BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2019

http://excel.fit.vutbr.cz

Evolutionary Algorithms in Convolutional Neural Network Design

Filip Badáň*

Abstract

Designing deep convolutional neural networks, which are nowadays successfully applied in a wide range of various fields, can be a very challenging task requiring a great deal of experience. The aim of this work is to minimize human effort needed to design convolutional neural networks and automatize the whole process of finding efficient and successful architectures. Proposed framework uses a modified version of an evolutionary algorithm, which is developed to find accurate and fully-trained convolutional neural networks to solve various image classification problems. The framework uses the so-called weight inheritance technique, which allows the training process to be considered as a special kind of mutation and by that drastically reduce time complexity of the evolution cycle. An innovative concept of the training age which gives ”younger” but potentially better candidates an opportunity to succeed is also proposed. The framework has been validated on the standard image classification datasets – MNIST and CIFAR10. The initial experiments yielded fully-trained networks with almost 99% test accuracy on MNIST dataset in a relatively short evolution time. The results showed that neuroevolution has a promising potential to automatize the process of designing neural networks.

Keywords: Evolutionary algorithms — Convolutional neural networks — Neuroevolution

Supplementary Material: *N/A*

*Faculty of Information Technology, Brno University of Technology

1. Introduction

Convolutional neural networks (CNNs) are currently successfully used in a wide range of fields such as computer vision, image classification and natural language processing [1]. For many of these application domains, very deep neural networks (i.e. networks with many hidden layers) are needed to solve the problem. Finding accurate (and possibly compact) neural network structures for solving complex problems is usually a hard task even for experienced neural network designers. It requires a significant amount of experiments and great knowledge of processed data. That is the reason, why there is growing potential for algorithms finding complex CNN architectures, which are capable of delivering the best results on a given dataset.

Evolutionary algorithms have proven to be effective in many optimization problems, for example, in hardware design [2], astronomy, robotics [3, 4] and planning. It is therefore natural, that evolutionary algorithms have also been used to automatize the process of developing neural network structures and they have quickly became a respected method used for this task. In the literature, the connection of evolutionary computing and neural networks is often referred to as *neuroevolution*.

The current state-of-the-art work in the neuroevolution field achieved promising results and showed a big potential for automatization of a neural network design with the help of evolutionary algorithms.

The goal of this work is to design and implement a framework which can be used by CNN designers to automatically discover effective and accurate network architectures or to optimize existing but not satisfactory ones. It uses a slightly modified version of common evolutionary algorithms with some novel techniques.

The results achieved after initial experiments showed that is is possible to use the proposed method to design convolutional neural networks. The best result achieved on MNIST dataset was 98.88% and 62.3% accuracy was achieved after the experiment on CIFAR10 dataset. These initial results were gained with limited computational resources allocated and within a few experiments only. The detailed explanation of these results and future ideas how to enhance them will be discussed in later sections.

Please note, that the work is still in progress and presented results are not final. More experiments on a larger scale are required to explore the full potential of the proposed algorithm.

2. Theoretical background

This section briefly describes convolutional neural networks and evolutionary algorithms, which are needed to understand this paper.

2.1 Convolutional neural networks

Convolutional neural networks are a very popular class of feedforward deep neural networks used for image classification and recognition tasks [1]. They consist of an input layer, an output layer and (usually) many hidden layers. Basic component of a CNN is a *convolutional layer*, whose role is to extract new features from an input image by the operation called *convolution*. The extracted features are then used by the network to determine an output class of an input image. Values of filters applied through the convolution at each convolutional layer are learned by means of training on the training data set. After the training process is finished, the network is validated on another data set – the test set. The second main type of CNN layers is called *pooling layer*. It is used for reducing the spatial size of an input image. A fully-connected layer is usually (but not necessarily) applied in the end of a CNN to perform the classification operation.

2.2 Evolutionary computing

The family of algorithms inspired by biological evolution is uniformly called evolutionary algorithms (EA). The main driving force of biological evolution is the natural selection or, in other words – the survival of the fittest principle. The idea behind the natural selection mechanism is following: in a population of individuals competing for limited resources, only the best ones are able to survive or to reproduce. In the context of evolutionary algorithms, the quality of each individual is measured by a so called fitness function, which is highly dependent on the problem being solved. Every potential solution – phenotype – is encoded as a *genotype* inside an EA. Variation operators called mutation and crossover (recombination) are then applied to some selected genotypes to produce offspring individuals. In order to increase the mean quality of individuals in a population, only individuals with the highest fitness are selected as parents or will survive to the next generation. The process of selection, recombination and evaluation is repeated until a suitable solution is discovered or the available time is exhausted.

