Particle Swarm Optimization for Hyper-Parameter Selection in Deep Neural Networks

Pablo Ribalta Lorenzo
Future Processing
Gliwice, Poland
pribalta@future-processing.com

Jakub Nalepa
Future Processing/Silesian University of Technology
Gliwice, Poland
jakub.nalepa@polsl.pl

Michal Kawulok
Future Processing/Silesian University of Technology
Gliwice, Poland
michal.kawulok@polsl.pl

Luciano Sanchez Ramos
Universidad de Oviedo
Departamento de Informática
Oviedo, Spain
luciano@uniovi.es

José Ranilla Pastor
Universidad de Oviedo
Departamento de Informática
Oviedo, Spain
ranilla@uniovi.es

ABSTRACT

Deep neural networks (DNNs) have achieved unprecedented success in a wide array of tasks. However, the performance of these systems depends directly on their hyper-parameters which often must be selected by an expert. Optimizing the hyper-parameters remains a substantial obstacle in designing DNNs in practice. In this work, we propose to select them using particle swarm optimization (PSO). Such biologically-inspired approaches have not been extensively exploited for this task. We demonstrate that PSO efficiently explores the solution space, allowing DNNs of a minimal topology to obtain competitive classification performance over the MNIST dataset. We showed that very small DNNs optimized by PSO retrieve promising classification accuracy for CIFAR-10. Also, PSO improves the performance of existing architectures. Extensive experimental study, backed-up with the statistical tests, revealed that PSO is an effective technique for automating hyper-parameter selection and efficiently exploits computational resources.

CCS CONCEPTS

•**Computing methodologies → Machine learning;** *Neural networks; Bio-inspired approaches;*

KEYWORDS

Deep neural networks, hyper-parameter selection, particle swarm optimization

ACM Reference format:
Pablo Ribalta Lorenzo, Jakub Nalepa, Michal Kawulok, Luciano Sanchez Ramos, and José Ranilla Pastor. 2017. Particle Swarm Optimization for Hyper-Parameter Selection in Deep Neural Networks. In *Proceedings of GECCO '17, Berlin, Germany, July 15-19, 2017,* 8 pages.
DOI: http://dx.doi.org/10.1145/3071178.3071208

1 INTRODUCTION

DNNs learned using backpropagation have achieved a remarkable level of success recently, even outperforming humans at many different tasks. However, these models are very dependent on their parametrization and often require experts to determine which hyper-parameters to modify and how should they be tuned, meaning that for a non-expert it might be hard to find good settings. The need of designing automatic methods for determining the hyper-parameters is especially important for increasingly complex DNN architectures for which trial-and-error is infeasible.

There exist hyper-parameter selection algorithms that have been shown to perform on par with human experts, and in some cases even surpassed them [2]. However, they are still difficult to apply in practice due to their large computational complexity. State-of-the-art methods range from simple grid and random searches [3], to more advanced approaches that balance the exploration and exploitation of the solution space [33]. The latter encompass—among others—the model-based approaches [15], and the Bayesian optimization based on Gaussian processes (GP) [2, 33]. Although evolutionary algorithms have been shown very efficient in solving a plethora of challenging optimization problems [31], so far they were not exploited to optimize the DNN hyper-parameters.

In this work, we tackle the problem of the automated hyper-parameter selection in DNNs using a particle swarm optimization (PSO) algorithm. Albeit the simplicity and generality of PSO, it has been shown to be extremely effective in solving multiple tasks in many fields [11], and it has a big potential for the large-scale parallelization. Finally, combinatorial and real-valued optimization problems are very well suited for PSO. Therefore, we propose this approach for navigating through the large hyper-parameter spaces and retrieving the desired parametrization of the DNNs.

1.1 Contribution

In this paper, we introduce a biologically-inspired method for the hyper-parameter selection, which can be easily parallelized and is independent from the target DNNs. We deploy PSO as a wrapper to the training process to retrieve hyper-parameters that minimize the classification error. We compared the DNNs optimized using PSO with the state of the art, and showed that PSO obtains consistent and very high-quality hyper-parameters. Also, it exposes desirable scalability properties while we benefit from the use of GPUs.

An extensive experimental study, involving scenarios with various DNNs on the MNIST and CIFAR-10 datasets (we focused on the multi-class classification) was undertaken. We showed that:

- PSO obtains competitive results with minimal convolutional neural networks (CNNs) on the MNIST dataset.
- Very small DNN architectures optimized using PSO deliver quite promising classification performance which steadily grows when these topologies are gradually augmented for the demanding CIFAR-10 dataset.
- PSO substantially improves the existent DNN topologies. We investigated the LeNet-4 network and notably boosted its classification performance on the MNIST dataset.

All these experiments are paired with the statistical tests that study the main factors influencing the quality of the final solutions, and were executed to verify the statistical significance of the elaborated results. Additionally, we provide a comprehensive set of illustrations which help get the insights into the behavior and capabilities of the proposed algorithm. The retrieved results emphasize the suitability of PSO to the hyper-parameter selection task, and its ability to yield DNN architectures that can consistently outperform human experts and other state-of-the-art methods.

1.2 Paper Structure

Section 2 discusses the state of the art on the DNN hyper-parameter selection. In Sect. 3, we introduce our PSO for this task. Section 4 displays the experimental results alongside the discussion on the PSO performance and behavior. Section 5 concludes the paper.

2 RELATED LITERATURE

Hyper-parameter selection can be interpreted as an optimization problem where the objective is to find a value that minimizes a loss function $\mathcal{L}(\boldsymbol{T};\mathcal{M})$ for a model $\mathcal{M}$ on a training set $\boldsymbol{T}$, sometimes under certain constraints. This model $\mathcal{M}$ is constructed by a learning algorithm $\mathcal{A}$ using $\boldsymbol{T}$, and typically involves solving an optimization problem. The model may be parametrized by the hyper-parameters $\boldsymbol{\lambda}$, and it is given as $\mathcal{M} = \mathcal{A}(\boldsymbol{T};\boldsymbol{\lambda})$ [5]. The goal of the hyper-parameter selection is to find the parametrization $\boldsymbol{\lambda}^*$ that yields a desired model $\mathcal{M}^*$, while minimizing $\mathcal{L}(\boldsymbol{V};\mathcal{M}^*)$, where $\boldsymbol{V}$ is the validation set. Formally, it becomes [6]:

$$\boldsymbol{\lambda}^* = \arg\min_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{T};\mathcal{M}) = \arg\min_{\boldsymbol{\lambda}} f(\boldsymbol{\lambda};\mathcal{A},\boldsymbol{T},\boldsymbol{V},\mathcal{L}). \tag{1}$$

The objective function f takes the hyper-parameters $\boldsymbol{\lambda}$, and returns the associated loss value. The datasets $\boldsymbol{T}$ and $\boldsymbol{V}$ (where $\boldsymbol{T} \cap \boldsymbol{V} = \emptyset$) are given, and the learning algorithm $\mathcal{A}$ along with the loss function $\mathcal{L}$ are chosen beforehand. The generalization ability of the trained model is quantified using the test set Ψ (unseen during the optimization). Theoretically, it might be possible to compute the gradient of the error measure with respect to the hyper-parameters [29]. Unfortunately, most often this gradient is unavailable in practice.

