Received February 10, 2020, accepted March 6, 2020, date of publication March 16, 2020, date of current version March 25, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2981141

Hyper-Parameter Selection in Convolutional Neural Networks Using Microcanonical Optimization Algorithm

AYLA GÜLCÜ, (Member, IEEE), AND ZEKI KUŞ
Department of Computer Science, Fatih Sultan Mehmet University, 34445 Istanbul, Turkey
Corresponding author: Ayla Gülcü (agulcu@fsm.edu.tr)

ABSTRACT The success of Convolutional Neural Networks is highly dependent on the selected architecture and the hyper-parameters. The need for the automatic design of the networks is especially important for complex architectures where the parameter space is so large that trying all possible combinations is computationally infeasible. In this study, Microcanonical Optimization algorithm which is a variant of Simulated Annealing method is used for hyper-parameter optimization and architecture selection for Convolutional Neural Networks. To the best of our knowledge, our study provides a first attempt at applying Microcanonical Optimization for this task. The networks generated by the proposed method is compared to the networks generated by Simulated Annealing method in terms of both accuracy and size using six widely-used image recognition datasets. Moreover, a performance comparison using Tree Parzen Estimator which is a Bayesion optimization-based approach is also presented. It is shown that the proposed method is able to achieve competitive classification results with the state-of-the-art architectures. When the size of the networks is also taken into account, one can see that the networks generated by Microcanonical Optimization method contain far less parameters than the state-of-the-art architectures. Therefore, the proposed method can be preferred for automatically tuning the networks especially in situations where fast training is as important as the accuracy.

INDEX TERMS Convolutional neural networks, hyper-parameter optimization, microcanonical optimization, tree Parzen estimator.

I. INTRODUCTION

Convolutional Neural Networks (CNN) belong to a special class of artificial neural networks where convolution operation is used instead of matrix multiplication in at least one of their layers [1]–[3]. The effectiveness of CNNs for object recognition was demonstrated in ILSVRC (IMAGENET large scale visual recognition challenge) [4] after which it was used to address various other visual recognition tasks such as face recognition, facial expression recognition and action recognition. However, the success of the CNNs is highly dependent on the selection of the hyper-parameters. Determining which hyper-parameters to tune and defining value domains for those hyper-parameters, and then selecting the best set of values require meticulous design and experiment processes which can only be conducted by the participation of an expert from the domain. The need for the automatic design of CNNs is especially important for complex CNN architectures where the parameter space is so large that trying all possible combinations is computationally infeasible. Random Search (RS) and Grid Search (GS) are among the most widely-used non-adaptive methods used for tuning the hyper-parameters of small CNNs [5], [44], [53]. They are called non-adaptive in the sense that they do not change the course of the search by considering any results that are already available. GS trains the CNN for every combination of hyper-parameter values in the predefined scale. It is shown in [52] that GS suffer from the curse of dimensionality because the number of these combinations grows exponentially with the number of hyper-parameters. Moreover, the allocation of too many trials to the exploration of dimensions that are not important may result in poor coverage in important dimensions. In [53], the hyper-parameters of the neural networks are optimized by GS. Then,

The associate editor coordinating the review of this manuscript and approving it for publication was Farid Boussaid.

Bergstra and Bengio [5] perform similar experiments using RS by taking these experiments as a point of comparison, and the results suggest that RS finds better models than GS in most cases and requires less computational time. The performance of both of these methods can be improved by making them somewhat adaptive by the manual updates provided by an expert. Bayesian Optimization based approaches [44], [54], [55] and meta-heuristics based approaches are among the adaptive methods in which the solution space is refined by the information obtained in earlier stages. Despite the implementational simplicity of Bayesian Optimization methods, most of the current research focuses on meta-heuristics based approaches, especially on evolutionary algorithms due to efficient coverage of the search space. The field of application of evolutionary computation to evolve some aspects of neural networks is called *neuroevolution* which arose by the end of 1980s. EPNet [6] and NEAT [7] are among the studies with huge impact in the field. Although neuroevolution is a well-established research area, recent improvements in computational resources, in particular in the availability of graphics processing units (GPUs) enabled optimization of complex networks more efficiently.

In the study proposed by Sun *et al.* [8] evolutionary approach which is called EvoCNN is used to automatically generate CNN architectures and the initial weights. The network model produced by the proposed approach achieved competitive results against more complex architectures. Ma *et al.* [9] propose another study based on the evolutionary algorithms which is tested on well-known datasets and achieved successful results against state-of-the-art architectures. Baldominos *et al.* [33] use genetic algorithms (GA) and grammatical evolution for the automatic generation of CNN architectures.

Particle Swarm Optimization (PSO) is among the meta-heuristics that has been successfully applied for the optimization of CNN hyper-parameters. In [10] PSO-based method is tested using well-known datasets and competitive results with state-of-the-art architectures are obtained. It is also shown in [11] that the PSO algorithm is a strong competitor to the state-of-the-art algorithms in terms of the classification error.

Differential Evolution (DE) and Harmonic Search (HS) are among the other meta-heuristics that have been used for optimizing CNN hyper-parameters (see [12], [13]). In addition, reinforcement learning (RL) [14], [15] method is also being used for finding the optimal architecture and hyper-parameters of CNNs (see [16], [17]).

In this study, Microcanonical Optimization (μO) algorithm which is a variant of Simulated Annealing (SA) algorithm is used for the hyper-parameter optimization in CNNs. To the best of our knowledge, this is the first study in which μO is used for optimizing CNN architecture and hyper-parameters. There is only one study proposed by Ayumi *et al.* [26] related to using μO in CNNs, but the proposed approach is used for optimizing the weights of a simple CNN rather than optimizing the network's architecture and hyper-parameters. In some of the meta-heuristics-based studies mentioned above, some

TABLE 1. Brief comparison of the hyper-parameters optimized in our study with the related studies. The studies are compared according to the criteria including whether the proposed approach uses variable number of layers (Dyn.), and whether it optimizes Convolution layer (Conv.), Pooling layer (Pool.) and Fully Connected Layer (FC) hyper-parameters, whether it optimizes Regularization (Reg.), Learning Process (LR.) and the way the initial weights (W) are defined.

Ref.	Dyn.	Conv.	Pool.	FC	Reg.	W	LR.
Dufourq et al. [28]	●	●	●	●	●		●
Bochinski et al. [29]	●	●		●			
Fujino et al. [30]		●	●	●			●
Rincon et al. [31]		●	●				
Ma et al. [9]	●	●	●	●	●		●
Assunçao et al. [32]	●	●	●	●	●		●
Baldominos et al. [33]	●	●	●	●	●		●
Yamasaki et al. [19]		●	●				
Lorenzo et al. [34]		●	●				
Sun et al. [10]	●	●	●		●		
Silva et al. [35]		●	●	●	●		●
Lorenzo et al. [36]		●	●				
Wang et al. [11]	●	●	●	●			
Wang et al. [12]	●	●	●	●			
Lee et al. [13]		●	●				
Bengio et al. [5]	●	●	●	●	●	●	●
Domhan et al. [37]	●	●	●	●	●	●	●
Neary et al. [17]	●	●		●			
Baker et al. [16]	●	●	●	●			
van Stein et al. [38]	●	●			●		●
Proposed Work	●	●	●	●	●		●

hyper-parameters are excluded from the optimization process. A fixed CNN architecture where the number of layers is kept constant is adopted, and only some of the hyper-parameters within a fixed architecture is optimized. In this study, an architecture that is allowed to extend or shrink dynamically is adopted and a wide range of hyper-parameters are selected for optimization. The hyper-parameters considered for optimization in this study and in other studies are summarized in Table 1. One of our motivation for selecting μO is to demonstrate the ability of this method to reach acceptable solutions within a small number of iterations. This is especially important for training CNNs where the computational cost of a single iteration can be very high. Another advantage of this method is that there are only a few parameters that need to be tuned. The networks generated by the μO method is compared to the networks generated by SA method in terms of both accuracy and size using six widely-known image recognition datasets. Additionally, a performance comparison using Tree Parzen Estimator (TPE) which is a Bayesion optimization-based approach is also presented. The amount of time spent on training a CNN depend very much on the size of that CNN which is defined as the total number of weights and biases. In both μO and SA, the size is considered as the second objective whereas the accuracy is the first objective. Whenever two network models generated by the same algorithm yield in the same accuracy, the one with smaller size is selected. The CNN architectures whose hyper-parameters are optimized by μO are able to achieve competitive classification results when compared to the state-of-the-art architectures. When the architectures generated by μO algorithm are compared to the state-of-the-art architectures in terms of the size of those models, it is observed that the CNNs produced by the proposed method by conforming

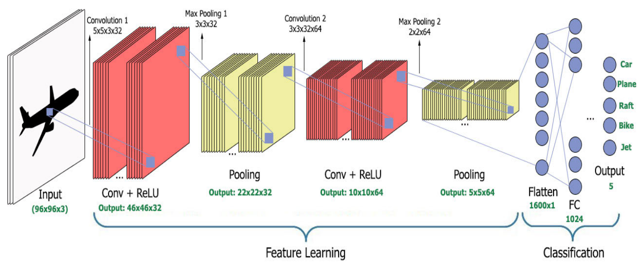


FIGURE 1. An example CNN architecture.

to the predefined constraints and design rules contain far less parameters than the state-of-the-art architectures.

