Herausgeber et al. (Hrsg.): SKILL 2018,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2018 11

Hyper-Parameter Search for Convolutional Neural Networks – An Evolutionary Approach

Victoria Bibaeva[1]

Abstract:

Convolutional neural networks is one of the most popular neural network classes within the deep learning research area. Due to their specific architecture they are widely used to solve such challenging tasks as image and speech recognition, video analysis etc. The architecture itself is defined by a number of (hyper-)parameters that have major impact on the recognition rate. Although much significant progress has been made to improve the performance of convolutional networks, the typical hyper-parameter search is done manually, taking therefore a long time and likely to disregard some very good values. This paper solves the problem by proposing two different evolutionary algorithms for automated hyper-parameter search in convolutional architectures. It will be shown that in case of image recognition these algorithms are capable of finding architectures with nearly state of the art performance automatically, sparing the scientists from much tedious effort.

Keywords: deep learning; convolutional neural networks; CNN; hyper-parameter search; evolutionary algorithms; genetic algorithm; memetic algorithm

1 Introduction

Recent decade brought an enormous scientific progress in the field of deep learning, which deals with deep, many-layered neural networks and their learning techniques. One of the most prevalent classes of such networks is Convolutional Neural Networks (**CNNs**). A CNN is a specific type of multi-layer perceptron (**MLP**) with more complex architecture. It was first proposed in 1989 as a model inspired by visual cortex of mammals, and was used to classify the images of hand-written digits [Le98]. In 2012 came a great breakthrough, as a CNN suggested by [KSH12] won the world's biggest object recognition challenge ILSVRC. It successfully classified natural objects in a dataset containing of 1.4 million images and 1000 categories. Since then, CNNs quickly became state of the art and were applied to such tasks as image and speech recognition, object classification and video analysis. Nowadays they are ubiquitous due to their outstanding performance and well-developed techniques of its improvement, in some cases already reaching the error rate of humans [KSH12]. However, the growing complexity of CNN architectures causes many problems, an important

[1] Hochschule für Angewandte Wissenschaften Hamburg (HAW Hamburg), Department Informatik, Berliner Tor 5, 20099 Hamburg, Germany firstname.lastname@haw-hamburg.de

one being "*overfitting*". It happens when the network memorizes the dataset, but is not able to recognize slightly different objects outside of the dataset.

The performance of CNNs on an object classification task can be defined as recognition rate, i.e. percentage of correctly classified images. It can be influenced through the choice of dataset, but also through changing the training parameters (like learning rate of gradient descent) or carefully tuning the network architecture (represented by all the corresponding **hyper-parameters**). Whereas the first two influence factors are generally well-studied [Hi12], the architecture tuning is typically carried out manually in a haphazard manner, considering merely a small number of proven values and techniques. Only few scientific studies are dedicated to the empirical impact of hyper-parameters (cf. [Ja09, MSM16]), while their mutual influence remains largely unknown. Thus, there is still a lack of knowledge concerning reasons behind a good performance of CNNs and its further improvements.

A sensible approach to design a CNN to solve any given task is to try out not only one, but many different hyper-parameter combinations. This is likely to result in a model with higher recognition rate. Yet the manual search for good hyper-parameter values can be very time and resource consuming, not to mention the chance of overseeing some promising values. An automated hyper-parameter search, on the other hand, has a potential to save a lot of tedious effort using some intelligent search strategy. Our work originated from an idea to implement such an algorithm using the up-to-date knowledge about CNNs. Two algorithms presented in this paper are based on evolutionary computation and have already shown their worth in finding good architectures of MLP. It will be demonstrated that they can be successfully adjusted to the case of CNNs, providing better CNN architectures than a baseline method and achieving nearly state of the art performance without any user intervention. By means of some representative datasets from the image recognition domain we will determine, which algorithm gives better results under certain conditions. Such criteria as architecture quality, complexity and runtime will be evaluated as well.