All approaches to the automatic hyper-parameter search are split into *model-free* and *model-based* methods (see Figure 1). Model-based techniques build a surrogate model of the hyper-parameter space through its careful exploration and exploitation. Alternatively, model-free algorithms do not utilize the knowledge about the solution space extracted during the optimization.

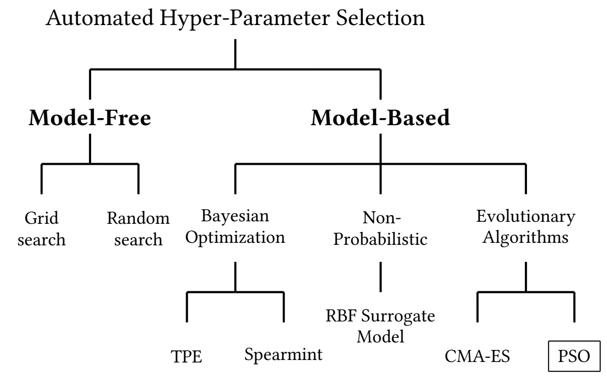


Figure 1: Approaches for the automated hyper-parameter selection (the proposed PSO is annotated with a rectangle).

2.1 Model-Free Hyper-Parameter Selection

Grid search (GS) is commonly used for optimizing the DNN hyper-parameters only if their number is very low. First, the user selects a range of the values to explore. Then, the DNN is trained for every joint specification of hyper-parameters. Typically, GS is performed for the *logarithmic scale* steps, where the best combinations are estimated, progressively refining the search [3]. The evident downside of GS is its time complexity—for k parameters taking n distinct values each (for simplicity, we assume that n is the same for all hyper-parameters), its cost grows exponentially at a rate of $O(n^k)$. This can be however alleviated by the means of loose parallelism.

Random search (RS) is a simple to implement alternative to GS, which often faster converges to an acceptable parametrization. RS is not *adaptive*, meaning that it is not dynamically updated during the course of the experiment (the solutions already found do not influence the search). However, there are approaches designed to enhance its capabilities—the random sampling can be intensified in the neighborhood of the best hyper-parameters. Finally, there exist hybrid algorithms which couple RS with other techniques for refining its performance (e.g., manual updates provided by an expert), quite often in the sequential manner [3, 21].

2.2 Model-Based Hyper-Parameter Selection

Model-based algorithms build a surrogate model of the hyper-parameter space. Then, the hyper-parameters are tuned using this model. Most of such techniques use a Bayesian regression model turning this problem into a trade-off between the exploration (traversing the unknown regions of the space, where the classification performance on $\boldsymbol{V}$ is unknown), and exploitation (analyzing the hyper-parameters which will likely perform well, and are close to the already-investigated "good" hyper-parameters).

In **Spearmint** [33], the density function of the validation error is estimated given a hyper-parameter configuration, i.e., $p(f(\boldsymbol{\lambda})|\boldsymbol{\lambda})$, where $\boldsymbol{\lambda}$ is the set of parameters, and $f(\boldsymbol{\lambda})$ is the error over $\boldsymbol{V}$, using the history of observations $\mathcal{H} = (\boldsymbol{\lambda}, f(\boldsymbol{\lambda}))$. Spearmint exploits the expected improvement, and the automatic relevance determination squared exponential kernel is often used for the GP regression. **HyperOpt** [2] is defined as a tree-structured Parzen estimator (TPE). The GP-based algorithms estimate $p(f(\boldsymbol{\lambda})|\boldsymbol{\lambda})$ directly, whereas the TPE models retrieve $p(\boldsymbol{\lambda}|f(\boldsymbol{\lambda}))$ and $p(f(\boldsymbol{\lambda}))$. TPE benefits from an adaptation that speeds up the search in low-quality regions of the hyper-parameter space. There is an empirical evidence that Spearmint reaches the state-of-the-art performance for small k's, while TPEs work better in large spaces. The major drawback of the GP-based Bayesian optimization methods is that the inference time grows cubically in the number of observations, as it necessitates the inversion of a dense covariance matrix.

Other model-based techniques encompass the covariance matrix adaptation evolution strategies (CMA-ES) [28] and non-probabilistic methods, including RBF surrogate models [16]. In [8], very initial results of evolving DNNs using genetic algorithms were reported.

3 HYPER-PARAMETER SELECTION USING PARTICLE SWARM OPTIMIZATION

Let $f : \mathbb{R}^k \rightarrow \mathbb{R}$ denote an objective function—it maps k hyper-parameters to a real number, being the classification accuracy of the trained DNN obtained for $\boldsymbol{V}$ (it is a fitness function in the proposed PSO). The gradient ∇f is unknown, so the goal is to find a solution $\boldsymbol{\lambda}^*$ for which $f(\boldsymbol{\lambda}^*) \geq f(\boldsymbol{\lambda})$ for all $\boldsymbol{\lambda} \in \chi$, where χ is the set of all available hyper-parameter combinations. In PSO, a population of s particles (representing the hyper-parameter values) undergoes the evolution (Procedure 1, lines 11–26). Each *i-th* particle has its position $\boldsymbol{\lambda}_i \in \mathbb{R}^k$ in the search space (it defines all the hyper-parameters), and the velocity $\boldsymbol{v}_i \in \mathbb{R}^k$ (influencing its movement). Let $\boldsymbol{\lambda}_i^*$ be the best known position of the *i-th* particle, and $\boldsymbol{\lambda}^S$ denote the best known position in the swarm. A noteworthy feature of the proposed PSO is that it is *independent* from the DNN which is being optimized, and can be easily tailored to any new architecture. The following sections discuss its pivotal components in more detail.

