USING METAHEURISTICS FOR HYPER-PARAMETER OPTIMIZATION OF CONVOLUTIONAL NEURAL NETWORKS

Victoria Bibaeva

Hamburg University of Applied Sciences (HAW Hamburg), Germany

ABSTRACT

Convolutional neural networks (CNNs) have attracted researchers' increasing attention for almost three decades now, achieving superior results in such domains as computer vision, signal processing etc. Their success can be mainly attributed to a specific network architecture, which is conceived by assigning values to a large number of hyper-parameters, each influencing the resulting error rate. Yet a search for good hyper-parameter values is a challenging task, being usually done manually and taking a considerable amount of work. This paper is dedicated to the problem of designing automated hyper-parameter search algorithms for convolutional architectures. We propose two algorithms based on such metaheuristics as evolutionary computation and local search. To our knowledge, they have never been applied to the case of CNN architectures before. Using image recognition datasets, we compare the algorithms and show that they can produce CNNs with nearly state of the art performance without any user interference, saving much tedious effort.

Index Terms— hyper-parameter search, convolutional neural networks, CNN, metaheuristics, memetic algorithm, hybrid genetic algorithm, simulated annealing

1. INTRODUCTION

Recent years brought an immense progress into the scientific area of deep learning, influenced by such factors as constant growth in hardware capacity and easy access to various large datasets for many kinds of real-world problems. As a result, huge advances of deep learning models and corresponding learning techniques can be observed [1]. One of the most popular deep learning models is convolutional neural network (**CNN**, [2]), which classifies the given data using its many neuron layers to extract feature hierarchy out of images, speech recordings, videos etc. The accuracy of a trained CNN can be affected through variating the training parameters (learning rate, number of epochs, batch size etc.), enhancing the dataset (augmentation, preprocessing) or tuning its architecture parameters. In this paper, we will call the latter **hyper-parameters**. Typical hyper-parameters are for instance number of layers, filter size, type of activation function and so on (see Table 1). Despite all available knowledge about CNNs, the impact of hyper-parameters on accuracy is discovered mostly empirically [3, 4] rather than theoretically, and their complex mutual interaction continues to be unknown.

Facing a task to train a CNN for a given dataset, it is advisable not only to try one model, but to tune its hyper-parameters afterwards in order to improve the original accuracy. However, the number of all available hyper-parameter combinations will grow exponentially with increasing network size. Searching through all of these combinations is practical only for small CNN models and short training cycles, otherwise an intelligent search strategy must be considered. Furthermore, even if the search strategy is employed, the elaborate manual search is prone to overlook some good combinations because of the sheer number of possible values for every hyper-parameter. Thus, an algorithm is needed to search for good hyper-parameter values. It should fulfill the described requirements by providing an efficient search strategy, handling a large number of hyper-parameters as well as automatically managing the hyper-parameter combinations.

This paper is dedicated to designing such an algorithm for CNNs. Our goal is to encourage researches to improve their deep learning models by automated hyper-parameter tuning, rather than tediously searching through all their variations. We propose two such algorithms based on popular families of metaheuristics – evolutionary computation and local search. They have already been successfully applied to smaller architectures like multilayer perceptron (**MLP**), but to our knowledge never to CNNs. Therefore, we demonstrate how to apply these algorithms to CNNs and use them for image recognition. Our experiments show that the proposed algorithms are able to improve the accuracy almost to the state of the art level, also being superior to currently used hyper-parameter search methods. Moreover, we compare the algorithms with respect to their complexity and the quality of generated architectures and discuss their advantages and drawbacks.

Let us assume that we are looking for a CNN model to solve the task of image recognition on a given dataset. The dataset itself can contain images of hand-written digits [2], natural objects like animals or vehicles [5], human faces [6] or even medical images of brain activity [7]. The goal is to find a model with the highest quality, i.e. a combination of loss and

978-1-5386-5477-4/18/$31.00 ©2018 IEEE

Layer Type	**Filter Bank**	**Pooling**	**Full Conn.**
Activation Function	ReLU ELU Maxout ELU + Maxout	–	ReLU ELU Maxout ELU + Maxout
Pooling Type	–	Max Avg Max + Avg	–
Filter Size	3, 5, 7, 9, 11	2, 3	–
Step Size	1, 2, 3, 4, 5	1, 2	–
No.Outputs	x	–	x
References	[9, 4, 10]	[5, 4, 10]	[5, 3, 4]