Our paper is organized as follows. First, we will introduce different hyper-parameters of a CNN and their possible values. Then, in section 3, the current methods of hyper-parameter search will be reported in case of CNNs as well as MLP. Section 4 gives detailed account on the two proposed evolutional algorithms. At last, the major results of our experiments are presented in section 5, leaving the last section for conclusion and future work.

2 Convolutional Neural Networks

In order to classify objects a CNN receives an input image and extracts certain visual features from it. The extraction is done step by step, so that the features extracted during the succeeding layers (rectangles, circles) are composed of features from the preceding layers (lines, angles, arcs). Thus, a feature hierarchy is gradually constructed [Le98]. Therefore a CNN contains of several so-called **feature extraction stages**, each of which is composed of three basic types of neuron layers [Ja09]: *filter bank layer*, *non-linearity layer*, *feature*

pooling layer. After the feature extraction stages follows a **classifier**, which is essentially a MLP with some *full connection layers* that calculates the probability of the object class. A famous instance of a CNN architecture is shown in Fig. 1. It consists of two feature extraction stages (C1 + S2, C3 + S4) and a three-layered classifier (C5 + F6 + Output).

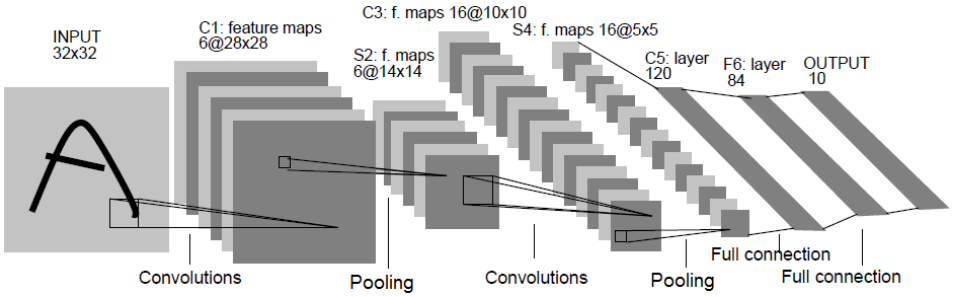

Fig. 1: CNN architecture [Le98].

So how can we generate variations of any given CNN architecture? In other words, what are the tunable hyper-parameters? These are, on the one hand, the number of feature extraction stages and their specific components (layer types and their sequence), but also the number of hidden layers in the classifier. The rest of this section will be dedicated to the description of different layer types and the corresponding hyper-parameters.

Filter Bank Layer This layer type is where the convolution operation takes place, to which CNNs owe their name [Ja09]. Convolution produces a mapping between a source image and target image by applying a convolutional kernel (also named "*filter*") onto overlapping regions of the source image. Thus, a filter highlights one certain feature in the entire image. Unlike MLP, the neurons within one filter bank layer are divided into several planes. All neurons in one plane share weights and perform the same convolution operation in order to extract one certain image feature. Classifying more complex objects requires more features and therefore more filters in one layer – hence the term "filter bank". Filter bank layer has 3 major hyper-parameters, namely: filter size, step size (to define the overlapping image regions) and filter count. Fortunately, not many values of these hyper-parameters have prevailed in common practice. Nevertheless, there are only rules of thumb to set the number of filters, depending on the expected number of features and object complexity [MSM16].

Non-linearity Layer Traditionally, the output of filter bank layer is then fed to non-linearity layer, which is often considered to be a part of the former [Ja09]. As in a common neuron, a non-linear activation function is applied here to each pixel value, in order to propagate or suppress the incoming visual signal. A very popular function is $g(x) = max(0, x)$, and the neurons using it are named ReLU ("rectified linear units", [KSH12, Hi12]). Nonetheless, a variety of other functions based on ReLU were presented in recent years. The study [MSM16] compared these functions within one particular CNN architecture being trained

on ILSVRC dataset. The lowest error rates were achieved by ELU ("exponential linear unit"), maxout and their combination (see [MSM16] for references). Accordingly, we have chosen these 4 functions for our trials, adding one hyper-parameter to a current layer type.