3. Related work

Roots of neuroevolution date back to 80s when a simple genetic algorithm evolving neural network structures was introduced [5]. Since then, evolutionary computing was successfully applied not only in discovering architectures of neural networks but also as a promising alternative (or addition) to the backpropagation algorithm which is the major method used for learning of neural networks [6] [7]. While our work is concerned with evolving only structures of neural networks, this paper will focus only on this area of neuroevolution.

American researcher Kenneth O. Stanley introduced his algorithm **NEAT** (NeuroEvolution of Augmenting Topologies) in 2002 [8]. NEAT has later grown into the foundation for many other successful applications of neuroevolution. The basic principle of NEAT algorithm is in creating nodes (neurons) and the connections between them, including their weights. This is done by a simple set of mutations and the evolution process. It uses a direct encoding which means in the context of neuroevolution that every neuron is directly represented by the particular gene. While NEAT works perfectly for smaller networks, it becomes very inefficient when it comes to discovering deeper ones.

This inefficiency was the main reason why in 2009 Stanley proposed **HyperNEAT** [9] which uses a special kind of indirect encoding called Compositional Pattern Producing Networks (CPPN). CPPNs use complex functions to set the weights of connections between two nodes. The structure of a network is partly predefined and only these functions are evolved. This allows the algorithm to be much more scalable and find more complex structures.

In 2017 **DeepNEAT** and **CoDeepNEAT** were presented [10]. The authors of these algorithms have tried to extend the original NEAT algorithm to be able to find very deep neural networks. To achieve this, instead of simple neurons they encoded entire layers (together with their parameters) into a genotype. They also brought a high level of modularity into the design and were able to evolve very deep and accurate architectures which could almost compete even with the state-of-the-art architectures.

The first relevant work on evolution of convolutional neural networks was **Large-scale evolution of image classifiers** proposed in 2017 by Real et al [11]. They were the first to achieve results comparable to the state-of-the-art CNNs on commonly used benchmark data sets for image classification – CIFAR10 and CIFAR100. In their work they used a modified version of a genetic algorithm with a set of mutations inspired by NEAT, but customized for CNNs. Similarly to DeepNEAT they also made use of encoding of entire layers to be capable of discovering very deep architectures. The paper also presented an idea of combining classical training process with evolution via the weight inheritance. In other words, it means that every offspring can inherit learned weights from its parent every time possible.

The most recent work on evolving CNNs is **CNN-GA** from 2018 [12]. It utilizes the power of residual connections and modularity to surpass even the results of the **large-scale evolution** and all other neuroevolution algorithms. The basic building block of CNN-GA is a module consisting of two convolutional layers and one residual connection called *skip layer*. Another type of a building block is the pool layer module and every neural network is a combination of these modules with various parameters. Evolution of networks is realized by the genetic algorithm utilizing a special set of mutation and crossover operators. As the obtained results are remarkable – with 95.22% accuracy on CIFAR10 and 77.97% on CIFAR100 data set – this work can be considered as the current state-of-the-art in this area.

4. Neuroevolution framework

The goal of this framework is to build up on concepts proposed in the literature and introduce some new ideas aimed to further reduce time and computational complexity of the neuroevolution process. The main principles of the proposed framework will be described in this section. Figure 1 shows a scheme of main components of the framework and the connections between them.

The evolutionary algorithm works independently of used CNN library thanks to the wrapper whose objective is to map a structure of a network represented by a genotype to the internal representation of CNN in the library. The library is responsible for training and evaluating candidate solutions and saving learned parameters. Storing weights in separate files is important because of the weight inheritance which will be described later in the text in more detail.

The library used for this work is called TinyDNN [13], which was chosen mainly because of its speed and flexibility. It only consists of header files and is, therefore, very compact and portable. It supports all common features of convolutional neural networks published in the literature. For optimizing the parameters it applies the standard backpropagation algorithm based on stochastic gradient descent.