Table 1. The values of hyper-parameters used in this paper.

accuracy, the latter being more important as an indicator of correct class predictions in the future (see [2] for the related discussion). Each CNN architecture can then be represented as a point in the coordinate space of all its hyper-parameters. By eliminating randomizations and fixing training parameters and type of applied image preprocessing we can ensure that each architecture can be mapped to a single CNN, which performs deterministically over time. We further assume that the said coordinate space is finite, if we take into account only finite number of different values for every hyper-parameter (as in Table 1). Thus, there exists at least one "*optimal*" CNN model with the highest quality. Its architecture can be found by iterating over all points in the space and evaluating them.

Hence, the problem of finding an optimal CNN architecture can be considered as a combinatorial optimization problem, its solutions being generally divided into exact methods and heuristics. While exact methods are too expensive in computation, incorporating all the existing knowledge about neural networks into one heuristic is very hard. Instead, another approach is preferred in the machine learning community – **metaheuristics**. They are designed to solve a wide range of combinatorial optimization problems without adjusting themselves to each problem instance [8]. Representative metaheuristics are Simulated Annealing, Local Search, Particle Swarm Optimization, Evolutionary Algorithms etc. [8]. Important property of metaheuristics is keeping balance between *exploration* of the search space (i.e. identifying promising regions) and its *exploitation* (intensifying the search in one interesting region) [8]. Accordingly, the metaheuristics differ as to how they implement these two search strategies by setting their configuration parameters (which we call **meta-parameters**). Due to "No Free Lunch" theorems, there is no metaheuristic that solves all the optimization problem instances better than others [8]. Instead, each instance should be solved independently to identify the best metaheuristic.

2. RELATED WORK

Hyper-parameter search has always been a demanding process for machine learning practitioners, traditionally solved by Grid Search and Random Search [11]. While more sophisticated algorithms like Bayesian Optimization were proposed [12], and despite the superiority of Random Search [11], Grid Search continues to be very popular [7]. Yet if the number of hyper-parameters grows, such methods become impractical. As stated in [1, 11], more research is needed to find new approaches suitable for deeper models. The obvious way to find these is to investigate the related models like MLP, which are already successfully tuned with evolutionary algorithms.

Solving hyper-parameter search problem for deep learning architectures with evolutionary algorithms is, in fact, an emerging research field. Recent attempts are described in [13, 14, 15], for comprehensible overview see [14]. The mentioned work differs from ours, on the one hand, in the encoding of CNN architecture. The authors of [13] use much less hyper-parameters, [15] evolve only one layer at a time instead of the whole architecture in order to create non-linear network structures. Also, studies [14, 15] evolve network weights together with network topology (hyper-parameters), or weights only [16]. On the other hand, these authors apply very specialized versions of population-based algorithms like Genetic Algorithm (**GA**), whereas we use distinct, straightforward metaheuristics (Memetic Algorithm and Simulated Annealing) without specific alterations, as found in the Operations Research literature. Our focus is not in utilizing all kinds of learning and preprocessing techniques to beat the state of the art accuracy on any given dataset, but rather in discovering general advantages of metaheuristics for hyper-parameter search. Our results show that the proposed metaheuristics are highly applicable to the described problem and produce substantial increase in accuracy. It proves that metaheuristics can be used "as is" with an appropriate encoding, without much alterations to any special architecture type.

3. OUR APPROACH

After analyzing the given problem setting we now introduce our solution to find the best CNN architecture for any task at hand. We identified that applying metaheuristics to solve this problem requires implementing three major mechanisms. The first mechanism is the appropriate representation of input data, i.e. the **encoding** of CNN architectures. We propose binary encoding of the entire architecture, where each layer has its own hyper-parameters encoded according to Table 1. All layers have the same constant length, which is calculated automatically depending on layer count and input image size.