3.1 Swarm Initialization

For each *i-th* particle in the swarm, its initial position $\boldsymbol{\lambda}_i$ (in the k-dimensional space) is randomly sampled from a uniform distribution $\mathcal{U}(\boldsymbol{b_l}, \boldsymbol{b_u})$, bounded by the lower and upper limits $\boldsymbol{b_l}$ and $\boldsymbol{b_u}$ (line 4). This position $\boldsymbol{\lambda}_i$ becomes then the best known position of this particle $\boldsymbol{\lambda}_i^*$ (line 5), and if $f(\boldsymbol{\lambda}_i^*) > f(\boldsymbol{\lambda}^S)$, then it is stored as the new best known position in the swarm $\boldsymbol{\lambda}^S$ (line 7). The velocity $\boldsymbol{v}_i$ of this particle is randomly drawn from a uniform distribution—the values are bounded by the upper and lower hyper-parameter limits (line 8). Once the initialization is concluded, the swarm of s particles (being the $\{\boldsymbol{\lambda}_i, \boldsymbol{v}_i, \boldsymbol{\lambda}_i^* = \boldsymbol{\lambda}_i\}_{i=0,1,\cdots<s}$ tuples) is evolved.

3.2 Swarm Evolution

In each swarm generation g (let $G_{\max}$ be the maximum number of generations), all particles have their velocity values updated using the following formula (line 14):

$$\boldsymbol{v}_i \leftarrow \omega \boldsymbol{v}_i + \phi_p \boldsymbol{r_p}(\boldsymbol{\lambda}_i^* - \boldsymbol{\lambda}_i) + \phi_g \boldsymbol{r_g}(\boldsymbol{\lambda}^S - \boldsymbol{\lambda}_i), \quad (2)$$

where $\boldsymbol{r_p}, \boldsymbol{r_g}$ are drawn from a uniform distribution $\mathcal{U}(0,1)$ with the purpose of adding a stochastic component to the velocity updates

Procedure 1 PSO for selecting the DNN hyper-parameters.

```
1:  procedure PSO($s, G_{\max}, \boldsymbol{b_l}, \boldsymbol{b_u}$)
2:      $f(\boldsymbol{\lambda}^S) \leftarrow -\infty$
3:      for $i = 0, 1, \ldots, s$ do                          ▷ Swarm initialization
4:          $\boldsymbol{\lambda}_i \sim \mathcal{U}(\boldsymbol{b_l}, \boldsymbol{b_u})$
5:          $\boldsymbol{\lambda}_i^* \leftarrow \boldsymbol{\lambda}_i$
6:          if $f(\boldsymbol{\lambda}_i^*) > f(\boldsymbol{\lambda}^S)$ then
7:              $\boldsymbol{\lambda}^S \leftarrow \boldsymbol{\lambda}_i^*$
8:          $\boldsymbol{v}_i \sim \mathcal{U}(-|\boldsymbol{b_u} - \boldsymbol{b_l}|, |\boldsymbol{b_u} - \boldsymbol{b_l}|)$
9:      $g \leftarrow 1$
10:     $\boldsymbol{\lambda}^S_{\text{prev}} \leftarrow \boldsymbol{\lambda}^S$
11:     while $g \leq G_{\max}$ do                              ▷ Swarm evolution
12:         for $i = 0, 1, \ldots, s$ do
13:             $\boldsymbol{r_p}, \boldsymbol{r_g} \sim \mathcal{U}(0, 1)$
14:             Update velocity $\boldsymbol{v}_i$                         ▷ See Equation 2
15:             $\boldsymbol{\lambda}_i \leftarrow \boldsymbol{\lambda}_i + \boldsymbol{v}_i$
16:             if $f(\boldsymbol{\lambda}_i) > f(\boldsymbol{\lambda}_i^*)$ then
17:                 $\boldsymbol{\lambda}_i^* \leftarrow \boldsymbol{\lambda}_i$
18:                 if $f(\boldsymbol{\lambda}_i^*) > f(\boldsymbol{\lambda}^S)$ then
19:                     $\boldsymbol{\lambda}^S \leftarrow \boldsymbol{\lambda}_i^*$
20:                     if $||\boldsymbol{\lambda}^S - \boldsymbol{\lambda}^S_{\text{prev}}|| < \delta$ then
21:                         return $\boldsymbol{\lambda}^S$
22:                     if $f(\boldsymbol{\lambda}^S) - f(\boldsymbol{\lambda}^S_{\text{prev}}) < \epsilon$ then
23:                         return $\boldsymbol{\lambda}^S$
24:         $g \leftarrow g + 1$
25:         $\boldsymbol{\lambda}^S_{\text{prev}} \leftarrow \boldsymbol{\lambda}^S$
26:     return $\boldsymbol{\lambda}^S$
```

(to diversify the search), ω is an inertia weight scaling the velocity, and ϕ_p and ϕ_g are the acceleration coefficients. These factors scale the influence of the best particle position ($\boldsymbol{\lambda}_i^*$), and the best swarm position ($\boldsymbol{\lambda}^S$) on the velocity changes. The position $\boldsymbol{\lambda}_i$ is finally updated (line 15). Afterwards, the best position for each particle is modified (line 17), along with the best swarm position (line 19)—they are updated only if they changed. The evolution progresses until one of the following termination conditions has been reached:

(1) The best position in the swarm $\boldsymbol{\lambda}^S$ has been displaced by less than a minimum *step size* denoted by δ (line 21).
(2) The fitness value of the best particle increased by less than a threshold denoted by ϵ (line 23).
(3) The maximum number of swarm generations $G_{\max}$ have been processed (line 11).

The first stopping condition helps avoid oscillating between two neighboring (high-quality) solutions, whereas the second one will be met if the swarm optimization converges to the well-fitted particle, which will likely not be improved further. Finally, the best position in the swarm $\boldsymbol{\lambda}^S$ is returned (line 26).

The PSO time complexity is affected by the number of hyper-parameters, and it can be expressed as:

$$\mathcal{T}_{\text{PSO}} = s \cdot f(\boldsymbol{\lambda}_k) \cdot G_{\max}. \quad (3)$$

Since s and $G_{\max}$ are constant, the time complexity is directly traced back to the evaluation of $f(\boldsymbol{\lambda}_k)$ which grows linearly with k.