5. Proposed evolutionary algorithm

This section will outline the most important characteristics of the proposed evolutionary algorithm. For now the algorithm uses only mutation operators which are slightly inspired by [11] and work as follows:

- **INSERT** - inserts a convolutional/pool layer on random index
- **REMOVE** - removes the layer on random index
- **ALTER** - alters the layer on random index according to the type of layer
- **ALTER LEARNING** - changes the learning rate
- **RESET** - resets the weights
- **TRAIN** - trains the individual for one epoch using backpropagation

The selection mechanism is based on modified $(\mu + \lambda)$ selection where μ is the number of parents and λ is the number of offspring. It is obvious that while the TRAIN mutation can raise the individual's fitness (i.e. accuracy), mutations like INSERT or REMOVE would almost surely lower it, at least until the weights are optimized again. It is, therefore, important to give a chance to survive even for more innovative but less trained individuals. That is the reason why the concept of *training age*, inspired by speciation (or niching) in NEAT algorithm, was introduced. While in NEAT the individuals are separated into species by a similarity in the structure, in this work the separation is done based on the *training age*. The training age is raised every time the individual undergoes a training process (via mutation) and lowered when some accuracy-lowering mutation (such as inserting or removing a convolutional layer) is executed. The selection mechanism then selects k best candidates from a group of individuals with the same age, where k is a random number between zero and a number given by the population size. This cycle is repeated until μ individuals are selected. The entire cycle of the evolutionary algorithm is given in Algorithm 1.