The second mechanism is the ability to validate every solution candidate produced by the metaheuristics. It is required because, unlike simpler topologies such as MLP, CNN architectures should adhere to certain constraints. This implies examining that each new CNN architecture complies with the set of common architecture rules. For example, its filter size should be odd (in filter bank layers), step size not exceeding filter size, number of outputs should not decrease during the feature extraction steps. This **set of rules** must be based on the domain-specific knowledge about CNNs, encompassing not only the reasonable values for hyper-parameters, but also their known interactions and the correct layer sequence (e.g. no filter bank or pooling after full connection layer).

The third mechanism is the **objective function** Q_f of the metaheuristics (also called fitness function). It should entail the concept of CNN model quality, combining both loss Q_l and accuracy Q_a: $Q_f = \alpha \cdot Q_l + \beta \cdot Q_a$, where $|\alpha| + \beta = 1$. As mentioned above, accuracy is the more important measurement of model quality, hence $|\alpha| < \beta$. Also note that high loss has negative impact on model quality ($\alpha \leq 0$). Q_f was chosen to be the same for all algorithms to provide unbiased comparison. In our experiments, the best architectures were achieved with $\alpha = -0.25$ and $\beta = 0.75$. Using $\alpha = 0$ as in [13, 16] led to architectures with higher loss being favored by the algorithms, while CNNs with similar accuracy and lower loss were discarded. After taking all these considerations into account, we proceed to describe the proposed algorithms.

3.1. Memetic Algorithm

Memetic Algorithms (**MA**s) are the combination of Evolutionary Algorithms and Local Search (**LS**), also known as hybrid GAs [6]. MA is inspired by natural evolution in a population as well as local refinement of its individuals [17]. As in GA, individuals with higher *fitness* are more likely to survive into the next generation, where they are subjected to LS, and then to standard genetic operators of GA – see Algorithm 3.1. The hybridization of GA with LS is motivated by the fact, that pure GA is not well suited for fine-grained search in complex combinatorial spaces [17]. Therefore, LS is utilized in MA to close this gap and increase the search efficiency.

```
Memetic Algorithm
1  Create initial population of $M$ random individuals
2  Evaluate the fitness of each individual
3  *LS*: The fittest of $S$ neighbors taken within radius $R$
   of each individual replaces it in the current population
4  Apply genetic operators to the current population:
     a  *Selection*: The fraction $p_s$ of fittest individuals
        survive to the next generation, rest is discarded
     b  *Crossover*: Random pair of individuals produces one offspring, until $M$ is reached
     c  *Mutation*: Alter a random gene in the fraction
        $p_m$ of the children
5  Repeat 2 – 4 until iteration count $N$ is reached
```

There are many variants of genetic operators in the literature [8], their choice being crucial to ensure the effective search strategy of MA. We used elitist variants of selection and mutation to keep the best solution quality through the generations. Also, 1-point-crossover was implemented, which makes a cut in both parent chromosomes at an arbitrary

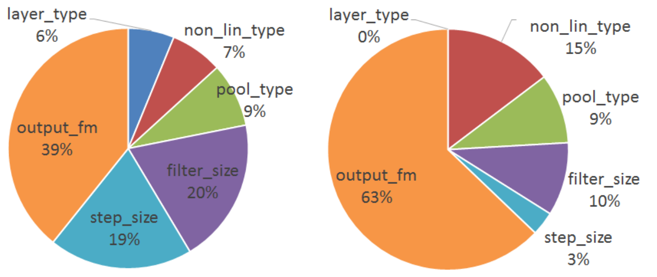

Fig. 1. MA: Probability of changing each hyper-parameter of one layer during crossover (left) and mutation (right).

position, so that they can swap the parts to generate one child. We considered that making more than one cut or randomly swapping single genes would not produce valid offsprings of good quality, because it would put certain hyper-parameters out of their context, where they had previously worked well.

Binary encoding of a CNN architecture brings up the question, whether the chosen genetic operators are feasible, i.e. able to produce a valid individual. The statistics about how often each hyper-parameter was changed by crossover and mutation are shown in Fig. 1. The distribution of probabilities to change any hyper-parameter with crossover is virtually the same as the distribution of hyper-parameters within a chromosome. Thus, crossover is observed to change every hyper-parameter equally likely. It makes the operator feasible, as any pair of chromosomes can be cut at virtually any position and still produce a valid child. In contrast, the distribution of mutated hyper-parameters diverges from its counterpart. E.g. the probability to change the number of outputs ("output_fm") or the activation function ("non_lin_type") is much higher, whereas it is less likely to change filter or step size. This is also plausible, because the latter hyper-parameters are more constrained according to our set of rules.