4 EXPERIMENTAL VALIDATION

4.1 Experimental Setup

PSO for the hyper-parameter selection was implemented in Python[1] using the NumPy library, and DNNs were trained using Keras [4] with a TensorFlow [1] backend over CUDA 8.0 and CuDNN 5.1. The experiments were run on the following hardware:

- Intel Xeon E5-2698 v3 (40M Cache, 2.30 GHz) with 128GB of RAM and NVIDIA Tesla K80 GPU 24GB DDR5.
- Intel i7-6850K (15M Cache, 3.80 GHz) with 32 GB RAM and NVIDIA Titan X Ultimate Pascal GPU 12GB GDDR5X.

We used a fixed parametrization for the internals of PSO across all experiments, where $\omega = \phi_p = \phi_g = 0.5$ (search parameters). The termination criteria are defined as: $G_{\max} = 100$ (maximum number of generations), $\delta = 10^{-4}$ (minimum change of the best particle position), and $\epsilon = 10^{-4}$ (minimum fitness improvement). For all experiments, we use 10-fold cross-validation, where $|\boldsymbol{T}| = 9\,|\boldsymbol{V}|$. The set $\boldsymbol{T}$ is used for the DNN training, while $\boldsymbol{V}$ is utilized to calculate the fitness during the PSO optimization (as discussed in Section 3). Finally, the classification accuracy over the test set Ψ quantifies the generalization performance of the trained DNN. In order to speed up the PSO execution, we exploit the *archive* and cache the trained DNNs (for all particles in the swarm). Also, this archive provides an insightful view to the execution of the algorithm and keeps track of which particle positions were visited during the optimization.

An early stopping condition was added to the DNN training, ensuring that if after 5 training epochs the accuracy on $\boldsymbol{V}$ does not increase, then the training is terminated. By default, networks are trained for a maximum of 100 epochs using batches of size 128, with the objective of minimizing the categorical cross-entropy (multi-class log loss). Finally, we used the ADAM optimizer [19] with the default parametrization for controlling the learning rate.

4.2 Datasets

In this paper, we focused on the multi-class classification, and utilized two well-known benchmark sets—**MNIST** [23] and **CIFAR-10** [20]. These sets were used to assess the classification performance of the DNNs optimized using PSO. As already mentioned, we perform 10-fold cross validation. The validation sets $\boldsymbol{V}$ are extracted from $\boldsymbol{T}$'s provided for both sets, $|\boldsymbol{T}| = 9\,|\boldsymbol{V}|$, and $\boldsymbol{T} \cap \boldsymbol{V} = \emptyset$ (each PSO experiment—for each configuration—is executed 10× without any overlaps between the validation images across folds).

MNIST is a dataset of handwritten digits consisting of 70,000 grayscale images (28×28 pixels) divided into 10 classes, with approx. 7,000 images per class (Figure 2). There are 60,000 training images in $\boldsymbol{T}$, and 10,000 test images in Ψ (sets are balanced).

Figure 2: Example images from the MNIST dataset.

The CIFAR-10 dataset encompasses 60,000 color images (three channels, 32×32 pixels) belonging to 10 classes, with 6,000 images per class. It is divided into 50,000 training ($\boldsymbol{T}$) images and 10,000 test (Ψ) images (sets are balanced). We present the example CIFAR-10 images in Figure 3. This set is much more challenging compared with MNIST (see the intra-class differences between the images).

Figure 3: Example images from the CIFAR-10 dataset.

4.3 Experimental DNN Architecture

Designing an appropriate DNN architecture is challenging and it is one of the main obstacles in exploiting DNNs in practice. Here, we verify if a very small DNN, optimized using PSO can achieve the acceptable accuracy over a non-trivial multi-class dataset.

Table 1: Parameters of the layers in SimpleNet.

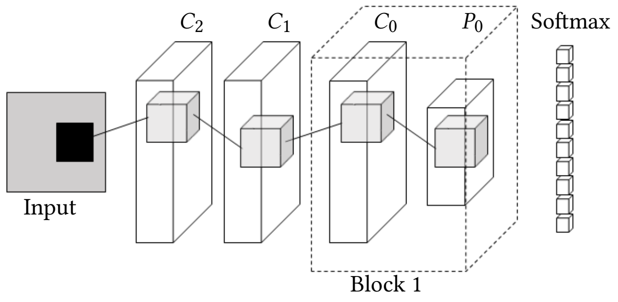
Layer type	Parameters	Values
Convolutional (C)	Receptive field size ($s_F \times s_F$)	$s_F \geq 2$
	No. of receptive fields (n)	$n \geq 1$
Max Pooling (P)	Stride size (ℓ)	$\ell \geq 2$
	Receptive field size (s_P)	$s_P \geq 2$

SimpleNet is our experimental architecture defined by a *block* of the convolutional and max pooling layers, terminated by a Softmax activation. It can be easily expanded by adding more blocks or more convolutional layers. The parameters of the convolutional and max pooling layers, along with their allowed ranges are given in Table 1 (the stride equals 1 for the convolutional layers). SimpleNet architectures can be distinguished using their unique names. Suffixes refer to the number of blocks, while subscripts indicate the number of the appended convolutional layers. Convolutional layers are denoted as C_i, whereas max pooling layers as P_i, where i is the position of the layer in the DNN. This way, a SimpleNet architecture with two blocks is given as SimpleNet-2. On the other hand, a SimpleNet model with one block and two additional convolutional layers is denoted as SimpleNet-1_2 (Figure 4). The reason for adding the convolutional layers in the shallower levels responds to the desire of maximizing the initial feature extraction, while ensuring at least one downsampling operation before the Softmax activation.

4.4 Analysis and Discussion

Our experimental study is divided into three main experiments. At first, we verify how the swarm sizes affect the PSO capabilities for SimpleNet-1, and compare the classification results retrieved using this architecture with the state of the art for MNIST (Section 4.4.1). Afterwards, we investigate various SimpleNet architectures optimized using PSO for CIFAR-10 (Section 4.4.2). Finally, we optimize an existent DNN architecture (LeNet-4) and show that PSO can significantly boost its classification performance (Section 4.4.3).

[1]The implementation has been provided as the supplementary material to this paper.