Feature Pooling Layer The last layer type of a feature extraction stage reduces the source image resolution even further, making the outcome independent of the exact feature position. It is done by applying the same filter onto the overlapping regions of the image. In contrast to filter bank layer, this filter is not learned during training, but calculates maximal or average pixel value of every region [Hi12]. Therefore, the layer is called max or average pooling layer, accordingly. The resulting target images have lower dimensions, and so a part of visual information is lost (proportionally to filter size). The choice of an appropriate pooling type (max or average) is controversial and cannot be answered in general for all the CNN architectures or datasets. Furthermore, there is evidence, that the sum of max and average pooling can be even more effective [MSM16]. As a result we consider 3 hyper-parameters of a pooling layer: filter size, step size and pooling type with 3 possible values.

Full Connection Layer As stated earlier, the classifier of a CNN aims to learn the relationship between the extracted features and the resulting object classes. It contains of several stacked full connection layers without weight sharing. This implies the biggest number of trainable weights in the whole network and the necessity to limit the count of such layers, typically using only 2 or 3 (cf. [KSH12, Ja09, MSM16]). Each full connection layer may be followed by a non-linearity layer, leading us to define 2 hyper-parameters: number of output neurons and type of activation function. We also used Dropout with probability 0.5 (see [Hi12]) after each full connection layer as a technique to avoid overfitting.

Hyper-Parameter Overview The correct order of layer types within one feature extraction stage has a very important role in designing a CNN. It was already studied in [Ja09], where the lowest error was achieved by CNNs with layer sequence "Filter Bank → Non-Linearity → Feature Pooling". Unsurprisingly, this layer sequence actually occurs in nature. All hyper-parameters introduced above and their possible values are summed up in Tab. 1. However, the interaction between the hyper-parameters remains largely unknown. The existing reports on this subject (eg. [Ja09, MSM16]) used a single dataset for evaluation and noted that their results cannot be automatically transferred to other datasets.

3 Related Work

The goal of object classification is to design a model, in our case a CNN, which after training *generalizes* the given dataset well. It means, the model can correctly classify images that were not included in the training set. For this purpose, a test set is employed, which is also a

Hyper-Parameters						Ref.
Filter Bank Layer	Activation Function (ReLU, ELU, Maxout, ELU + Maxout)	–	Filter Size (3, 5, 7, 9, 11)	Step Size (1, 2, 3, 4, 5)	No. of outputs	[Hi12]
Feature Pooling Layer	–	Pooling (Max, Average, Max + Average)	Filter Size (2, 3)	Step Size (1, 2)	–	[KSH12]
Full Connection Layer	Activation Function (ReLU, ELU, Maxout, ELU + Maxout)	–	–	–	No. of outputs	

Tab. 1: Hyper-parameters used in this paper and their values.

part of the given dataset, but is not generally used for training. So the quality of our model can be summed up as **accuracy** on the test set, and we are aiming to find a model with the highest accuracy (percentage of correct classifications). On the other hand, the objective function of gradient descent during training is the so-called **loss** function which measures error on training set. Both loss and accuracy indicate how well the given model performs, accuracy being more important criterion for the future productive use of the model [Le98]. Other quality criteria such as training time, computational complexity or memory usage are of less importance, because they depend on the chosen network implementation tool.

Previous section illustrated the fact that there is an enormous number of different CNNs applicable for any given dataset. It makes training and testing them all quite impossible, considering that each training cycle might take several days or weeks. Even on the modern hardware this task can be insurmountable, leading to the necessity of using an automated hyper-parameter search algorithm. Its benefits would be sparing time and resources due to an intelligent search strategy and minimizing the risk of overseeing promising hyper-parameter values. On the other hand, it should be able to leverage the existing knowledge about CNNs to avoid certain pitfalls and produce models with above-average quality. The real challenge is to find an algorithm which quickly and reliably finds a competitive architecture.