The advantages of MA are the higher quality of its population, which consists of local optima and speeds up the convergence, and lower sensitivity with respect to initialization. Otherwise, MA has cubic complexity and therefore requires more solution evaluations (i.e. CNN training cycles) than GA in order to converge. In our setting, it has 6 meta-parameters (N, M, S, R, p_s, p_m) to be appropriately set beforehand.

3.2. Simulated Annealing

The second proposed algorithm is a single-solution based metaheuristic named Simulated Annealing (**SA**). SA solves optimization problems by introducing temperature T, a parameter that controls the search [8]. SA can be viewed as an extension of LS, requiring a predefined neighborhood structure in the search space [18]. We made use of binary encoding of individuals needed for MA to define neighborhood as follows. Let Ω^* denote the search space of all binary encoded valid CNN architectures with length l. Then, the neighborhood $\mathcal{N}(A, R)$ with radius R of an architecture $A \in \Omega^*$ is expressed with help of bitwise XOR operator $\oplus$ as:

$$\mathcal{N}(A, R) = \left\{ B \in \Omega^* \middle| d(A, B) = \frac{\text{bitsum}(A \oplus B)}{l} \leq R \right\}$$

SA has linear complexity, as it starts with one initial solution and searches for better solutions within its neighborhood. In contrast to LS, SA can also accept worse solutions with some probability, which is diminishing with T [19]. The goal is to maximize the objective function f – see Algorithm 3.2.

The success of SA depends on how the acceptance probability and the cooling of temperature are calculated. The latter is confirmed to be crucial for convergence [18], being also bound to the iteration count N. The ideal cooling schedule guarantees convergence, but is very slow (for references and formula see [18, 8]); hence, faster schedules are preferred:

$$T_i = \theta^i \cdot T_0 \text{ with } 0.8 \leq \theta \leq 0.99$$

We tried to approximate the ideal cooling schedule as much as possible using different values of T_0 and θ in the equation above, finally setting $N = 100$, $T_0 = 2$ and $\theta = 0.9$.

As for the acceptance probability of a worse candidate solution, it is commonly calculated as below, with $d = 1$ [18]:

$$P(T_i, f(s), f(s')) = exp\left(\frac{f(s') - f(s)}{d \cdot T_i}\right)$$

However, for our problem setting it was necessary to scale the probability due to prior choice of cooling schedule in order to obtain the desired search behavior. According to the resulting formula, small fitness decrease will be more likely tolerated than big one, in the beginning rather than in the end.

Ultimately, SA has 5 meta-parameters to be carefully chosen in order to ensure convergence and effective search: N, R, d, T_0, θ. Three of these were set as mentioned above, while the process of finding good values of the remaining R and d will be discussed in the next section. These two parameters are responsible for shifting the search direction and, as a result, for exploration behavior in the beginning of SA. The exploitation properties of SA are owed to the nature of LS.

```
Simulated Annealing
1  Specify initial temperature $T_0$ and random solution $s$
2  At iteration $i$, evaluate $f(s)$
3  Choose random neighbor $s' \in \mathcal{N}(s, R)$, evaluate it
4  Acceptance criterion: if $f(s') \geq f(s)$ then set $s := s'$, else set $s := s'$ with probability $P(T_i, f(s), f(s'))$
5  Cooling schedule: decrease temperature $T_i$
6  Repeat 2 – 5 until iteration count $N$ is reached
```

The benefits of SA are its simple implementation, absence of genetic operators (defining of which can be demanding, ineffective or even impossible for some problem instances) and linear complexity. On the other hand, SA is prone to perform badly if the search space has many local optima, as the algorithm is highly dependent on initial solution [8]. Yet given the opportunity to acquire a good initial solution, SA can quickly search its wider neighborhood to find a better one.