II. CONVOLUTIONAL NEURAL NETWORKS

In this section, we aim to remind some key concepts of CNNs for better understanding the paper rather than to present an exhaustive study about the CNNs (for more detailed information refer to [3]). CNNs are deep neural networks in which convolution operation (navigating filters over inputs) is applied in at least one of the layers with the aim of extracting useful features.

Architecture of a simple convolutional neural network (CNN) consists of the following layers: Input Layer, Convolution Layer, Pooling Layer, Fully Connected Layer and Output Layer. The images are fed into the CNN through the input layer. Then, in the convolution layer, different features of the input (feature maps) are extracted by using filters (kernels). The number of feature maps, the size of the filters, and other parameters such as stride and padding define the structure of a convolutional layer. Moreover, the number of convolution layers changes from one CNN to another. For example, the CNN architecture shown in Fig.1 (elements are not to scale) consists of 2 convolution layers each of which having different number of filters with different sizes. In the figure, images of size $96 \times 96 \times 3$ are fed into the CNN through the input layer, where the first two dimensions denote the width and height of the image, and the third dimension denotes the number of channels which is 3 for a colored image. The first convolution layer includes 32 filters each with a size of 46×46. Following the convolution operation, linear structure is transformed into a non-linear structure with *ReLU* activation function. As shown in the figure, the size of the filters are reduced after the the pooling operation. These features are then transformed into a one-dimensional vector which is then feeded to the fully connected layer (FC) where the classification result is obtained. The error value is calculated using the outputs, and the weights are updated based on the error.

In the pooling layer, the size of the feature maps are reduced according to selected hyper-parameters such as the filter size, the stride and the pooling method. The stride hyper-parameter, which is common to the convolution and pooling layer, determines the number of steps the filter takes as it moves through the input. In the pooling method, the highest or the average value (depending on the selected pooling method) in the corresponding field of the filter is produced. As an alternative to the pooling method, Springenberg *et al.* [18] propose Strive method in which a convolution layer with 3×3 or 2×2 filter sizes with a stride of 2 is used. Filters in this layer do not have weights to learn. Only size reduction is applied and the summary information is transferred to other layers.

TABLE 2. Hyper-parameters that can be optimized for CNN.

Convolution	Pooling	Fully Connected
Stride	Filter Size	Number of Neurons
Padding	Stride	Number of Layer
Filter Size	Pooling Method	
Number of Layer	Number of Layer	
Number of Feature Map		

Learning Process	Regularization	Weights
Batch Size	Dropout Rate	Weight Decay
Learning Rate	Regularization Method	Weight Initializer
Activation Function	L1-L2 Coefficient	Weight Multiplier
Optimizer	Bias Initializer	Weight Normalization

The hyper-parameters that can be set for each of the layers are summarized in Table 2. The hyper-parameters that are common to all layers like the learning rate, dropout rate and weight initializer are given in separate columns.

III. SOLUTION REPRESENTATION

Solutions can be represented either by block structure as in [9] or by layer structure as in [8], [10], [19]. In this study, an architecture that is allowed to extend or shrink dynamically is adopted, and the block structure is selected for this purpose. Taking the VGGNet [20] architecture as a reference, each convolution layer (CONV), activation function (ACT), batch normalization (BN), pooling / strive layer (SUBS) and dropout (DROP) parameters are collected under a single block which is called the convolution block. (see Fig. 2a).

Fully connected layer, activation function, batch normalization and the dropout are collected in a single block which is called the fully connected block (see Fig.2b). Solutions are formed by combining a number of convolution and fully connected blocks as shown in Fig.2c according to the following pattern: $[(CONV - ACT - BN) * N - (SUBS - DROP)] * M - [(FC - ACT - BN - DROP)] * K$, where N, M and K takes on values in $\{2, 3\}$, $\{2, 4\}$ and $\{0, 2\}$, respectively. There cannot be more than 3 convolution layers in a convolution block. If the current convolution block has 2 convolution layers already, a new convolution block is added with a probability of 0.5. Likewise, if there are 3 convolution layers in the present convolution block, it is allow to delete a convolution layer with a probability of 0.5. Each solution can consist of at least 2, and at most 4 convolution blocks. Adding a new convolution block can be performed with a very low probability (probability of 0.0625). It is also allowed for a solution to have zero or at most 2 fully connected blocks. For the solutions that do not contain fully connected blocks, dropout with a rate of 0.5 is applied after the last convolution block in order to improve the generalization performance.

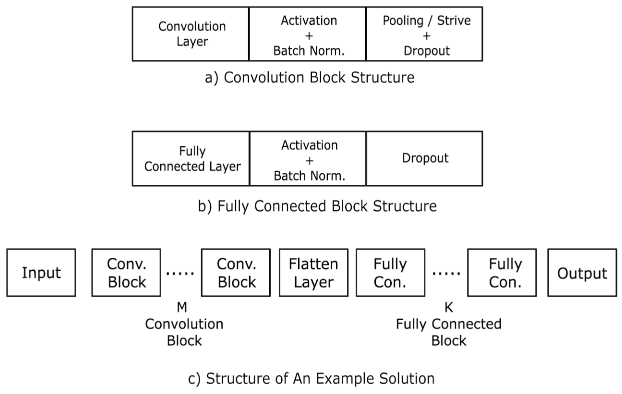


FIGURE 2. Convolution block, fully connected block and a complete solution.

TABLE 3. Hyper-parameters and associated value ranges adopted in this study.

Layer	Hyper-parameters	Ranges of Values
Convolution	Filter Size	3, 5, 7
	Number of Feature Map	32, 64, 96, 128, 160, 192, 224, 256
Pooling Layer	Filter Size	2, 3
	Pooling Type	MAX, AVG
Strive	Filter Size	2, 3
Fully Connected	Number of Neurons	128, 256, 512
Learning Process	Activation Function	ReLU, Leaky-ReLU, Elu
Regularization	Dropout Rate	0.3, 0.4, 0.5

The activation function and the pooling or strive methods are selected at the beginning and these parameters remain the same for all convolution blocks.

The hyper-parameters considered for optimization in this study and value ranges defined for each of those parameters are shown in Table 3. When no restriction is applied on the selected parameter values, complex architectures with huge number of parameters leading high training times are generated. Random selection of these parameter values may also lead to inefficient networks. For example, if the hyper-parameters such as the filter size and stride are assigned very large values at the beginning, then the input size is reduced prematurely causing the loss of information. The models begin to move away from some patterns observed in the state-of-the-art architectures such as selecting the number of neurons to be associated with dropout rate, increasing the number of feature maps gradually and reducing the size of the filter gradually. Following the general rules discovered in the previous studies, but at the same time adding some flexibility to be able to explore the search space efficiently, we aim to obtain smaller models (models with fewer parameters) that achieve competitive results with larger models.

Hyper-parameter constraints considered during the generation of a new solution are listed below (the studies recommending those values are given in brackets):

1) In order to prevent premature feature map size reduction, stride hyper-parameter is fixed to 1 for the convolutional layer, and it is fixed to 2 for the pooling and strive layers.
2) The convolution layers within the same convolution block will use the same hyper-parameter values ([18], [20]).
3) Padding hyper-parameter is selected to "SAME" for all convolution and pooling layers.
4) The filter size selected for a convolution layer should be less than or equal to the filter size in the previous convolution block.
5) The dropout value selected for a pooling layer should be greater than or equal to the dropout value in the pooling layer in the previous convolution block. Dropout can be at most 0.5 ([21]–[24]). For the first convolution layer, dropout value will be selected as 0.2 ([18], [21], [25]).
6) The number of feature maps in a convolutional block should be at least 32 more than the number of feature maps in the previous convolutional block ([18], [20]).
7) The number of neurons in a fully connected layer can be set equal or twice the number of neurons in the previous fully connected layer.
8) The dropout rate selected for a fully connected layer should be greater than or equal to the dropout value selected in the previous fully connected layer ([20]–[22]). For the first fully connected layer, a dropout rate of 0.3 is used.