Surprisingly, relatively little scientific research is done to solve this problem yet, see [Be11, BB12, Sn15]. These studies were motivated by the fact that the advances in object classification can be achieved through hyper-parameter tuning, rather than inventing new models or training techniques. In the authors' opinion, the hyper-parameter search is nowadays "more of an art than a science", even though it should belong to the designing process of each deep learning model [Be11]. Thus, they proposed Grid Search and Random Search as an alternative to manual hyper-parameter exploration.

If the desired values of hyper-parameters are known in advance, then all their conceivable combinations can be easily generated, essentially defining all the points in the initial search

space. **Grid Search** [BB12] reduces the overall search space by selecting a few values of each hyper-parameter. If combined with each other, these selected values create a grid within the search space. Subsequently, grid points are considered as solution candidates, and the corresponding architectures are tested to identify the model with the best accuracy. Advantages of Grid Search are simple implementation and a possibility of parallel processing.

As opposed to Grid Search, **Random Search** selects some random combinations of hyper-parameters. These are basically random points in the search space which may get closer to the better solutions than grid points. The experiments in [BB12] confirm that Random Search requires less candidates than its counterpart in order to reach certain accuracy level. However, the probability to randomly encounter a very good solution is inverse proportionate to spatial dimensions. Furthermore, Random Search is not capable of pursuing any promising directions or selectively explore certain areas. Hence, both Random and Grid Search contain an inherent tendency to step over the best solutions.

The problem of finding the best hyper-parameters is far from being solved [BB12]. In spite of the availability of such methods as Bayesian Optimization [Sn15], Grid and Random Search remain the tools of choice. Thus, there is still a need to exploit new algorithms from other domains, including less complex networks like MLP, where hyper-parameter optimization has been in research focus for a long time. The most widely used methods to find the best MLP architectures are evolutionary algorithms [CG11, OI11]. Consequently, our approach to solve the hyper-parameter search problem is based on evolutionary algorithms and the assumption that they can be adjusted to the case of CNN architectures. Two such algorithms will be presented in the following section: both are well-studied, straightforward to implement and provide an efficient search strategy that avoids local optima.

4 Proposed Algorithms

4.1 Genetic Algorithm

Genetic Algorithm (**GA**) was introduced in the 1970s [Ch11] and is since then the most well-known evolutionary algorithm. It is population-based and attempts to replicate the processes of natural evolution. Accordingly, the population individuals with higher fitness, i.e. ability to adapt to the environment, have more chances to pass their best characteristics to the next generation, whereas the unfavorable characteristics rather disappear [CG11].

A lot of articles have been published concerning hyper-parameter search for MLP using GA [OI11]. Generally, GA works as follows. Firstly, MLP architectures should be represented with binary strings in order to serve as individuals in the population. Each hyper-parameter value is expressed through one or more genes (0 or 1), so the MLP architecture can be transformed into a *chromosome*. Every iteration of GA changes the current population of chromosomes/individuals by applying genetic operators to them: selection, crossover and mutation. To measure the quality of individuals a fitness function is used, which in case of

MLP architectures can depend on training loss, test accuracy or network size [OI11]. The fitness function is subjected to maximization during the GA iterations.

Adjusting GA to the case of CNN architectures, we have chosen the following version of it:

1. Create the initial population with M random individuals
2. Evaluate the fitness of each individual
3. Apply genetic operators to the current population:
   a) Selection: The fraction p_c of the fittest individuals survive to the next generation, the rest is discarded
   b) Crossover: random pair of individuals produces one offspring by swapping some genes, until the population size is reached
   c) Mutation: p_m random genes of some individuals will be altered
4. Repeat steps 2 – 3 until the required iteration count N is reached

Genetic operators should be chosen to accomplish an efficient search strategy. Thus, **selection** attempts to enhance the average quality of the population and lead the search in the direction of promising solutions. The chosen variant of selection is based on elitism [CG11], which means that the fittest individuals should survive at any rate, not just by a given probability or ranking. It guarantees a non-decreasing fitness and genetic diversity of the population.