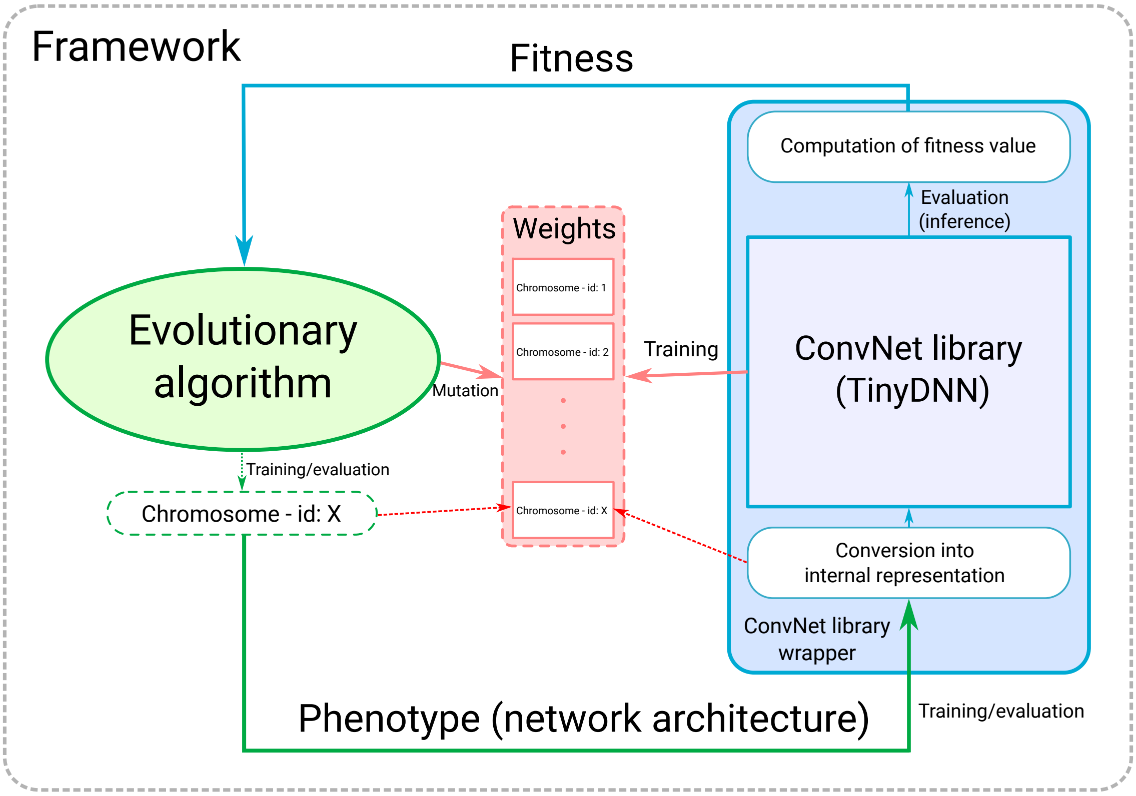

Figure 1. Framework scheme. The evolutionary algorithm is generating genotypes (chromosomes) which are converted into the internal representation of CNN library in the library wrapper. Afterwards, the network can be trained or evaluated by the library. The accuracy achieved by the network is sent as a fitness value back to the EA. The weights are stored in separate files and can be changed by either the library or the evolutionary algorithm.

Algorithm 1 Pseudo-code of the evolutionary cycle.

```
procedure EA
    initialize a population
    evaluate the initial population
    while CurGeneration < Generations do
        mutate all parents to create offspring
        evaluate the new individuals
        while individuals != population size do
            for every age class do
                get random number k from (0,n)
                select k best individuals
            end for
        end while
    end while
end procedure
```

5.1 Problem encoding

Encoding of a phenotype into a genotype is shown in Figure 2. Every genotype consists of a header with a basic information about the network represented by the genotype and its structure (body). Structure of the network is a combination of two different types of layers:

- *Convolutional layer* – basic building block of convolutional neural networks;
  parameters: kernelSize, outChannels, stride and padding.
- *Pool layer* - downsampling of the input;
  parameters: stride, poolSize and poolType.

Because of the proposed straightforward encoding of candidate networks, the conversion from a genotype to the internal representation of a CNN in the CNN library is very simple. Only differences between the encoded genotype and the final representation are the activation layers which come after convolutional layers and two final layers responsible for smooth transition to the output classes.

5.2 Weight inheritance

In order to reduce the time and computational complexity of the entire evolution, all learned weights are inherited from the parent to offspring as proposed in [11]. This technique is also further enhanced by preserving as many weights as possible, even in the case mutation changes shape of some layer. If there are fewer weights in the mutated network, old weights are resized and cut accordingly. Other way around, if

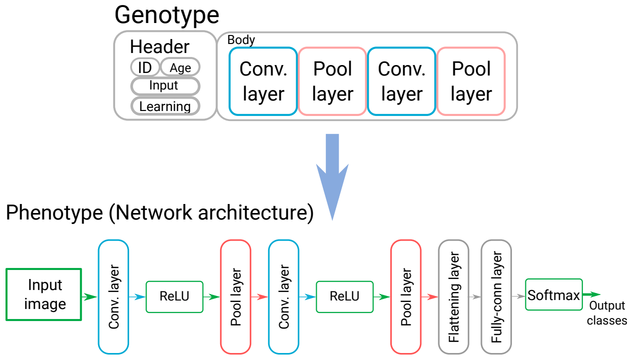

Figure 2. Example of conversion of the genotype into the phenotype (network architecture). There is a ReLU activation behind every convolutional layer and two final layers (flattening and fully connected layer) responsible for the transition to classification classes. For the sake of simplicity the parameters which are also encoded in the chromosome are disregarded in this figure.

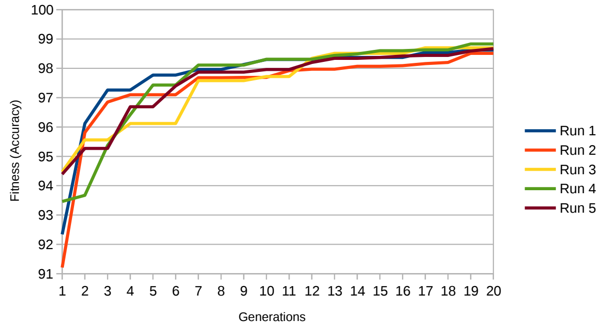

Figure 3. The progress of five independent EA runs in the task of CNN-based MNIST classifier design. The horizontal axis represents the fitness i.e. the accuracy of the best solution and the vertical axis represents the number of generation.

there is more weights in the mutated layer, the remaining weights are initialized to zero by the evolutionary algorithm.

6. Experimental results

All experiments were conducted on the IT4T supercomputer Anselm in Ostrava. The experimental results described in this section are only initial and thus only demonstrate the basic potential of the proposed algorithm. More work is required to reach a higher level of accuracy and compete with the state-of-the-art results.