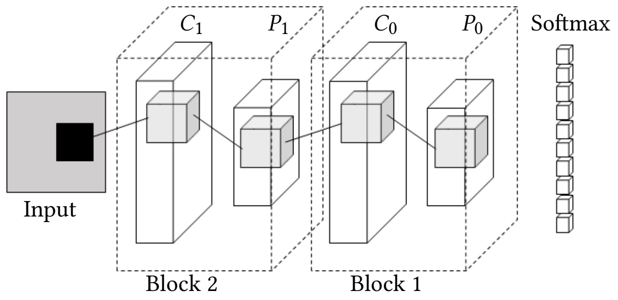


Figure 4: Example of the SimpleNet-1_2 architecture with one *block* and two convolutional layers (top). Example of the SimpleNet-2 architecture with two *blocks* (bottom).

The experiments are coupled with the pair-wise Wilcoxon tests executed to check the statistical importance of the results.

4.4.1 Impact of the Swarm Size on PSO. In this experiment, we verify the influence of the swarm size s on the PSO performance and convergence capabilities (the globally optimal hyper-parameter values were retrieved using GS), and compare its performance with the state of the art. The following swarm sizes were investigated: $s = \{4, 10, 16\}$, and the hyper-parameter limits are gathered in Table 2. This experiment was run in the NVIDIA K80 GPU setup.

Table 2: Lower and upper hyper-parameter boundaries.

	Layer	b_l	b_u
Block 1	C_0	$\{n = 1, s_F = 2\}$	$\{n = 16, s_F = 8\}$
	P_0	$\{s_P = 2, \ell = 2\}$	$\{s_P = 4, \ell = 4\}$

In Table 3, we display the results retrieved for SimpleNet-1 with different values of s, together with those elaborated using GS and RS. Since GS traverses the entire solution space, it always reports the globally optimal hyper-parameter values for a given architecture. Interestingly, RS also reaches this global optimum. PSO extracts very similar solutions for all s values (the maximum difference in the Ψ classification accuracy between SimpleNet-1 optimized by GS and PSO is $4.5 \cdot 10^{-3}$ for $s = 4$) at only a fraction of the GS and RS execution time (PSO with 4 particles is almost 94× faster than GS and 43× faster than RS).

It is worth mentioning that only 4.86% of the entire hyper-parameter space was explored using PSO with $s = 16$. For other swarm sizes, the number of visited positions (i.e., hyper-parameter configurations) during the optimization is even smaller, but the Ψ classification accuracy is consistently very high. Also, we report the number of generations g_s after which PSO converged to the final (best) DNN parameter values (they were not further improved). Since g_s's are quite small, PSO could have been terminated earlier without influencing the quality of the hyper-parameters. This would allow for decreasing its execution time even more.

Table 3: Comparison between GS, RS and PSO for the SimpleNet-1 dataset with different swarm sizes s.

Algorithm	s	Time (sec.)	Positions	g_s	Acc. on Ψ
GS	-	87,356	1,008	—	0.9897
RS	-	39,906	400	—	0.9897
PSO	4	934	14	14	0.9852
PSO	10	2,091	29	20	0.9864
PSO	16	13,892	49	23	0.9871

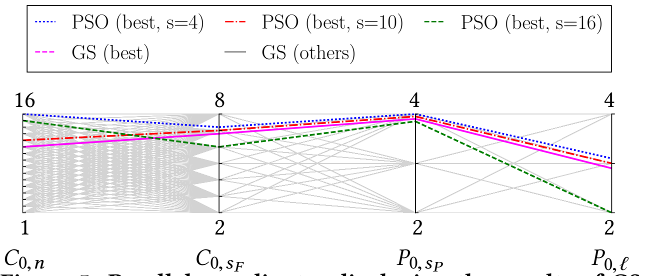


Figure 5: Parallel coordinates displaying the results of GS for SimpleNet-1 along with the PSO results.

An important aspect of the PSO optimization process is to ensure that the particles converge to the global optimum, and they do not get trapped in suboptimal regions of the solution space. In Figure 5, we visualize the final (best) DNN parameters elaborated using PSO and compare them with the GS results. It can be observed that all particles converge to the global optimum obtained using GS.

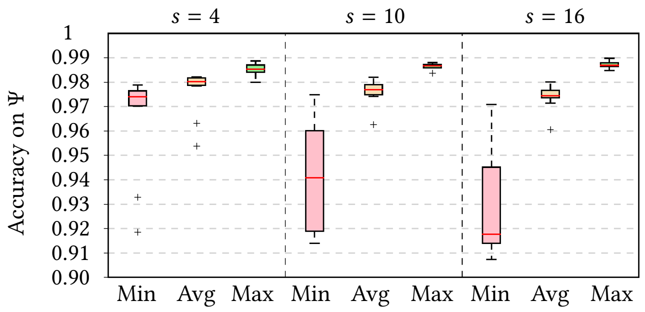


Figure 6: Minimum, average and maximum accuracy (of the best PSO particles across all folds) obtained for Ψ using the optimized SimpleNet-1 (for all swarm sizes s).

Another desirable property displayed by PSO in this experiment was the consistency in the quality of the hyper-parameters obtained across ten independent runs. In Figure 6, we show that for all swarm sizes SimpleNet-1 optimized using PSO performs consistently well for Ψ. Interestingly, increasing s may cause worsening the *worst* hyper-parameters (see the minimum values in Figure 6)—large swarms could not effectively improve the initial low-quality positions. This is alleviated for smaller s values, due to more rapid particle movement. On the other hand, the *best* parameters were gradually improved for larger swarms (however, the differences were statistically important at $p = .05$ only between $s = 4$ and $s = 16$, according to the two-tailed Wilcoxon tests). Therefore, even very small swarms retrieve high-quality DNN hyper-parameters.

Table 4: Classification error (in %) for the MNIST dataset. We boldfaced the DNNs optimized using PSO.