4. EXPERIMENTAL RESULTS

In order to evaluate the performance of the proposed algorithms, we implemented them using Python and `caffe` (http://caffe.berkeleyvision.org). We also implemented Random Search (**RS**) and GA with our own binary encoding. The only meta-parameter of RS (N) was set as in SA, meta-parameters of GA (N, M, p_s, p_m) were taken from MA. All algorithms were tested on two popular image recognition datasets – MNIST and CIFAR-10 [9]. The only dataset preprocessing we considered was mean subtraction and normalization. Comparable state of the art accuracy on MNIST is 0.9905 [2], or 0.9947 with unsupervised learning [3]; on CIFAR-10 – 0.8487, achieved by a specific pooling type [20].

As our goal was to find the best CNN architecture, all parameters of CNN except hyper-parameters were set to fixed values, out of several reasons. Firstly, it reduced the search space to architecture-related parameters. Secondly, the chosen genetic operators would not be able to produce valid, high quality CNNs, e.g. if the good performing hyper-parameters of one CNN were to be recombined with other network's training parameters. Thirdly, we assume that to evolve one excellent architecture and to tune its training parameters afterwards is less time consuming, than evolving many mediocre architectures together with their training parameters. Thus, each CNN was trained with Stochastic Gradient Descent using standard values for batch size, momentum, weight decay etc., also applying dropout with $p = 0.5$ in the classifier (for values see [9], as they are out of this paper's scope). Random seeds were used to attain deterministic training results.

First of all, we experimented with meta-parameters of MA and SA to let them accomplish their full potential. As SA is highly dependent on neighborhood structure, its search behavior is influenced by radius R and acceptance probability coefficient d. For example, if $d = 1$ (typical value), then worse solution candidates are generally accepted until the end, see turquoise line in Fig. 2, left. This obviously does not improve the initial solution, see purple line of the same diagram. Very small values like $d = 0.001$ (Fig. 2, middle) imply little likelihood of accepting any worse candidates, eliminating the benefits of SA compared to pure LS. A good value turned out to be $d = 0.05$, see Fig. 3, left. As for neighborhood radius, small values like $R = 0.05$ allow only small steps in the search space, leading to small fitness increments, see Fig. 2, right. Larger radius $R = 0.5$ ensures wide variety of candidates; yet most of these are invalid, making the selection of valid ones very time consuming. Thus, we chose $R = 0.15$ (see Fig. 3, left) to allow bigger steps in the search

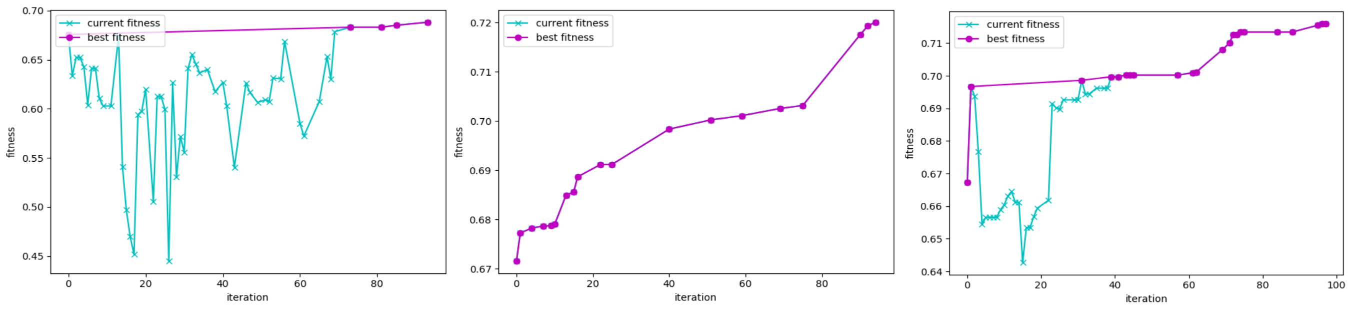

Fig. 2. SA: Development of current and best fitness for $d = 1$ (left), $d = 0.001$ (middle) and $R = 0.05$ (right).

space and to find valid neighbors easier. Accordingly, we set neighborhood radius of MA to $R = 0.15$. Furthermore, because MA demanded certain population diversity, we fixed $N = 10$, $M = 30$ and $S = 5$, so that MA evaluated 1500 CNNs in total. As for meta-parameters of genetic operators, we decided to use $p_s = 0.5$ and $p_m = 0.75$, because it caused more population diversity and higher fitness increase than combinations with less selection or mutation probability.