Crossover, on the other hand, brings variation into the population by combining good properties of selected individuals and facilitates faster convergence. The variant of crossover we used is called 1-point-crossover, as it cuts the parent chromosomes in a random position and swaps the resulting parts between the parents to produce a child chromosome [CG11]. It ensures mostly feasible and good offspring architectures, which accelerates the search and does not tear the specific CNN architecture layers completely out of their context.

The basic idea of **mutation** is to avoid local optima, intensify the population diversity and acquire new genetic material. Mutation is also affected through elitism, as the fittest individuals (selected parent chromosomes) should be excluded from changing genes [CG11].

The challenges of GA are the choice of suitable genetic operators, fitness function and binary representation of CNN architecture. Also, GA obviously contains its own specific parameters such as probability of selection and mutation, as well as population size and number of iterations. Nevertheless, the advantages of GA make it a popular tool to explore the architecture search space [Ya99]. Firstly, GA is capable to search globally, avoid local optima and guarantee the sequential fitness improvement. Secondly, it can generate architectures with the desired characteristics, by means of including them into the fitness function. In the end, GA is less sensitive with respect to initialization than one-solution-based algorithms.

4.2 Memetic Algorithm

For some instances of search problems, the GA's efficiency was reported to be unsatisfactory [KS05]. The reason is that GA lacks mechanisms to perform fine-grained search in a region with very good solutions. As opposed to global search of GA, an algorithm called **Local Search** is designed to explore the region of its initial solution in order to find the better ones. Therefore, an obvious attempts were made to hybridize GA with some form of Local Search [Ch11]. One of the most successful hybrid algorithms is Memetic Algorithm (**MA**). Its name is based on the notion of "meme", which represents a unit of cultural evolution that can exhibit local refinement [KS05]. Thus, MA uses all the key components of GA, inserting Local Search step before applying genetic operators to the current population [Ch11]. Thereby the resulting population strictly consists of local optima.

Local Search within MA is usually carried out on a subset of the population. Some neighbors of every individual in this subset are then evaluated with fitness function. If one particular neighbor has higher fitness value than the original individual, then it replaces the latter in the population. Moreover, Local Search can be done in 2 different ways [Ch11]. Firstly, the search for the individual's replacement can be continued until the first fitter neighbor is encountered. Secondly, the search can iterate over a fixed number of random neighbors, so that the complexity of one MA iteration increases as a square function of population size.

Despite higher computational complexity MA has also a number of advantages, which can be beneficial for many real-life search problem instances [KS05]. On the one hand, it has only two more specific parameters than GA, namely the number of neighbors to evaluate and the radius of neighborhood. On the other hand, as the population consists of higher quality individuals, Local Search leads to quicker convergence of MA.

In order to fully exploit the benefits of hybridization we have chosen the second, greedy variant of Local Search for our experiments. Also, we executed Local Search on the whole population, not only on its subset. These choices are consistent with the earlier established elitism, because the fittest individuals can only be replaced with even fitter ones.

5 Evaluation

We implemented the proposed algorithms in Python, using open source framework `caffe` [2] for training of CNNs. Next, a series of experiments was conducted with intention to find the best configuration for each algorithm and to assess their solution quality. For this purpose we utilized 2 datasets from object classification domain, keeping reasonable training time in mind. These datasets were chosen to represent real-world classification problems and exemplify a large number of documented results. The first dataset named CIFAR-10 is composed of 60,000 colored images of natural objects scaled to 32×32 pixel. Its 10 classes

[2] `http://caffe.berkeleyvision.org`

are: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck (see [ZF13]). The other dataset MNIST includes 70,000 images of hand-written digits in gray scale [Le98].