Classifier	Error rate (%)
Pairwise linear classifier [22]	7.6
Convolutional Clustering [10]	1.4
SimpleNet-1, s=4	**1.13**
SimpleNet-1, s=10	**1.12**
LeNet-4 [22]	1.1
SimpleNet-1, s=16	**1.08**
Product of stumps on Haar features [17]	0.87
Boosted LeNet-4 [22]	0.7
LeNet-4 with PSO	**0.66**
K-NN with non-linear deformation [18]	0.52
NiN [27]	0.47
Maxout Networks [12]	0.45
DSN [24]	0.39
R-CNN [25]	0.31
MultiColumn DNN [32]	0.23

Although SimpleNet-1 is a remarkably small architecture, the PSO-extracted parameters made it competitive with the state of the art for MNIST (Table 4). The main drawback of the top performing classifiers for this set is their complexity, making them difficult to parameterize and expensive to train. Also, the data augmentation or preprocessing (e.g., in [17, 32]) are very problem-dependent and influence their usability. Finally, the improvements in the learning structure require detailed *a priori* knowledge of the problem in many approaches (e.g., in [7, 35]). These issues are mitigated by PSO, and the SimpleNet-1 results underpin its potential for achieving high performing models that are much easier to interpret.

Table 5: Comparison between GS, RS and PSO for the CIFAR-10 dataset with different swarm sizes s.

Algorithm	s	Time (sec.)	Positions	g_s	Acc. on Ψ
GS	—	152,527	2,160	-	0.5566
RS	—	61,151	400	-	0.5343
PSO	4	1,012	29	76	0.5106
PSO	10	2,584	76	98	0.536
PSO	16	4,951	125	99	0.5508

4.4.2 Incrementing the SimpleNet Architecture. In this experiment, we analyze the ability of PSO to maximize the classification performance of various small (however gradually increased) SimpleNet-based architectures for a more demanding CIFAR-10 dataset. The experiment was run in the NVIDIA Titan GPU setup.

First, GS and RS results are compared with the best PSO hyper-parameters for SimpleNet-1 with different swarm sizes (Table 5). Best hyper-parameters reported by PSO are of very similar quality, and they were obtained in significantly shorter time. For $s = 16$, PSO significantly outperforms RS. The results for CIFAR-10 exhibit the same properties as those obtained for MNIST (Table 3), even though it is a much more demanding dataset. For all values of s for CIFAR-10, the PSO optimization time is repeatedly reduced by two orders of magnitude when compared to GS. Additionally, it is visible that for a complex dataset, increasing s helps improve the search capabilities of PSO more substantially.

To explore the capacity of PSO ($s = 4$) to find suitable hyper-parameters in a vast search space, the following SimpleNet-1 variations were introduced: SimpleNet-1_1, SimpleNet-1_2, SimpleNet-1_3, and SimpleNet-2 (see Section 4.3 for details). Another area of focus here is to determine if PSO can scale as the DNN architectures become complex and the cost function is more expensive to evaluate.

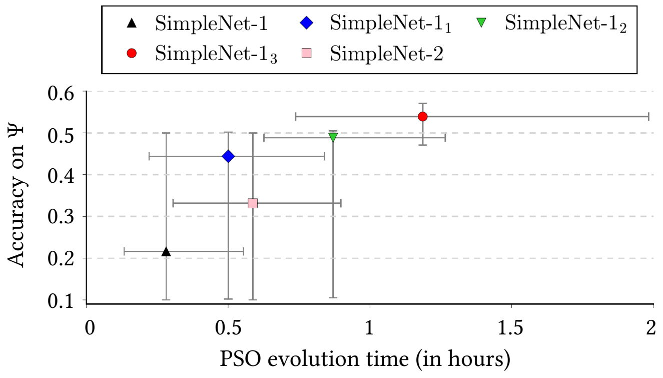


Figure 7: Pareto front of the results (avg. Ψ accuracy of the best PSO particles across all folds vs. avg. PSO time—the horizontal and vertical bars render the min. and max. values) of different architectures (we also show the dominated DNN).

SimpleNet-2 retrieves slightly better hyper-parameters compared with SimpleNet-1 on average (Figure 7), however its limited ability to perform an effective feature extraction renders it quite inefficient for CIFAR-10. On the other hand, the variations of SimpleNet-1 augmented with additional convolutions progress better, achieving promising results and displaying very steady and predictable increase in the accuracy—their feature extraction capabilities are slowly boosted by adding new convolutional layers. For SimpleNet-1_3, we observe a remarkable ability to provide consistent and high-quality hyper-parameters, especially compared with the smaller SimpleNet-1 variants, where many combinations of hyper-parameters are inappropriate for CIFAR-10. Ultimately, this reflects the ability of PSO to avoid local optima while converging towards the best possible results, even in complex solution spaces. Small DNNs optimized using PSO deliver promising accuracy over Ψ. Although they are still far from the state of the art (Table 6), these results are achieved with much simpler DNNs. It shows that CIFAR-10 is quite challenging and requires larger architectures—increasing our DNNs should allow for providing the performance competitive to the state of the art, but it requires further investigation. The differences in accuracy between the baseline SimpleNet-1 and all other augmented topologies were statistically significant (pair-wise Wilcoxon test, $p = .05$).

Table 6: Classification error (in %) for the CIFAR-10 dataset. We boldfaced the DNNs optimized using PSO.

Classifier	Error rate (%)
SimpleNet-1, $s = 4$	**48.94**
SimpleNet-1, $s = 10$	**46.40**
SimpleNet-2, $s = 4$	**44.89**
SimpleNet-1, $s = 16$	**44.92**
SimpleNet-1$_1$, $s = 4$	**42.48**
SimpleNet-1$_2$, $s = 4$	**41.53**
SimpleNet-1$_3$, $s = 4$	**40.41**
Fast-learning shallow CNN [30]	25.14
Recursive perceptual representations [34]	20.3
Discriminative feature learning with CNNs [9]	18.00
Stochastic Pooling [35]	15.13
Competitive multi-scale convolution [26]	6.87
Deep residual learning [14]	6.43
Fractional MaxPooling [13]	3.47

The results showed that PSO efficiently optimizes small DNNs for a difficult set, and proved that it reliably extracts hyper-parameters that are close to the global optimum. Moreover, as the architectures get increasingly complex and the number of hyper-parameters grows, PSO scales gracefully offering a steady and predictable increase in the computation time (see Figure 7), as opposed to other algorithms, e.g., GS, which become unusable quickly.

Table 7: Original LeNet-4 hyper-parameter values, lower and upper parameter boundaries for PSO, along with the hyper-parameters obtained using PSO.