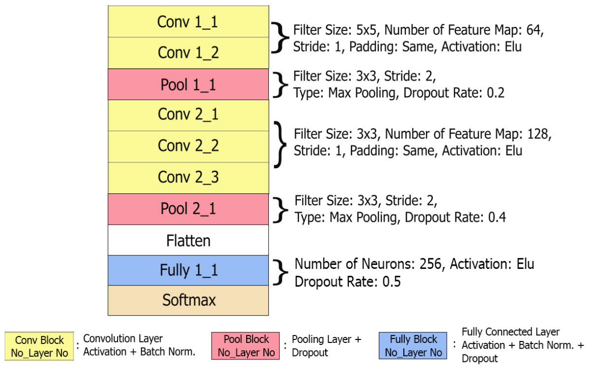

Conv 1_1
Conv 1_2
Filter Size: 5x5, Number of Feature Map: 64, Stride: 1, Padding: Same, Activation: Elu
Pool 1_1
Filter Size: 3x3, Stride: 2, Type: Max Pooling, Dropout Rate: 0.2
Conv 2_1
Conv 2_2
Conv 2_3
Filter Size: 3x3, Number of Feature Map: 128, Stride: 1, Padding: Same, Activation: Elu
Pool 2_1
Filter Size: 3x3, Stride: 2, Type: Max Pooling, Dropout Rate: 0.4
Flatten
Fully 1_1
Number of Neurons: 256, Activation: Elu Dropout Rate: 0.5
Softmax
Conv Block No_Layer No : Convolution Layer Activation + Batch Norm.
Pool Block No_Layer No : Pooling Layer + Dropout
Fully Block No_Layer No : Fully Connected Layer Activation + Batch Norm. + Dropout

FIGURE 3. Structure of an initial solution.

An initial solution formed considering the constraints related to the network architecture and the hyper-parameters is shown in Fig.3. This solution consists of 2 convolution blocks, where the first convolution block consists of 2 convolution layers each with 64 filters, whereas the second block consists of 3 convolution layers each with 128 filters. Each block is ended with a max pooling layer, and there is only one fully connected layer. The same activation function, *elu* is used in all of the layers.

IV. OPTIMIZATION ALGORITHMS

In this study, Microcanonical Optimization (μO) algorithm is used for architecture selection and hyper-parameter optimization in CNNs. The networks generated by the proposed method is compared to the networks generated by another meta-heuristics, namely Simulated Annealing, in terms of both accuracy and size. In addition to these two methods, Tree Parzen Estimator (TPE) which is a Bayesion optimization-based approach is also used for comparison. These three algorithms are explained in sections IV-A to IV-C.

A. SIMULATED ANNEALING

Simulated Annealing (SA) is a well-established meta-heuristic which uses an adaptation of the Metropolis algorithm to accept non-improving moves based on a probability [27]. If a random move improves the current objective function, then it is accepted. Worsening moves are accepted according to an exponential probability distribution that is biased by the amount of worsening in the objective function and a temperature parameter, T_{cur}. Initially, the system is started with a high initial temperature which is called T_0 and the temperature is lowered gradually according to a prescribed annealing schedule. The search is stopped when the temperature reaches a minimum temperature value, T_f, that is, when no solution that increases the cost function is accepted anymore. Pseudo-code presenting the general structure of the SA method is given in Algorithm 1.

Algorithm 1 Simulated Annealing

```
Generate $S$ ;
initialize_params ( $T_f$, $cr$, $nbr\_iter$)
compute_params( $T_0$, $nbr\_out\_iter$, $nbr\_in\_iter$ )
$T_{cur} \leftarrow T_0$
for $out\_counter \leftarrow 0$ to $nbr\_out\_iter$ do
    for $in\_counter \leftarrow 0$ to $nbr\_in\_iter$ do
        Generate $S^{'}$
        $\Delta F = |F(S^{'})| - |F(S)|$
        $p_{acc} \longleftarrow \min\{1, \exp(-\frac{\Delta F}{T_{cur}})\}$
        if $accept(p_{acc}, p_{random})$ then
            $S \leftarrow S^{'}$
        end
    end
    $T_{cur} \leftarrow T_{cur} * cr$
end
```

B. MICROCANONICAL OPTIMIZATION ALGORITHM

Microcanonical Annealing (MA) algorithm which is a variant of SA uses Creutz's technique [39] to control the system by its internal energy, not by its temperature. In the Creutz's algorithm, total energy of the system is conserved. To accomplish this, total energy is defined as the sum of the potential energy, $E_P(S)$, which is the actual objective function value of the current solution S and the kinetic energy, E_D, (also called the *demon*) which is the variable amount of energy (see Eq.1) that is always non-negative.

$$E(S) = E_P(S) + E_D \tag{1}$$

The algorithm starts with a random state which is presumably of high energy, so initial E_D is usually set to zero. As shown in Algorithm 2, after the system reaches equilibrium at a particular energy level, then the energy of the system is reduced by a specific amount. Barnard [40] suggests considering the rate of accepted moves to states of higher energy as a reasonable test for equilibrium. He also takes the amount of energy to be removed from the system after it reaches equilibrium as the initial demon, $E_{D0}/300$. MA algorithm has several advantages over SA algorithm. First, MA is based on Creutz algorithm which does not require the generation of high quality random numbers, or heavy computations. Second, the experiments on the Creutz method show that, it can be programmed to run an order of magnitude faster than the conventional Metropolis method for discrete systems [41].

Algorithm 2 Microcanonical Annealing

```
Initialize $E_D$;
while termination condition is not satisfied do
    repeat
        Generate S';
        $\triangle E \leftarrow$ E(S') - E(S);
        if $\triangle E \leq 0$ then
            $S \leftarrow S'$;
            $E_D \leftarrow E_D - \triangle E$;
        else
            if $E_D - \triangle E \geq 0$ then
                $S \leftarrow S'$;
                $E_D \leftarrow E_D - \triangle E$;
    until system reaches equilibrium;
    Reduce $E_D$;
end
```

In the MA method, a new state, S', is randomly selected and is accepted or rejected based on the change in energy which is calculated as $\Delta E = E(S') - E(S)$. If $\Delta E \leq 0$, then S' is accepted and kinetic energy of the system, E_D, is increased in order to conserve total energy. If $\Delta E > 0$, the acceptance of S' is contingent upon E_D, if $\Delta E \leq E_D$ the change is accepted and demon energy is decreased, otherwise S' is rejected. MA is controlled by periodically removing energy from the system. After a number of iterations performed at a particular energy level, the energy of the system is reduced by removing a fixed amount from demon. As shown in Algorithm 2, the procedure terminates if the number of iterations is reached or the best solution remains constant for a specified number of iterations.

In this study, we employed a slightly modified version of this algorithm defined in [42]. The algorithm consists of two procedures that are alternately applied: *initialization* and *sampling* (see Algorithm 3). In the initialization phase, a greedy local search is employed. When the algorithm gets stuck at a local minima, then the sampling phase starts. This phase tries to free the solution from the local minima by resorting to Creutz's algorithm. The authors, Torreão and Roe [42] state that their method does not resort to annealing, so they call this method *Microcanonical Optimization* (μO) algorithm, and we adopt the same name.

Algorithm 3 Microcanonical Optimization (μO)

```
while termination condition is not satisfied do
    Initalization();
    Sampling();
end
return $S_{Best}$
```

Initialization Phase: During this phase, μO performs a greedy local search (LS) in which only the moves leading to a state of lower cost are accepted. As a termination criterion, Linhares and Torreão [43] suggest using the number of moves rejected in a row for interrupting this phase. A maximum iteration limit for this phase may also be set. The initialization steps are shown in Algorithm 4.

Algorithm 4 Initialization Procedure

```
Input: S, max_rejected, max_init_iter
$\mathcal{L} \leftarrow \{\}$;  // list of rejected solutions' $\triangle E$
nbr_rej ← 0;
nbr_iter ← 0;
while (nbr_rej < max_rejected & nbr_iter< max_init_iter) do
    nbr_iter ← nbr_iter + 1;
    Generate S';
    △E ← E(S') - E(S);
    if $\triangle E \geq 0$ then
        Add △E to $\mathcal{L}$ ;
        nbr_rej ← n + 1;
    else
        nbr_rej ← 0;
        S ← S';
end
```

Sampling Phase: This phase tries to free the solution from the local minima by resorting to the Creutz's algorithm. For a given fixed energy level (see Eq.1), a sample of solutions are generated, but the acceptance of these solution is dependent on E_D. All improving moves are accepted, and the worsening moves are accepted only if *demon* is capable of absorbing the cost difference incurred. Linhares and Torreão [43] suggest using an adaptive strategy to define E_D, by observing the rejected moves during the initialization phase. Rejected moves are put into a list denoted by $\mathcal{L}$, then the cost jumps associated with these moves are used to define initial demon, D_I, that will be used during the sampling phase. Initially, E_D is set to the value of D_I. The steps of this algorithm are shown in Algorithm 5.