After determining meta-parameters, we compared MA and SA, measuring their performance as an ability to improve the initial fitness level. Unlike absolute results of the last iteration, which strongly depend on initial solution and other factors like batch normalization or learning method, relative fitness improvement reflects the performance objectively. Representative fitness curves of SA and MA are shown in Fig. 3, left and middle. In the beginning of SA, worse candidates are accepted, bringing the search to further areas, ultimately allowing significant fitness increase. Yet only small fitness changes can be observed at last iterations. In turn, MA achieves strictly monotonic increase of fitness due to substituting all individuals through their better neighbors at each iteration. Most notably, MA always starts with at least one excellent individual in the population delivering stable fitness increase, whereas SA's initial solution shows much more variations, and its fitness increase reaches from 1% for excellent initial solutions up to 32% for bad ones. SA also achieved no fitness increase as it was initialized with the best known solution of MA with accuracy 0.9902. Best accuracy of SA was 0.9904 with 5 times as many training iterations as usual, taking as much time as MA (4 days) and proving that our results can be surpassed using other training parameters.

Our final trials were dedicated to a fair comparison of the proposed algorithms with RS and GA, where all algorithms started with the same random initial solution or population. The mean results of these trails are illustrated in Fig. 3 right. Unsurprisingly, the weakest performance was observed of RS, as it has no exploitation properties and can reach good solutions only by chance. GA is second to last, often displaying fitness plateaus and having little exploitation. The proposed metaheuristics were able to beat both of them: SA achieved 8.4% fitness increase having the same linear complexity as RS, and MA was the best with 14.8% – a double fitness increase of GA. The experiments with CIFAR-10 demonstrated similar result, except that it required changing the set of rules for CNN architectures, because all CNNs with Max Pooling or Maxout performed badly (cf. [20]). The average fitness increase of SA on CIFAR-10 was 8.9%, of MA – 15.6%.

5. SUMMARY

The problem of finding optimal values for hyper-parameters in CNN architectures suffers from the curse of dimensionality, as the networks grow deeper. We viewed this problem as a combinatorial optimization problem and identified two metaheuristics as its solutions. The first one is MA, which combines the advantages of GA and LS; the second is SA, also

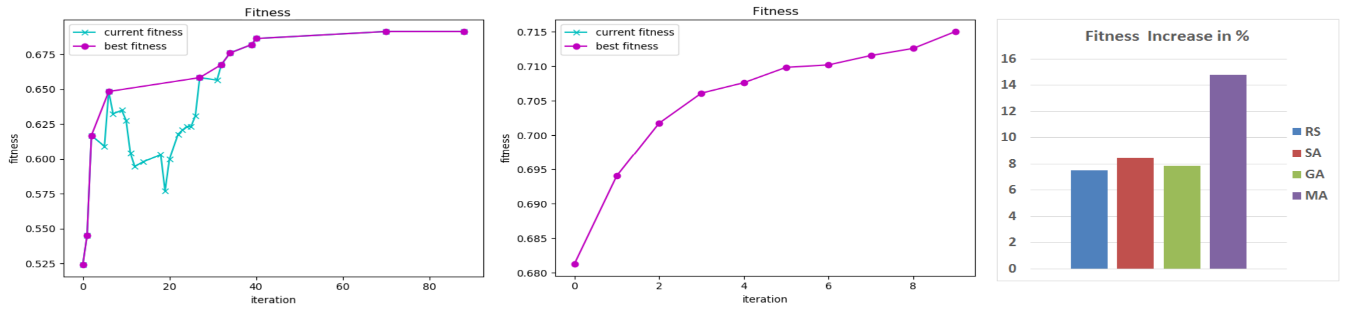

Fig. 3. Best fitness development in SA (left) and MA (middle); performance comparison of all implemented algorithms (right).

based on LS. To our knowledge, they have not been applied to CNN architectures before. In this paper, we successfully applied them to the described problem instance. Our experiments with image recognition datasets demonstrate that the proposed algorithms can surpass the current state of the art methods for hyper-parameter optimization like RS and GA, also being able to reach accuracy comparable to state of the art on these datasets. The knowledge about valid CNN architectures is incorporated in metaheuristics by the choice of architecture encoding, and is therefore exchangeable, without the need to alter the algorithms themselves. This makes them applicable to wide variety of datasets or even other deep learning topologies. To reach their full potential, MA and SA can be easily tuned for each task as illustrated above, while involving considerably less parameters than CNNs. We assessed their ability to increase the objective function and discovered, that MA performs nearly twice as good as SA, having also thrice its complexity. In turn, SA is easier to implement and does not require genetic operators, despite producing more unstable results that depend highly on initialization.