Layer	Orig. LeNet-4 [22]	b_l	b_u	**PSO results**
C_0	$n = 24$	$n = 1$	$n = 48$	$n = 38$
	$s_F = 5$	$s_F = 2$	$s_F = 10$	$s_F = 6$
P_0	$s_P = 2$	$s_P = 2$	$s_P = 4$	$s_P = 4$
	$\ell = 2$	$\ell = 2$	$\ell = 4$	$\ell = 2$
C_1	$n = 50$	$n = 1$	$n = 100$	$n = 96$
	$s_F = 5$	$s_F = 2$	$s_F = 10$	$s_F = 9$
P_1	$s_P = 2$	$s_P = 2$	$s_P = 4$	$s_P = 2$
	$\ell = 2$	$\ell = 2$	$\ell = 4$	$\ell = 4$
FC_0	$s_{FC} = 500$	$s_{FC} = 1$	$s_{FC} = 1000$	$s_{FC} = 23$

4.4.3 Optimizing an Existent DNN Architecture. Here, we optimize an existent DNN using PSO ($s = 5$) for MNIST—we focus on the well-known canonical DNN model (LeNet-4) which has a reported error rate of 1.1% on this dataset [22]. Let FC_i define a fully connected layer with s_{FC} denoting its size. The original LeNet-4 hyper-parameters are given in Table 7, along with the parameter limits for PSO—the original LeNet-4 values have been doubled. The experiment was run in the NVIDIA Titan GPU setup.

In Figure 8, we observe the example snapshots of the swarm in the first, second and last PSO generation (top to bottom). The particles steadily converge to the final solution, in contrast with the first generation, in which particles were scattered uniformly in the search space. This behavior shows that even in large hyper-parameter spaces, PSO is able to effectively guide the search and that it is a suitable solution to the hyper-parameter optimization.

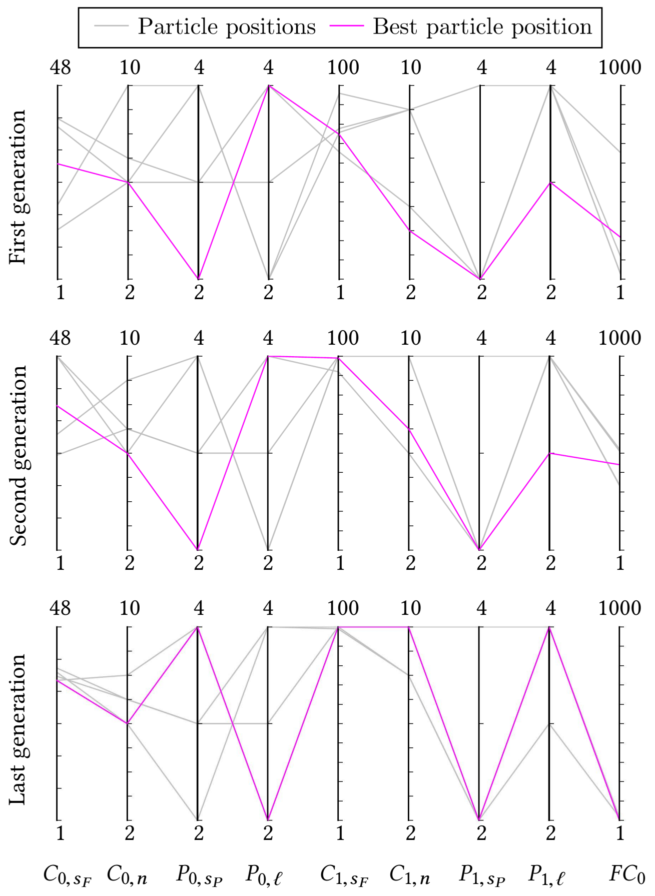


Figure 8: Parallel coordinates showing particle positions in the first, second, and last PSO generation for LeNet-4.

Table 7 shows that PSO can produce new settings for LeNet-4 which increase its classification accuracy on Ψ to the maximum of 0.9934 (0.66% error rate). With this new automatically obtained hyper-parameters (note that they notably differ from the original LeNet-4, see e.g., C_0, C_1, and FC_0 parameter values), LeNet-4 exposes the performance very close to best-known state-of-the-art models (Table 4), and significantly outperforms LeNet-4 with its original hyper-parameters. Figure 9 visualizes the consistency of the results (in terms of the Ψ accuracy), as well as the optimization time, with an average of less than 2 hours. It makes PSO very affordable for such complex DNN model. Also, the results show that even an architecture considered as a perfect example of design by experts can be successfully optimized automatically by PSO, and in this case, even halve its previous error rate. The two-tailed Wilcoxon tests revealed that LeNet-4 optimized using the proposed PSO retrieved better Ψ classification performance for MNIST compared with all investigated SimpleNet-based DNN architectures (the differences are statistically important at $p = .05$)

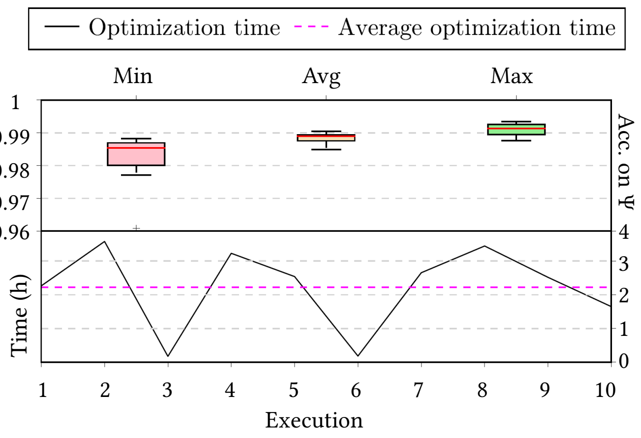


Figure 9: Minimum, average and maximum accuracy (for the best PSO particles across all folds) on Ψ of the optimized LeNet-4 for MNIST (top). Time for each PSO execution (in h), and the average time (bottom).

5 CONCLUSIONS AND OUTLOOK

In this paper, we proposed a PSO algorithm for the hyper-parameter optimization in DNNs. The population of particles (each representing a set of hyper-parameter values) is evolved in search of hyper-parameters which yield the best classification performance of the DNN. An experimental study performed on multi-class datasets (MNIST and CIFAR-10), coupled with the statistical tests revealed that PSO effectively traverses the solution space and delivers consistent and very high-quality results across different experimental setups, clearly surpassing human expertise when optimizing an existent DNN architecture designed by experts. A comprehensive set of illustrations provided insights into the PSO capabilities and behavior. We showed that augmenting minimal DNNs and optimizing them using PSO can be an effective tool for challenging datasets. Finally, PSO exposed very desired scalability properties.