One cycle of the algorithm is illustrated in Table 4. In the example, *max_rejected* and *max_init_iter* takes the values of 5 and 9, respectively. That is, the initialization phase stops if 5 moves in a row are rejected, or if the number of iterations exceeds 9. In addition, adaptive selection of E_D in the sampling phase is also illustrated in the table. The median of the rejected objective values is selected as E_D, and the moves yielding a cost increase greater than this value are rejected.

Algorithm 5 Sampling Procedure

```
Select $D_I$ from the list-of-rejected-moves;
Let max_samp_iter be the maximum number of sampling iterations;
Let $S$ be the starting solution of the sampling phase;
num_iter ← 0;
$E_D \leftarrow D_I$;
while (num_iter < max_samp_iter) do
    Generate S';
    △E ← E(S') - E(S);
    if $\triangle E \leq 0$ then
        $S \leftarrow S'$;
        $E_D \leftarrow E_D$ - △E;
    else
        if $E_D - \triangle E \geq 0$ then
            $S \leftarrow S'$;
            $E_D \leftarrow E_D$ - △E;
    num_iter ← num_iter + 1;
end
```

TABLE 4. Illustration of a cycle in μO.

Cycle 1		
Initialization phase		
$\triangle E$	accept	$\mathcal{L}$
0.0157	No	0.0157
0.011	No	0.0157, 0.011
-0.0037	Yes	0.0157, 0.011
-0.0008	Yes	0.0157, 0.011
0.006	No	0.0157, 0.011, 0.006
0.0065	No	0.0157, 0.011, 0.006, 0.0065
0.0067	No	0.0157, 0.011, 0.006, 0.0065, 0.0067
0.0097	No	0.0157, 0.011, 0.006, 0.0065, 0.0067, 0.0097
0.0209	No	0.0157, 0.011, 0.006, 0.0065, 0.0067, 0.0097, 0.0209
	Stop	median= 0.0097
Sampling phase		E_D= 0.0097
$\triangle E$	$\triangle E < E_D$	
0.011	No	
	Stop	

C. TREE-STRUCTURED PARZEN ESTIMATOR

Bayesian optimization approaches keep track of past evaluation results to focus on the most promising hyper-parameters in contrast to random or grid search where significant amount of time can be spent evaluating bad hyper-parameters. Sequential Model Based Optimization (SMBO) algorithms are a formalization of Bayesian optimization where each trial is executed one after another. The true objective function that is costly to evaluate is approximated by a probability model (surrogate) that is cheaper to evaluate. The steps of a generic SMBO are given in Algorithm 6. At each step, the point x^* that gives the best result for the surrogate is selected for calculating the true objective function f. Then this new point along with its true objective function value is added to the observation history which is then used to update the current surrogate model M_t. There are several variants of the SMBO algorithms that differ in terms of building the surrogate of the objective function and the criteria for selecting the next set of hyper-parameters. In this study, we consider SMBO with Tree Parzen Estimators [44], [45] as the surrogate model and the Expected Improvement (EI) as the selection criterion.

Algorithm 6 The Pseudo-Code of Generic Sequential Model Based Optimization

```
Input: $f, M_0, T, S$
$\mathcal{H} \leftarrow \varnothing$ ;                // observation history
for $t \leftarrow 0$ to $T$ do
    $x^* \leftarrow argmin_x\ S(x, M_{t-1})$
    evaluate $f(x^*)$ ;          // true objective
    function calculation
    $\mathcal{H} \leftarrow \mathcal{H} \cup (x^*, f(x^*))$ ;          // update $\mathcal{H}$
    fit a new model $M_t$ to $\mathcal{H}$ ;          // update
    surrogate model $M_t$
end
return $\mathcal{H}$
```

Expected Improvement is the expectation under some model M(surrogate) of f that $f(x)$ will be smaller (better) than some threshold y^* which is computed as in (2). $p_M(y|x)$ is the surrogate probability model expressing the probability of y given x. If $p_M(y|x)$ is zero for every point that $y < y^*$, then the hyper-parameters x are not expected to give any improvement. Conversely, if $EI_{y^*}(x)$ is positive, then the hyper-parameters x are expected to yield smaller values than y^*. TPE build the model by applying Bayes rule, that is, $p_M(y|x)$ is computed as $p_M(y|x) = p(x|y)p(y)/p(x)$. Then the expected information equation becomes as shown in (3).

$$EI_{y^*}(x) = \int_{-\infty}^{y^*} max(y^* - y, 0)p_M(y|x)dy \tag{2}$$

$$EI_{y^*}(x) = \int_{-\infty}^{y^*} max(y^* - y, 0)\frac{p(x|y)p(y)}{p(x)}dy \tag{3}$$

The probability of the hyper-parameters x given the score y, $p(x|y)$, is defined using two different distributions, namely $l(x)$ and $g(x)$, where $l(x)$ is the distribution formed using the observations better than y^*, whereas $g(x)$ is the distribution of the remaining ones (see (4)).

$$p(x|y) = \begin{cases} l(x), & \text{If } y < y^* \\ g(x), & \text{If } y \geq y^* \end{cases} \tag{4}$$

Intuitively, TPE wants to draw values of x which are more likely under $l(x)$ than under $g(x)$ and the parameter y^* is chosen to be some quantile γ of the observed y values, so that $p(y < y^*) = \gamma$. The candidates are selected according to $l(x)$ and evaluated according to $l(x)/g(x)$ and at each iteration, the x^* which gives the greatest EI which is computed as in (5) is returned.

$$EI_{y^*}(x) = \frac{\gamma y^* l(x) - l(x)\int_{-\infty}^{y^*} p(y)dy}{\gamma l(x) + (1-\gamma)g(x)} \alpha \left(\gamma + \frac{g(x)}{l(x)}(1-\gamma)\right)^{-1} \tag{5}$$

Prior to the run of TPE, random search is applied in order to collect data for building the initial surrogate, M_0. The number of random search iterations depends only on the computation budget allocated for this step of the algorithm. The more computational budget the better initial probability distribution. In this study, the number of initial random search iterations is selected as the 10% of the total number of solutions to be produced. The ratio of best observations, γ is selected as 0.2. At each step, 1000 candidate solutions whose approximate objective function values of are calculated using the surrogate probability model are generated. For the SMBO with TPE implementation, we use Hyperopt python library [46].

V. RESULTS

A. EXPERIMENTAL SETUP

μO algorithm and SA was implemented in Python, and the CNNs were trained using Keras [47] with a Tensorflow backend on the following hardware: Intel Xeon CPU @ 2.30GHz CPU, 12.6 GB Memory, 16 GB Tesla T4 GPU.

Both μO and SA algorithms start with the same initial solution presented in Section III and each of the algorithms iterates by moving from one solution to the next by conforming to the architecture and hyper-parameter constraints. Evaluating the objective function of a solution (network) obtained at any iteration involves a complete training of the network using the whole training data and for a number of epochs and then testing it using the whole test data. However, performing these evaluations at each iteration is computationally inefficient. In order to overcome this drawback, we used a function approximation approach that is widely accepted in CNN hyper-parameter optimization studies. A given network is trained for a small number of epochs, like 5, using a reduced sample of the original data, although providing poor results. This pessimistic estimation of the network quality will not effect the evolution of the good networks, since what is important here is to perform a fair comparison between different networks rather than their actual objective function values. From the original training data, 50% of them are selected randomly for training, and 10% of this reduced sample of training data is used for validating a given network. The original test set is never used during training and it is only used for evaluating the actual performance of the best solution obtained for a whole run of the algorithm. The image classification datasets used in this study are given in Table 5. The number of training and test samples in the original dataset and in the reduced dataset is also given in the table. For each dataset, among the models produced by the algorithm, top 5 CNNs according to the accuracy are trained for long training epochs (200 epochs) using the original training data and tested on the original test data.