As the only kind of image preprocessing we used in our work is normalization and subtracting the mean image of the entire dataset, it would be of interest to know state of the art accuracy of CNN models with the same preprocessing steps. For example, the authors of [ZF13] achieved an accuracy 0.8487 on CIFAR-10 using a specific type of pooling layer. The top score for MNIST has long been 0.9905, accomplished by the CNN architecture in Fig. 1, before it was beaten by [Ja09] with the accuracy 0.9947 and unsupervised learned filters. One can draw on these examples to estimate the reasonable layer count, which in our case varies from 6 to 8. Note that the results described below were derived from MNIST experiments, the trials with CIFAR-10 being similar are omitted due to space constraints.

The choice of the fitness function is crucial for the success of GA and MA. It should reflect the quality of a CNN architecture, duly incorporating both loss and accuracy: $Q_{fit} = -\alpha \cdot Q_{loss} + \beta \cdot Q_{acc}$, where $0 \leq \alpha < \beta$ and $\alpha + \beta = 1$. We limited ourselves to several combinations of α and β and extensively tested them, varying other specific parameters of the algorithms. The best combination turned out to be $\alpha = 0.25$ and $\beta = 0.75$, proving the hypothesis about smaller significance of loss for network quality.

Next step was to determine the best configuration for GA and MA. The number of iterations ($N = 10$) and the population size ($M = 30$) were set taking not only the available hardware into account, but also the desired variety of CNN architectures. Other specific parameters like selection and mutation ratio (p_c and p_m), shared between both algorithms, were comprehensively analyzed as well. The values $p_c = 0.5$ and $p_m = 0.75$ proved to induce the best search behavior, meaning that the fittest half of the population should be chosen to reproduce, and $^3/_4$ of their children should be subjected to mutation. As a result of more population diversity, the fitness over all iterations grew in this case always stronger than in configurations with less selected individuals or less mutation.

The remaining specific parameters of MA (the number of neighbors S to evaluate and the radius of neighborhood R) were examined separately. For instance, the radius R is calculated as the longest distance from the initial individual using its coordinates in the search space. A very good value of radius turned out to be $R = 0.15$, given the fact that it allowed us to find enough valid CNN architectures in the corresponding neighborhood. Moreover, relatively large jumps in the search space are possible, which can be very helpful to avoid local optima. We also found out that higher radius leads to longer search, as the probability of finding valid individuals decreases. In contrast, lower radius helps to preserve the local nature of the search, but unfortunately reduces the variety of neighbors, preventing MA to significantly increase the current fitness. The number of neighbors was fixed to $S = 5$ as a balance between added computational complexity and range of the search.

By setting the specific parameters of GA and MA to the aforementioned values we ensured that both algorithms perform to the best capability. Subsequently, we analyzed their properties

in order to find out how the solution was found in each case. This was accomplished by inspecting the generated architectures and the shift of their fitness values during runtime.

So how do the proposed algorithms change CNN architectures in the course of N iterations? As the initial architectures are random, there is a great variety of hyper-parameter values at the start of each algorithm. Then the variety is considerably reduced by GA, as the search quickly concentrates on one region with the fittest individuals known so far. The mechanism of crossover does not allow for significant changes in the genetic material, for which mutation remains solely responsible. So the last iteration of GA produces architectures that are very similar to each other, mostly differing in the number of outputs. MA, on the other hand, makes use of Local Search, constantly inserting new fitter neighbors into the population and thus keeping the range of genetic diversity more stable.

Another question arises: Does genetic diversity bring better results? In other words, what fitness shift can be expected from both algorithms? To answer this, we tracked the currently fittest individual in each population. Fig. 2 illustrates typical instances of fitness shift for GA and MA. Due to elitism, fitness can only increase in both cases. However, because of elitism GA does not change the fittest individuals, causing fitness plateaus as seen in Fig. 2, left. MA differs from GA in this respect, as it substitutes all individuals through their fitter neighbors, thus inducing the strictly monotonic fitness growth (Fig. 2, right).