Our future work focuses on developing a systematic methodology for balancing the exploration and exploitation of the hyper-parameter space, and running PSO for larger DNNs. We work on the adaptation schemes which will allow for increasing the initial (simple) DNNs for challenging tasks in the medical imaging field.

ACKNOWLEDGMENTS

PRL, JN and MK were supported by the Polish National Centre for Research and Development under the Innomed grant POIR.01.02.00-00-0030/15. LSR and JRP were supported by the MEIC from Spain/ FEDER (grants TEC2015-67387-C4-3-R and TIN2014-56967-R) and by the Principality of Asturias grant FC-15-GRUPIN14-073.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). http://tensorflow.org/ Software available from tensorflow.org.

[2] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for Hyper-Parameter Optimization. In *Proc. NIPS*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger (Eds.). Curran Assoc., 2546–2554.

[3] James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research* 13 (2012), 281–305.

[4] François Chollet. 2015. Keras. https://github.com/fchollet/keras. (2015).

[5] Marc Claesen and Bart De Moor. 2015. Hyperparameter Search in Machine Learning. *CoRR* abs/1502.02127 (2015).

[6] Marc Claesen, Jaak Simm, Dusan Popovic, Yves Moreau, and Bart De Moor. 2014. Easy Hyperparameter Search Using Optunity. *CoRR* abs/1412.1114 (2014).

[7] Sanjoy Dasgupta and David Mcallester. 2013. Regularization of Neural Networks using DropConnect. In *Proc. ICML*, Vol. 28. JMLR Conf. Proc., 1058–1066.

[8] Omid E. David and Iddo Greental. 2014. Genetic Algorithms for Evolving Deep Neural Networks. In *Proc. GECCO*. ACM, USA, 1451–1452.

[9] Alexey Dosovitskiy, Jost Tobias Springenberg, Martin A. Riedmiller, and Thomas Brox. 2014. Discriminative Unsupervised Feature Learning with Convolutional Neural Networks. *CoRR* abs/1406.6909 (2014).

[10] Aysegul Dundar, Jonghoon Jin, and Eugenio Culurciello. 2015. Convolutional Clustering for Unsupervised Learning. *CoRR* abs/1511.06241 (2015).

[11] Ahmed A. Esmin, Rodrigo A. Coelho, and Stan Matwin. 2015. A Review on Particle Swarm Optimization Algorithm and Its Variants to Clustering High-dimensional Data. *Artificial Intelligence Review* 44, 1 (2015), 23–45.

[12] Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron C. Courville, and Yoshua Bengio. 2013. Maxout Networks. In *Proc. ICML*. 1319–1327.

[13] Benjamin Graham. 2014. Fractional Max-Pooling. *CoRR* abs/1412.6071 (2014).

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *CoRR* abs/1512.03385 (2015).

[15] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Sequential Model-based Optimization for General Algorithm Configuration. In *Proc. LION*. Springer-Verlag, Berlin, Heidelberg, 507–523.

[16] Ilija Ilievski, Taimoor Akhtar, Jiashi Feng, and Christine A. Shoemaker. 2016. Hyperparameter Optimization of Deep Neural Networks Using Non-Probabilistic RBF Surrogate Model. *CoRR* abs/1607.08316 (2016).

[17] Balázs Kégl and Róbert Busa-Fekete. 2009. Boosting Products of Base Classifiers. In *Proc. ICML*. ACM, New York, NY, USA, 497–504.

[18] Daniel Keysers, Thomas Deselaers, Christian Gollan, and Hermann Ney. 2007. Deformation models for image recognition. *IEEE TPAMI* 29, 8 (2007), 1422–1435.

[19] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014).

[20] Alex Krizhevsky. 2009. *Learning multiple layers of features from tiny images*. Technical Report. MIT.

[21] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. 2007. An Empirical Evaluation of Deep Architectures on Problems with Many Factors of Variation. In *Proc. ICML*. ACM, USA, 473–480.

[22] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. In *Proc. IEEE*. 2278–2324.

[23] Yann LeCun and Corinna Cortes. 2010. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/, (2010).

[24] Chen-Yu Lee, Saining Xie, Patrick W. Gallagher, Zhengyou Zhang, and Zhuowen Tu. 2015. Deeply-Supervised nets. In *Proc. AISTATS*, Guy Lebanon and S. V. N. Vishwanathan (Eds.), Vol. 38. JMLR.

[25] Ming Liang and Xiaolin Hu. 2015. Recurrent Convolutional Neural Network for Object Recognition. In *Proc. CVPR*. 3367–3375.

[26] Zhibin Liao and Gustavo Carneiro. 2015. Competitive Multi-scale Convolution. *CoRR* abs/1511.05635 (2015).

[27] Min Lin, Qiang Chen, and Shuicheng Yan. 2013. Network In Network. *CoRR* abs/1312.4400 (2013).

[28] Ilya Loshchilov and Frank Hutter. 2016. CMA-ES for Hyperparameter Optimization of Deep Neural Networks. *CoRR* abs/1604.07269 (2016).

[29] Dougal Maclaurin, David Duvenaud, and Ryan Adams. 2015. Gradient-based Hyperparameter Optimization through Reversible Learning. In *Proc. ICML*, David Blei and Francis Bach (Eds.). JMLR Conf. Proc., 2113–2122.

[30] Tony Vladusich Mark D. McDonnell. 2013. Enhanced Image Classification With a Fast-Learning Shallow Convolutional Neural Network. *CoRR* abs/1503.04596 (2013).

[31] Jakub Nalepa and Michal Kawulok. 2014. A Memetic Algorithm to Select Training Data for Support Vector Machines. In *Proc. GECCO*. ACM, USA, 573–580.

[32] Jurgen Schmidhuber. 2012. Multi-column Deep Neural Networks for Image Classification. In *Proc. CVPR*. IEEE Computer Society, USA, 3642–3649.

[33] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *Proc. NIPS*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran Assoc., 2951–2959.

[34] Oriol Vinyals, Yangqing Jia, Li Deng, and Trevor Darrell. 2012. Learning with Recursive Perceptual Representations. In *Proc. NIPS*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran Assoc., 2825–2833.

[35] Matthew D. Zeiler and Rob Fergus. 2013. Stochastic Pooling for Regularization of Deep Convolutional Neural Networks. *CoRR* abs/1301.3557 (2013).