B. PARAMETER TUNING

For the SA algorithm, the selection of T_0, T_f and the associated cooling scheme is very important for the performance. There are many studies from different domains in which the effect of these parameters have been carefully observed. Thompson and Dowsland [56] tested different

TABLE 5. Image datasets used in the experiments.

	original dataset		reduced dataset	
Dataset	#train	#test	#train	#valid
MNIST [2]	60.000	10.000	27.000	3.000
CIFAR10 [48]	50.000	10.000	22.500	2.500
FashionMNIST [49]	60.000	10.000	27.000	3.000
EMNIST-Digits [50]	240.000	40.000	108.000	12.000
EMNIST-Letters [50]	88.800	14.800	39.960	4.440
EMNIST-Balanced [50]	112.800	18.800	50.760	5.640

cooling schedules using different *cr* values, and the results indicated that the best cooling strategy was to employ slow geometric cooling with $cr = 0.99$. They set T_0 at a value where it was estimated that approximately 75% of moves would be accepted. In a more recent study [57], it is shown that *cr* in the geometric cooling scheme is not significant as long as its value is close to 1. Smith *et al.* also proposed a real time initial temperature selection strategy [58]. Initial temperature is adjusted so that non-improving solutions will be accepted with a probability of 50%. In order to estimate the energy levels of the solutions, a short "burn-in" period is used and then the initial temperature is set equal to the average positive change of energy divided by $ln(2)$.

In this study, we adopted a real time initial temperature selection strategy that employs a short burn-in period in which both worsening and improving moves are accepted. We set $T_0 = \frac{-1 * avg\delta P}{\ln(p_{acc})}$, where $avg\delta P$ is the average worsening energy observed during the burn-in period, and p_{acc} is the initial acceptance probability. For the p_{acc}, we selected a value of 0.5 which ensures that non-improving moves will be accepted with a probability of 50%, and for *cr*, we adopted the most agreed upon value of 0.99. There are several ways to determine the termination condition. For example, SA might terminate when when T_{cur} reaches a predefined final temperature T_f, or after a number of iterations has passed. Additionally, a time limit can also be set on the algorithm. In this study, we set this condition as the number of solutions created during a single run of SA, and the algorithm stops if the total number of created solutions exceeds the solution budget. This approach ensures that approximately the same number of solutions are created at each run of the algorithm and it allows a fair comparison between different algorithms. The solution budget or the total number of iterations, *tot_nbr_iter*, is defined as 200 solutions. In order to define the number of outer iterations, *nbr_out_iter*, in which the current temperature is cooled we tested for two different values, 10, 20, and the number of solutions sampled at each temperature level, *nbr_in_iter*, is defined as $nbr\_in\_iter = \frac{tot\_nbr\_iter}{nbr\_out\_iter}$.

There are a few parameters within the μO algorithm that might affect the performance of the algorithm. These parameters are *max_init_iter* and *max_samp_iter*, where the first parameter determines the maximum number of iterations in the initialization phase, whereas the second parameter determines the maximum number of iterations in the sampling phase. Also, the parameter *max_rejected_moves* which defines the number of rejected solutions in a row that ends the initialization phase should be defined. As the algorithm alternates between the initialization and the sampling process, a termination condition on the number of these cycles is also needed. We set this condition as the number of solutions created during the μO algorithm run and the algorithm stops if the total number of created solutions exceeds the solution budget. This approach ensures that approximately the same number of solutions are created at each run of the algorithm. Within the solution budget which is defined as 200 solutions, the parameters *max_init_iter* and *max_samp_iter* are defined as the ratio of the solutions created in each cycle. The algorithm is tested using two different values for the minimum number of cycles, *min_cycle*, and the ratio of the solutions in the initialization phase, *init_soln_ratio*, and the ratio of the solutions in the sampling phase, *samp_soln_ratio*, are selected as given in Table 6

TABLE 6. Test configurations for μO.

Configs	*min_cycle*	*init_soln_ratio*	*samp_soln_ratio*
Cfg 1	20	0.7	0.3
Cfg 2		0.8	0.2
Cfg 3		0.9	0.1
Cfg 4	10	0.7	0.3
Cfg 5		0.8	0.2
Cfg 6		0.9	0.1

For the configurations where the *min_cycle* is selected as 10, the maximum number of solutions generated at each cycle is computed as 200/10 = 20. For the Config 4 where *init_soln_ratio* is selected as 0.7, the maximum number of solutions generated in the initialization phase is set to $20 * 0.7 = 14$, and the remaining solution budget of 6 solutions can be used in the sampling phase. The *max_rejected_moves* parameter is set to be the half of the *max_init_iter* value.

The μO method was run for 6 configurations defined in Table 6 using the CIFAR10 and FashionMNIST datasets. The solutions accepted throughout the run of the algorithm are recorded and used for performance comparison. Kruskal-Wallis h-test is applied to determine if there are significant differences among those configurations. The test results reveal that Config 4 is significantly better than 1, 2, 5 and 6, but there is no significant difference between Config 3 and 4. In order to further identify the best configuration, top 5 solutions obtained for Config 3 and 4 were trained over 200 training periods using the same datasets. T-test results again revealed no significant differences, therefore Config 3 which had better average accuracy is chosen as the winning configuration.

C. PERFORMANCE COMPARISON

The μO algorithm is run independently for each of the six image recognition datasets starting from the initial solution given in Section III. The algorithm iterates by moving from one solution to the next by conforming to the architecture and hyper-parameter constraints within the predefined solution

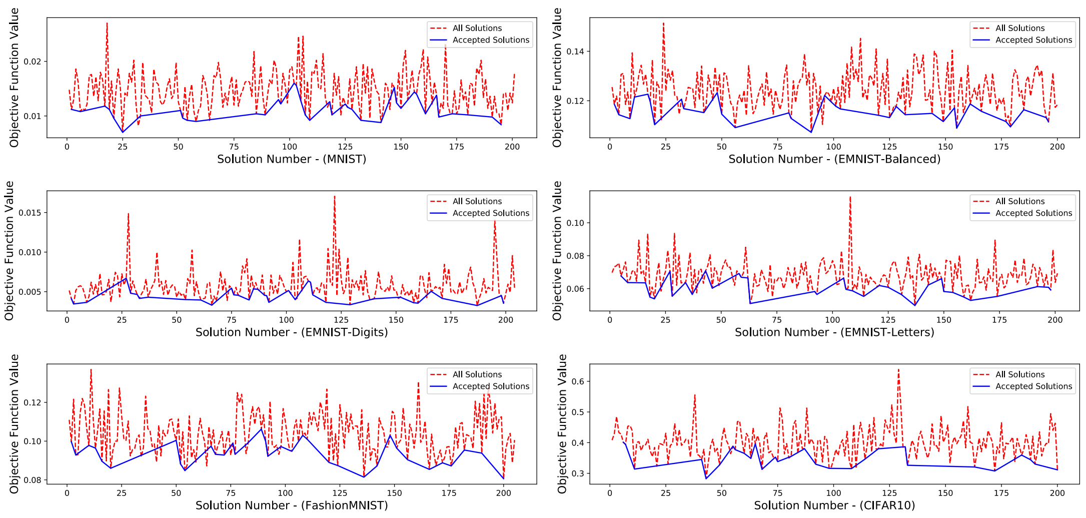


FIGURE 4. μO trajectory for each of six datasets - blue line connects the costs of the accepted solutions.

budget of 200 solutions. The solutions produced during a single run of the μO algorithm and the objective function values (error rates) of the generated solutions are shown in Fig.4. The blue line connects the objective values of the accepted solutions. As the figure suggests, the search trajectory of the algorithm changes from one dataset to another. For example, for the MNIST dataset, the best solution is obtained within the first 25 iterations, whereas for the FashionMNIST dataset, best error rates are obtained just a few iterations before the 200th iteration.

SA is also implemented and run under the same architecture and hyper-parameter constraints and budget limitations. Then, the performance of SA and μO algorithms are compared in terms of both the classification accuracy and the size of the networks. In addition to these two methods, we considered other studies from the literature that employ meta-heuristics for the same task and also report the size of the networks clearly. According to the results given in Table 7, μO algorithm achieves the best accuracy on 4 of 6 datasets, and SA achieves the best accuracy on 3 datasets. As can be seen from the table, the best or the second best accuracy is achieved either by μO or SA both of which generate the networks within predefined architecture and hyper-parameter constraints. When it comes to the size of the networks, μO algorithm generates the smallest networks for 4 datasets, whereas SA generates the smallest network for only 1 dataset. For MNIST, MNIST-Digits and FashionMNIST datasets, μO achieves best accuracy rates with much smaller networks. Although there are other studies reporting better accuracy rates for the same datasets, we have selected only the ones that perform automatic hyper-parameter and/or architecture optimization without any data augmentation or data preprocessing.

The performance of μO and SA are also compared when the constraints given in Section III are relaxed. In Table 8, μO-u stands for the unconstrained version of the μO algorithm. Results obtained by TPE (Tree Parzen Estimator) is

TABLE 7. Comparison of μO with other meta-heuristics.