Worthy of noting is the fact that both algorithms start with at least one very good CNN architecture in the first generation, which is a great advantage of population-based search algorithms. While the best accuracy of the first generation was rarely under 0.9, the outcoming CNNs had average accuracy above 0.97. The best observed accuracy of GA was 0.9903, MA achieved 0.9902 with less layers, approaching state of the art accuracy 0.9905.

Finally, to compare the performance of GA and MA we strategically provided them with the same preconditions, i.e. same initial population containing 7-layered CNN architectures. The averaged results of this series of trials are demonstrated in Tab. 2. As a reference algorithm

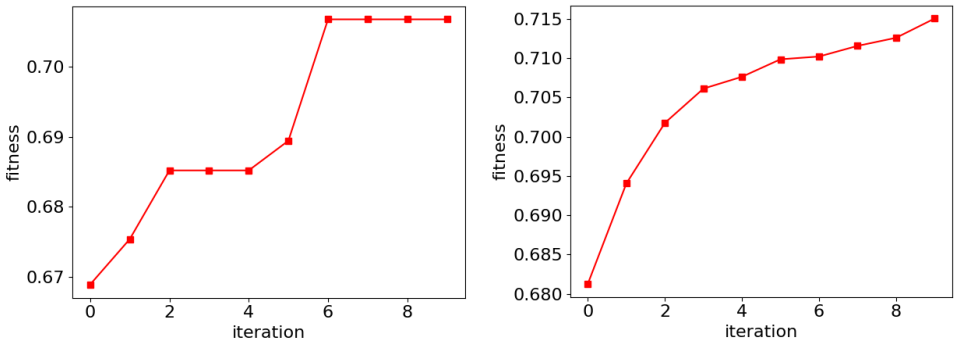

Fig. 2: The development of each generation's best fitness in GA (left) and MA (right).

to compare against we also implemented Random Search (**RS**, see section 3), starting it with the initially fittest individual of the given population and running for 100 iterations. In order to measure the success of the algorithms we have chosen a fitness increase as metrics, i.e. the difference between the best fitness of the first and the last generation. As expected, RS shows the lowest average fitness increase of all – 7.5 %. Surprisingly, GA gets only marginally better than RS (7.8 %) due to its limitations with respect to diminishing genetic diversity. The winner of these trials is MA with its ability to increase the given fitness up to 14.7 % – nearly twice as much as GA. This excellent performance can be explained with the benefits of Local Search, which enables MA to explore the neighborhood of every individual, thus avoiding too quick convergence and local optima. The average runtime of both algorithms shown in Tab. 2 should be regarded as reasonable, considering the fact that the manual search through all hyper-parameters of Tab. 1 might take longer than a week.

Algorithm	Count Specific Parameters	Mean Fitness Increase (%)	Computational Complexity	Avg. Runtime (days)	Best Known Accuracy
RS	1	7.5	linear	<1	0.9857
GA	4	7.8	quadratic	<2	0.9903
MA	6	**14.7**	cubic	5	0.9902 *

Tab. 2: Summary of our experimental results (* – achieved with less layers).

6 Conclusion and Future Work

In this paper, we analyzed the problem of finding good hyper-parameter values for CNN architectures and solved it using two evolutionary algorithms. The first one is Genetic Algorithm, the second is Memetic Algorithm, a hybrid of GA and Local Search. We demonstrated that these algorithms can be successfully adjusted to the case of CNN and even be superior to the standard search algorithms from the literature, like Random Search.

