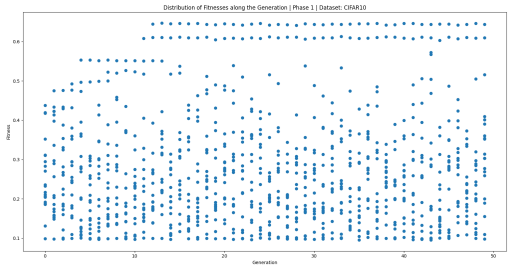


(d) Phase 3

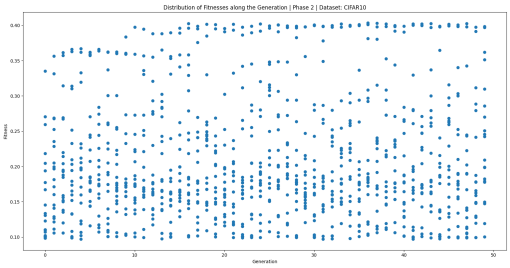
Figure 13: Distribution of the fitnesses of the solutions per generations per phases on Fashion-MNIST data set.



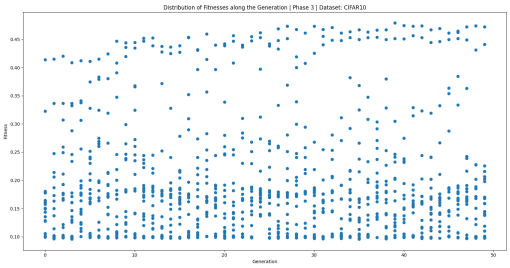
(a) Phase 0



(b) Phase 1



(c) Phase 2



(d) Phase 3

Figure 14: Distribution of the fitnesses of the solutions per generations per phases on CIFAR-10 data set.