Dataset	Method	# parameters	accuracy
MNIST	μO	**306,73**	**99.64**
	SA	779,178	99.59
	GA [28]	1,857,601	98.4
	PSO [10]	2,673,000	99.50
EMNIST-Letters	μO	1,005,381	94.25
	SA	**936,827**	**94.51**
EMNIST-Digits	μO	**798,026**	**99.74**
	SA	1,322,922	**99.74**
	GA [28]	3,001,576	99.3
EMNIST-Balanced	μO	**879,055**	90.34
	SA	1,627,887	**90.55**
	GA [28]	1,688,43	88.3
FashionMNIST	μO	**361,834**	**93.78**
	SA	1,196,362	93.43
	GA [28]	4,624,447	90.06
CIFAR10	μO	2,845,962	**86.56**
	SA	1,631,626	85.71
	GA [28]	**172,767**	74.5
	PSO [10]	1,410,797	83.5

also included in the table, since no constraints are imposed on this method. The results state that SA-u achieves the best accuracy on 4 datasets, whereas μO-u and TPE each achieves the best accuracy on only 1 dataset. However, there is no significant accuracy difference between μO and SA. SA achieves 0.09% and 0.175% better accuracy than μO for MNIST and EMNIST-Letters datasets, respectively, but this accuracy improvement comes with a significant increase in computational cost.

The performance of all the approaches given in Tables 7 and 8 is also illustrated in Figure 5, where the methods that result in non-dominated solutions are connected by a red line. For better visualization, this red line is extended so that it intercepts the x and y axis at the extreme values

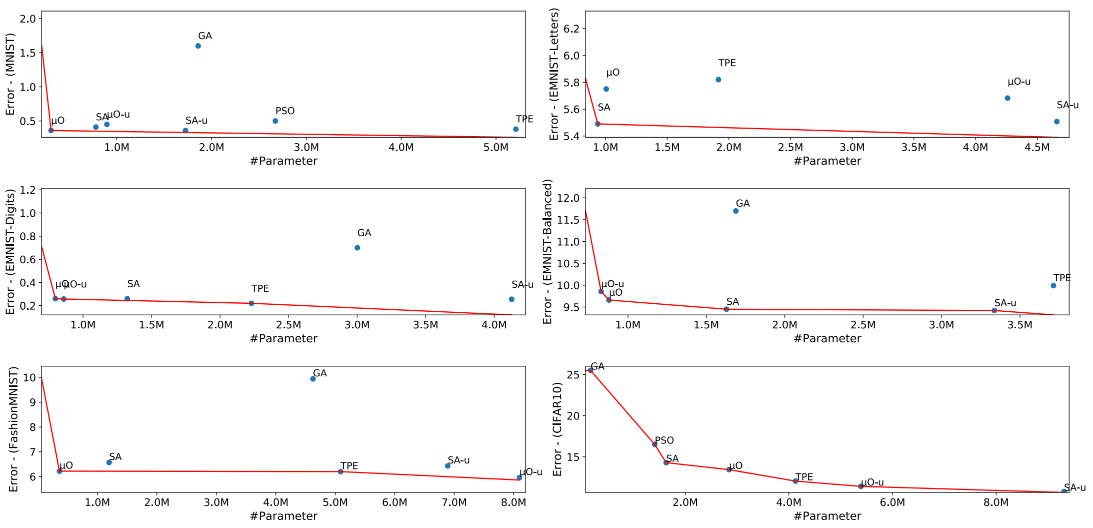


FIGURE 5. Pareto fronts of the methods for each of six datasets.

TABLE 8. Comparison among μO-u, SA-u and TPE.

Dataset	Method	# parameters	accuracy
MNIST	μO-u	**895,402**	99.55 ↓
	SA-u	1,724,458	**99.64** ↑
	TPE	5,210,090	99.62
EMNIST-Letters	μO-u	4,260,123	94.318 ↑
	SA-u	4,659,227	**94.493** ↓
	TPE	**1,915,227**	94.18
EMNIST-Digits	μO-u	**859,338**	99.743 ↑
	SA-u	4,127,626	99.745 ↑
	TPE	2,228,906	**99.78**
EMNIST-Balanced	μO-u	**828,687**	90.144 ↓
	SA-u	3,337,039	**90.58** ↑
	TPE	3,714,095	90.01
FashionMNIST	μO-u	8,096,714	**94.04** ↑
	SA-u	6,890,922	93.57 ↑
	TPE	**5,089,610**	93.80
CIFAR10	μO-u	5,390,506	88.59 ↑
	SA-u	9,309,930	**89.23** ↑
	TPE	**4,127,754**	87.95

of the corresponding objective values. One can see from the figure that μO is always on the front except for one dataset where SA dominates all the others.

We also compared the performance of the networks generated by μO to the state-of-the-art architectures. Two state-of-the-art architectures, namely VGGNet [20] and ResNet [51], are selected for comparison. To be more specific, VGGNet-16 and ResNet-50 architectures are selected and they have been rebuilt by adhering to the original articles [20], [51]. The results presented in Table 9 suggest that the CNNs generated by μO by conforming to the predefined constraints and design rules are able to achieve competitive results with the state-of-the-art architectures with significantly smaller number of parameters. As illustrated in Fig.6, the size has dramatic effect on training duration.

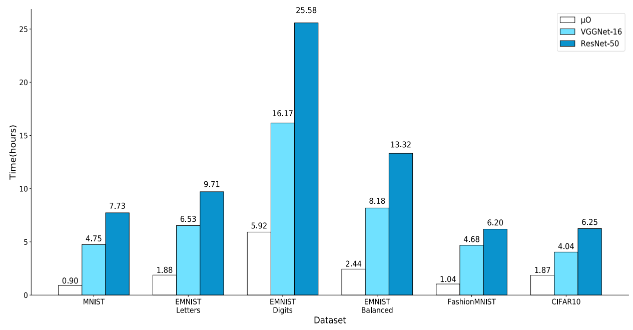


FIGURE 6. Comparison of the training duration of the μO with the training duration required for the state-of-the-art architectures.

D. INVESTIGATING OTHER HYPER-PARAMETERS

In all of the experiments, the batch size, the learning rate, weight initializer and the optimizer method hyper-parameters were excluded from the optimization process. For these parameters, we had adopted the values recommended in the literature. A batch size of 32, a learning rate of 1e-4, weight initializer of Xavier was adopted in all of the experiments and Adam method was selected as the optimization method. In order to investigate the effect of these parameters on the performance of the CNN topologies, we trained and tested the best topology proposed by the μO method for each of the datasets for different values of these parameters given in Table 10. At each step, only one parameter value is allowed to change, other parameters remain the same.

The optimizer method SGD produced slightly better results than the Adam optimizer for all the datasets except CIFAR10 dataset. For the MNIST, EMNIST-Digits, EMNIST-Balanced and FashionMNIST datasets, SGD with the learning-rate of 1e-2 and learning-rate decay of 1e-4; for the EMNIST-Letters dataset, the learning-rate of 1e-3 and the learning-rate decay of 1e-8 produced slightly better results than the Adam optimizer. Xavier method proved to be the

TABLE 9. Comparison of the best architectures obtained by each method in terms of accuracy and the size for each dataset.

Dataset	method	# parameters	accuracy
MNIST	*μO*	306,730	99.64
	VGGNet-16	33,637,066	99.53
	ResNet-50	23,601,930	99.43
EMNIST-Letters	μO	1,005,381	94.25
	VGGNet-16	33,747,685	93.62
	ResNet-50	23,657,253	92.18
EMNIST-Digits	μO	798,026	99.74
	VGGNet-16	33,637,066	99.64
	ResNet-50	23,601,930	99.63
EMNIST-Balanced	μO	879,055	90.34
	VGGNet-16	33,788,655	89.05
	ResNet-50	23,677,743	88.21
FashionMNIST	μO	361,834	93,78
	VGGNet-16	33,637,066	93.02
	ResNet-50	23,601,930	90.15
CIFAR10	μO	2,845,962	86.56
	VGGNet-16	33,638,218	80.91
	ResNet-50	23,608,202	72.62

TABLE 10. Hyper-parameters and the values investigated for further performance increase.

Hyper-parameters	Ranges of Values
Batch Size	32, 64, 128
Learning Rate	1e-2, 1e-3, 1e-4
Optimizer	Adam and SGD with learning decay {1e-4, 1e-8}
Weight Initializer	Xavier, He Uniform, Random Uniform

TABLE 11. Accuracy rates obtained after investigating different values for the fixed hyper-parameters.