Both MA and SA are shown to be very promising algorithms, which can be further improved by using parallel computation (as in [14]) and the latest advances in Operations Research. Also, different kinds of data preprocessing and augmentation [20] or utilizing other modern learning techniques can additionally raise the resulting accuracy level. Concluding our paper, the proposed algorithms contribute to the existing variety of the hyper-parameter search methods and should be integrated into common deep learning frameworks.

6. REFERENCES

[1] Li Deng and Dong Yu, "Deep learning: Methods and applications," *Foundations and Trends in Signal Processing*, vol. 7, no. 34, pp. 197–387, 2014.

[2] Y. LeCun et al., "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[3] K. Jarrett et al., "What is the best multi-stage architecture for object recognition?," in *Computer Vision, IEEE 12th Intl. Conference on*, 2009, pp. 2146–2153.

[4] D. Mishkin et al., "Systematic evaluation of CNN advances on imagenet," *CoRR*, vol. abs/1606.02228, 2016.

[5] A. Krizhevsky et al., "Imagenet classification with deep convolutional neural networks," in *Adv. in Neural Information Processing Systems 25*, F. Pereira et al., Eds., pp. 1097–1105. 2012.

[6] Y. Chang et al., "Feature selection for improved automatic gender classification," in *2011 IEEE Workshop on Computational Intelligence in Biometrics and Identity Management (CIBIM)*, April 2011, pp. 29–35.

[7] W. Yan et al., "Discriminating schizophrenia from normal controls using resting state functional network connectivity: A deep neural network and layer-wise relevance propagation method," in *2017 IEEE 27th Intl. Workshop on Machine Learning for Signal Processing (MLSP)*, Sept 2017, pp. 1–6.

[8] I. Boussaïd et al., "A survey on optimization metaheuristics," *Information Sciences*, vol. 237, pp. 82 – 117, 2013, Pred., Control and Diagnosis using Adv. Neur. Comp.

[9] G. E. Hinton et al., "Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, vol. abs/1207.0580, 2012.

[10] P. Sermanet et al., "Overfeat: Integrated recognition, localization and detection using convolutional networks," in *Intl. Conference on Learning Representations (ICLR 2014)*. CBLS, 2013, p. 16.

[11] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 281–305, Feb. 2012.

[12] J. Snoek et al., "Scalable bayesian optimization using deep neural networks," in *Proc. of 32nd Intl. Conference on Machine Learning - Vol. 37*, 2015, pp. 2171–2180.

[13] S. R. Young et al., "Optimizing deep learning hyper-parameters through an evolutionary algorithm," in *Proc. of the Workshop on Machine Learning in High-Performance Computing Environments*. 2015, MLHPC '15, pp. 4:1–4:5, ACM.

[14] M. Jaderberg et al., "Population based training of neural networks," *CoRR*, vol. abs/1711.09846, 2017.

[15] R. Miikkulainen et al., "Evolving deep neural networks," *CoRR*, vol. abs/1703.00548, 2017.

[16] L.M. Rere et al., "Metaheuristic algorithms for convolution neural network," *Computational Intelligence and Neuroscience*, vol. 2016, May 2016.

[17] N. Krasnogor and J. Smith, "A tutorial for competent memetic algorithms: model, taxonomy, and design issues," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 5, pp. 474–488, Oct 2005.

[18] L. Bianchi et al., "A survey on metaheuristics for stochastic combinatorial optimization," *Natural Computing*, vol. 8, no. 2, pp. 239–287, 2009.

[19] J. S. Sartakhti et al., "Simulated annealing least squares twin support vector machine (sa-lstsvm) for pattern classification," *Soft Computing*, pp. 1–13, 2016.

[20] M. D. Zeiler and R. Fergus, "Stochastic pooling for regularization of deep convolutional neural networks," *CoRR*, vol. abs/1301.3557, 2013.