The proposed algorithms contain significantly less specific parameters than hyper-parameter amount in CNNs (4 or 6 instead of 42 in 7-layered CNNs, cf. Tab. 1, 2), making it easier to determine them manually if needed. We also illustrated the considerations behind the choice of these specific parameters. After they are set to reasonable values, both algorithms are able to efficiently explore the search space and find CNN architectures with nearly state of the art accuracy on the given dataset without any user interference. In addition, we evaluated the performance of both algorithms using fitness increase as metrics. The experiments show that GA provides stable fitness increase and improves the properties of CNN architectures with genetic operators. MA is more complex, but its performance level is twice as high as that of GA due to Local Search. Both GA and MA are population-based, independent of initialization and capable of searching for good architectures in several promising areas.

However, GA and MA can be further improved by means of parallel processing. It would facilitate the experiments with extremely large datasets like ILSVRC, which would take a week to train one CNN on. Besides, other techniques to increase the accuracy level can be

taken into account, for example, other kinds of preprocessing [Hi12, ZF13] or tuning the training parameters. In the end, the proposed algorithms can be considered as a good support for scientists who desire to improve the accuracy of their CNN models by systematically searching for appropriate hyper-parameter values.

References

[BB12] Bergstra, James; Bengio, Yoshua: Random Search for Hyper-parameter Optimization. J. Mach. Learn. Res., 13(1):281–305, February 2012.

[Be11] Bergstra, James S.; Bardenet, Rémi; Bengio, Yoshua; Kégl, Balázs: Algorithms for Hyper-Parameter Optimization. In (Shawe-Taylor, J. et al., eds): Advances in Neural Information Processing Systems 24, pp. 2546–2554. 2011.

[CG11] Correa, B.A.; Gonzalez, A.M.: Evolutionary Algorithms for Selecting the Architecture of a MLP Neural Network: A Credit Scoring Case. In: Data Mining Workshops (ICDMW), 2011 IEEE 11th International Conference on. pp. 725–732, Dec 2011.

[Ch11] Chang, Y.; Wang, Y.; Ricanek, K.; Chen, C.: Feature selection for improved automatic gender classification. In: 2011 IEEE Workshop on Computational Intelligence in Biometrics and Identity Management (CIBIM). pp. 29–35, April 2011.

[Hi12] Hinton, Geoffrey E. et al.: Improving neural networks by preventing co-adaptation of feature detectors. CoRR, abs/1207.0580, 2012.

[Ja09] Jarrett, K.; Kavukcuoglu, K.; Ranzato, M.; LeCun, Y.: What is the best multi-stage architecture for object recognition? In: Computer Vision, 2009 IEEE 12th International Conference on. pp. 2146–2153, Sept 2009.

[KS05] Krasnogor, N.; Smith, J.: A tutorial for competent memetic algorithms: model, taxonomy, and design issues. IEEE Transactions on Evolutionary Computation, 9(5):474–488, 2005.

[KSH12] Krizhevsky, A.; Sutskever, I.; Hinton, G. E.: ImageNet Classification with Deep Convolutional Neural Networks. In (Pereira, F. et al., eds): Adv. in Neural Information Processing Systems 25, pp. 1097–1105. 2012.

[Le98] LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, Nov 1998.

[MSM16] Mishkin, Dmytro; Sergievskiy, Nikolay; Matas, Jiri: Systematic evaluation of CNN advances on the ImageNet. CoRR, abs/1606.02228, 2016.

[OI11] Oong, Tatt Hee; Isa, N.A.M.: Adaptive Evolutionary Artificial Neural Networks for Pattern Classification. Neural Networks, IEEE Transactions on, 22(11):1823–1836, Nov 2011.

[Sn15] Snoek, J. et al.: Scalable Bayesian Optimization Using Deep Neural Networks. In: Proc. of the 32nd Intl. Conf. on Machine Learning - Vol. 37. ICML'15, pp. 2171–2180, 2015.

[Ya99] Yao, Xin: Evolving artificial neural networks. Proc. of the IEEE, 87(9):1423–1447, 1999.

[ZF13] Zeiler, Matthew D.; Fergus, Rob: Stochastic Pooling for Regularization of Deep Convolutional Neural Networks. CoRR, abs/1301.3557, 2013.