Dataset	Original acc.	New acc.	Difference
MNIST	99.57	99.64	+ 0.07
EMNIST-Letters	94.17	94.25	+ 0.08
EMNIST-Digits	99.72	99.74	+ 0.02
EMNIST-Balanced	90.03	90.34	+ 0.31
FashionMNIST	93.32	93.78	+ 0.46
CIFAR10	86.56	86.56	0

best weight initialization technique for all of the datasets. As shown in Table 11, the effect of these new values on the performance of the model is minimal.

E. DISCUSSION OF THE BEST ARCHITECTURES

In this section, a discussion on the best CNN topologies along with the hyper-parameters generated by the μO method is provided. Fig.7 illustrates the best CNNs obtained for each of the datasets (notation is described in Fig.8). One can observe from the figure that all of the topologies excluding the one for the EMNIST-Letters dataset are composed of two convolution blocks each ending with a pooling layer. Another interesting observation is about the selection of the pooling method. Although no constraint had been imposed for this hyper-parameter, maximum pooling is selected at the end of

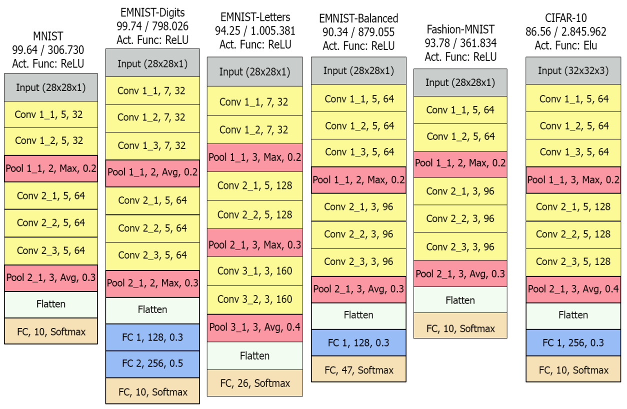


FIGURE 7. The architecture and hyper-parameters of the best CNNs generated for each of the six datasets - the size and the accuracy rates are written in the upper part.

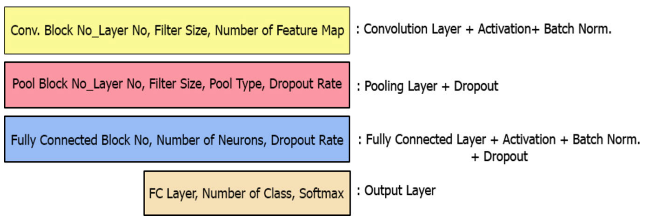

Conv. Block No_Layer No, Filter Size, Number of Feature Map
: Convolution Layer + Activation+ Batch Norm.
Pool Block No_Layer No, Filter Size, Pool Type, Dropout Rate
: Pooling Layer + Dropout
Fully Connected Block No, Number of Neurons, Dropout Rate
: Fully Connected Layer + Activation + Batch Norm. + Dropout
FC Layer, Number of Class, Softmax
: Output Layer

FIGURE 8. Notation used to describe the best CNN topologies.

each intermediary convolution block in all of the topologies except for EMNIST-Digits. For the last convolution block, average pooling is selected which is often used and desired in state-of-the-art architectures. It is also interesting that some best topologies do not include fully connected layers, but still yield in high accuracy rates. Regarding the activation function, *ReLU* is selected in all of the topologies except for the CIFAR-10 dataset. The stability of the μO method is also quantified using a similarity-based stability metric introduced by Dunne *et al.* [60] (see [59] for a detailed discussion on several stability measures). A stability value is calculated in order to determine the amount of variation in the distribution of architectural features and hyper-parameters in the resultant networks generated for six datasets. Using Hamming distance as the similarity measure as proposed in [60], average pair-wise similarity between each possible pairs of the networks is calculated and a stability value of 0.64 is obtained. According to the benchmark scale proposed by Fleiss *et al.* [61], this value indicates a good stability.

VI. CONCLUSION

In this study, Microcanonical Optimization (μO) algorithm which is a variant of Simulated Annealing (SA) is used to automatically discover the best components of the CNNs in terms of both the architecture and the hyper-parameters. Instead of limiting the optimization process within a fixed network architecture, an architecture that is allowed to extend or shrink dynamically is adopted. Moreover, a wide range of hyper-parameters are selected for optimization. In order to expedite the search process, we defined the value ranges for each of those hyper-parameters by following the general design rules recommended in previous studies.

The performance of μO algorithm is compared to the well-established SA meta-heuristic in terms of both the classification accuracy and the size of the networks. The two algorithms are first allowed to run under the same architectural and hyper-parameter constraints and budget limitations. These algorithms are also compared to other studies from the literature that employ meta-heuristics for automatic hyper-parameter optimization and also clearly indicate the size of the networks. The test results based on the six widely-used image recognition data sets reveal that the best and the second best accuracy is achieved by the networks generated by μO and SA methods. However, for some datasets, proposed μO method achieves the best accuracy rates with much smaller networks. When the architectural and hyper-parameter constraints are relaxed, the proposed method again achieves high quality results when compared to SA and also a Bayesian optimization-based method, Tree Parzen Estimator (TPE). In fact, SA achieves slightly better accuracy rates than μO and TPE, but this accuracy improvement comes with a significant increase in computational cost. When the size of the networks is taken into account, unconstrained μO can be preferable to other methods despite slightly reduced accuracy.

The CNN architectures generated by μO algorithm are also compared to the state-of-the-art architectures. Experimental results reveal that the proposed method is able to generate networks that achieve competitive classification results with the state-of-the-art architectures. Moreover, similar or better accuracy rates are achieved with much smaller networks that have been trained on a single GPU within smaller training durations. Therefore, μO method can especially be preferred in situations where fast processing is important, but processing capacity is limited and energy consumption is a big concern. For future research, this problem can be considered as a multi-objective problem, and a Pareto set of CNNs rather than a single CNN can be provided to the user.

REFERENCES

[1] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Proc. Adv. Neural Inf. Process. Syst.*, 1990, pp. 396–404.

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

[3] I. Goodfellow and Y. Bengio, *Courville (2016) a Deep Learning*, vol. 1. Cambridge, MA, USA: MIT Press, 2016.

[4] 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," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[5] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, Feb. 2012.

[6] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Trans. Neural Netw.*, vol. 8, no. 3, pp. 694–713, May 1997.

[7] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, Jun. 2002.

[8] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Evolving deep convolutional neural networks for image classification," *IEEE Trans. Evol. Comput.*, to be published.

[9] B. Ma, X. Li, Y. Xia, and Y. Zhang, "Autonomous deep learning: A genetic DCNN designer for image classification," *Neurocomputing*, vol. 379, pp. 152–161, Feb. 2020.

[10] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "A particle swarm optimization-based flexible convolutional autoencoder for image classification," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 8, pp. 2295–2309, Aug. 2019.

[11] B. Wang, Y. Sun, B. Xue, and M. Zhang, "Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2018, pp. 1–8.

[12] B. Wang, Y. Sun, B. Xue, and M. Zhang, "A hybrid differential evolution approach to designing deep convolutional neural networks for image classification," in *Proc. Australas. Joint Conf. Artif. Intell.* Cham, Switzerland: Springer, 2018, pp. 237–250.

[13] W.-Y. Lee, S.-M. Park, and K.-B. Sim, "Optimal hyperparameter tuning of convolutional neural networks based on the parameter-setting-free harmony search algorithm," *Optik*, vol. 172, pp. 359–367, Nov. 2018.

[14] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, vol. 135. Cambridge, MA, USA: MIT Press, 1998.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," 2013, *arXiv:1312.5602*. [Online]. Available: http://arxiv.org/abs/1312.5602

[16] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," 2016, *arXiv:1611.02167*. [Online]. Available: http://arxiv.org/abs/1611.02167

[17] P. Neary, "Automatic hyperparameter tuning in deep convolutional neural networks using asynchronous reinforcement learning," in *Proc. IEEE Int. Conf. Cognit. Comput. (ICCC)*, Jul. 2018, pp. 73–77.

[18] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller, "Striving for simplicity: The all convolutional net," in *Proc. ICLR*, 2015, pp. 1–14.

[19] T. Yamasaki, T. Honma, and K. Aizawa, "Efficient optimization of convolutional neural networks using particle swarm optimization," in *Proc. IEEE 3rd Int. Conf. Multimedia Big Data (BigMM)*, Apr. 2017, pp. 70–73.

[20] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 3rd Int. Conf. Learn. Represent. (ILCR)*, 2015, pp. 1–13.

[21] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," 2012, *arXiv:1207.0580*. [Online]. Available: http://arxiv.org/abs/1207.0580

[22] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[23] H. Wu and X. Gu, "Max-pooling dropout for regularization of convolutional neural networks," in *Proc. Int. Conf. Neural Inf. Process.* Cham, Switzerland: Springer, 2015, pp. 46–54.

[24] H. Wu and X. Gu, "Towards dropout training for convolutional neural networks," *Neural Netw.*, vol. 71, pp. 1–10, Nov. 2015.

[25] S. Park and N. Kwak, "Analysis on the dropout effect in convolutional neural networks," in *Proc. Asian Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 189–204.

[26] V. Ayumi, L. M. R. Rere, M. I. Fanany, and A. M. Arymurthy, "Optimization of convolutional neural network using microcanonical annealing algorithm," in *Proc. Int. Conf. Adv. Comput. Sci. Inf. Syst. (ICACSIS)*, Oct. 2016, pp. 506–511.

[27] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[28] E. Dufourq and B. A. Bassett, "EDEN: Evolutionary deep networks for efficient machine learning," in *Proc. Pattern Recognit. Assoc. South Afr. Robot. Mechatronics (PRASA-RobMech)*, Nov. 2017, pp. 110–115.

[29] E. Bochinski, T. Senst, and T. Sikora, "Hyper-parameter optimization for convolutional neural network committees based on evolutionary algorithms," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2017, pp. 3924–3928.

[30] S. Fujino, N. Mori, and K. Matsumoto, "Deep convolutional networks for human sketches by means of the evolutionary deep learning," in *Proc. Joint 17th World Congr. Int. Fuzzy Syst. Assoc., 9th Int. Conf. Soft Comput. Intell. Syst. (IFSA-SCIS)*, Jun. 2017, pp. 1–5.

[31] A. Lopez-Rincon, A. Tonda, M. Elati, O. Schwander, B. Piwowarski, and P. Gallinari, "Evolutionary optimization of convolutional neural networks for cancer miRNA biomarkers classification," *Appl. Soft Comput.*, vol. 65, pp. 91–100, Apr. 2018.

[32] F. Assunçao, N. Lourenço, P. Machado, and B. Ribeiro, "DENSER: Deep evolutionary network structured representation," *Genetic Program. Evolvable Mach.*, vol. 20, no. 1, pp. 5–35, Mar. 2019.

[33] A. Baldominos, Y. Saez, and P. Isasi, "Evolutionary convolutional neural networks: An application to handwriting recognition," *Neurocomputing*, vol. 283, pp. 38–52, Mar. 2018.

[34] P. R. Lorenzo, J. Nalepa, M. Kawulok, L. S. Ramos, and J. R. Pastor, "Particle swarm optimization for hyper-parameter selection in deep neural networks," in *Proc. Genetic Evol. Comput. Conf. - GECCO*, 2017, pp. 481–488.

[35] G. L. F. da Silva, T. L. A. Valente, A. C. Silva, A. C. de Paiva, and M. Gattass, "Convolutional neural network-based PSO for lung nodule false positive reduction on CT images," *Comput. Methods Programs Biomed.*, vol. 162, pp. 109–118, Aug. 2018.

[36] J. Nalepa and P. R. Lorenzo, "Convergence analysis of PSO for hyper-parameter selection in deep neural networks," in *Proc. Int. Conf. P2P, Parallel, Grid, Cloud Internet Comput.* Cham, Switzerland: Springer, 2017, pp. 284–295.

[37] T. Domhan, J. T. Springenberg, and F. Hutter, "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves," in *Proc. 24th Int. Conf. Artif. Intell. (IJCAI)*, 2015, pp. 3460–3468.

[38] B. van Stein, H. Wang, and T. Bäck, "Automatic configuration of deep neural networks with EGO," 2018, *arXiv:1810.05526*. [Online]. Available: http://arxiv.org/abs/1810.05526

[39] M. Creutz, "Microcanonical Monte Carlo simulation," *Phys. Rev. Lett.*, vol. 50, no. 19, pp. 1411–1414, May 1983.

[40] S. T. Barnard, "Stereo matching by hierarchical, microcanonical annealing," in *Proc. 10th Int. Joint Conf. Artif. Intell. (IJCAI)*, 1987, pp. 832–835.

[41] G. Bhanot, M. Creutz, and H. Neuberger, "Microcanonical simulation of Ising systems," *Nucl. Phys. B*, vol. 235, no. 3, pp. 417–434, Jul. 1984.

[42] J. A. Torreão and E. Roe, "Microcanonical optimization applied to visual processing," *Phys. Lett. A*, vol. 205, nos. 5–6, pp. 377–382, Sep. 1995.

[43] A. Linhares and J. R. A. Torreão, "Microcanonical optimization applied to the traveling salesman problem," *Int. J. Mod. Phys. C*, vol. 09, no. 01, pp. 133–146, Feb. 1998.

[44] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2011, pp. 2546–2554.

[45] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proc. 30th Int. Conf. Mach. Learn.*, 2013, pp. 115–123.

[46] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. D. Cox, "Hyperopt: A python library for model selection and hyperparameter optimization," *Comput. Sci. Discovery*, vol. 8, no. 1, p. 14008, 2015.

[47] F. Chollet. (2015). *Keras*. [Online]. Available: https://keras.io

[48] A. Krizhevsky and G. Hinton, *Learning Multiple Layers of Features From Tiny Images*, vol. 1, no. 4. Toronto, ON, Canada: Univ. of Toronto, 2009, p. 7.

[49] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:1708.07747*. [Online]. Available: http://arxiv.org/abs/1708.07747

[50] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "EMNIST: An extension of MNIST to handwritten letters," 2017, *arXiv:1702.05373*. [Online]. Available: http://arxiv.org/abs/1702.05373

[51] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[52] R. E. Bellman, *Adaptive Control Processes: A Guided Tour*, vol. 2045. Princeton, NJ, USA: Princeton Univ. Press, 2015.

[53] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," in *Proc. 24th Int. Conf. Mach. Learn. (ICML)*, 2007, pp. 473–480.

[54] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Proc. Int. Conf. Learn. Intell. Optim.* Berlin, Germany: Springer, 2011, pp. 507–523.

[55] M. W. Hoffman and B. Shahriari, "Modular mechanisms for Bayesian optimization," in *Proc. NIPS Workshop Bayesian Optim.*, 2014, pp. 1–5.

[56] J. M. Thompson and K. A. Dowsland, "A robust simulated annealing based examination timetabling system," *Comput. Oper. Res.*, vol. 25, nos. 7–8, pp. 637–648, Jul. 1998.

[57] S. Ceschia, L. Di Gaspero, and A. Schaerf, "Design, engineering, and experimental analysis of a simulated annealing approach to the post-enrolment course timetabling problem," *Comput. Oper. Res.*, vol. 39, no. 7, pp. 1615–1624, Jul. 2012.

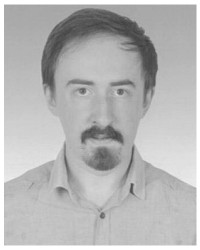
[58] K. I. Smith, R. M. Everson, and J. E. Fieldsend, "Dominance measures for multi-objective simulated annealing," in *Proc. Congr. Evol. Comput.*, 2004, pp. 23–30.

[59] S. Nogueira, K. Sechidis, and G. Brown, "On the stability of feature selection algorithms," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6345–6398, 2017.

[60] K. Dunne, P. Cunningham, and F. Azuaje, "Solutions to instability problems with sequential wrapper-based approaches to feature selection," *J. Mach. Learn. Res.*, vol. 28, pp. 1–22, Sep. 2002.

[61] J. L. Fleiss, B. Levin, and M. C. Paik, "The measurement of interrater agreement," in *Statistical Methods for Rates and Proportions*. Hoboken, NJ, USA: Wiley, 2004, pp. 598–626.

AYLA GÜLCÜ (Member, IEEE) received the B.S. and M.S. degrees from the Electronics and Computer Science Department, Marmara University, Istanbul, Turkey, in 2006, and the Ph.D. degree in engineering management from Marmara University, in 2014. She has been working as an Assistant Professor with the Computer Engineering Department, Fatih Sultan Mehmet University, since 2015. She has been teaching data mining, algorithm design and analysis, data science, and programming courses. Her research interests include discrete optimization, meta-heuristics, machine learning, and deep neural networks.



ZEKI KUŞ received the B.S. degree and the M.S. degree in computer engineering from Fatih Sultan Mehmet University, Istanbul, Turkey, in 2017 and 2019, respectively, where he is currently pursuing the Ph.D. degree in computer engineering. His research interests include meta-heuristics, deep neural networks, and image processing.