In the first experiment, we started with a CNN with the following parameters: one convolutional layer with six kernels of size 5x5 and one max pooling layer with pool size of 4. The starting learning rate was set to 0.1. The goal of the EA was to modify the topology and find suitable weights. The weights of individual mutations were set as follows:

- *INSERT* – 55
- *REMOVE* – 10
- *ALTER* – 15
- *ALTER LEARNING* – 5
- *RESET* – 5
- *TRAIN* – 45

The MNIST database was chosen as the base benchmark dataset because it can demonstrate the potential of the proposed method in a short amount of time. EA employed a relatively small population of eight individuals and produced 20 generations in each run.

The results on this data set reached 98.70% test accuracy on average from five runs and the best network from all these runs reached 98.88% accuracy. The average execution time is 5.53 hours on a 16-core node of the Anselm computer. It is also important to mention that the reported networks emerged only from the evolutionary process which means that the framework is able to discover accurate networks without the need of additional training. The results are promising in the context of future experiments of larger scale (i.e. bigger population and more generations) and new algorithm improvements. Additional tuning of the parameters of the EA, such as mutation probabilities, can also help to reach a higher test accuracy.

A single run of the EA on CIFAR10 data set was also conducted with the same initial CNN and the EA operating with 10 individuals in the population and 30 generations. The test accuracy reached in this experiment was 62.37% which is far from the state-of-the-art results. The reason for this is that CIFAR10 data set requires deeper networks to reach a reasonable accuracy and our experiment was executed for too short period of time. More experiments are needed to determine the ability of the framework to find such deep neural networks.

7. Conclusion

This paper introduced a new framework intended for minimizing human effort needed for designing convolutional neural networks on a specific task in the category of image classification. To achieve this it uses modified version of the evolutionary algorithm with some innovative techniques such as *training age*.

The framework was validated using a few initial experiments on the standard benchmark data sets of MNIST and CIFAR10.

7.1 Future work

While this project is still in progress, many new ideas are being worked on. There is an ongoing process of tuning evolution parameters to get the best results possible. Another new feature being explored is an employment of a crossover operator which could significantly fasten up the evolution. Last but not the least, integrating residual connections which have proven to be effective in many deep CNNs could help in finding very deep neural networks architectures.

Acknowledgements

I would like to sincerely thank my supervisor prof. Lukas Sekanina for his help and assistance while working on this project. This work was supported by The Ministry of Education, Youth and Sports from the Large Infrastructures for Research, Experimental Development and Innovations project „IT4Innovations National Supercomputing Center – LM2015070“.

References

[1] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, and Gang Wang. Recent advances in convolutional neural networks. *CoRR*, abs/1512.07108, 2015.

[2] Lukas Sekanina. Evolutionary hardware design. In *VLSI Circuits and Systems v*, volume 8067, page 806702. International Society for Optics and Photonics, 2011.

[3] Mehrdad Dianati, Insop Song, and Mark Treiber. An introduction to genetic algorithms and evolution. 2002.

[4] Adam Marczyk. Genetic algorithms and evolutionary computation. *The Talk Origins Archive: http://www. talkorigins/faqs/genalg/genalg. html*, 2004.

[5] Geoffrey F Miller. Designing neural networks using genetic algorithms. In *ICGA*, 1989.

[6] José Parra, Leonardo Trujillo, and Patricia Melin. Hybrid back-propagation training with evolutionary strategies. *Soft Comput.*, 18(8):1603–1614, August 2014.

[7] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *CoRR*, abs/1712.06567, 2017.

[8] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evol. Comput.*, 10(2):99–127, June 2002.

[9] Kenneth O. Stanley, David B. D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artif. Life*, 15(2):185–212, April 2009.

[10] Risto Miikkulainen, Jason Zhi Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, and Babak Hodjat. Evolving deep neural networks, 2017.

[11] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc Le, and Alex Kurakin. Large-Scale Evolution of Image Classifiers. *arXiv e-prints*, page arXiv:1703.01041, March 2017.

[12] Yanan Sun, Bing Xue, Mengjie Zhang, and Gary G. Yen. Automatically designing CNN architectures using genetic algorithm for image classification. *CoRR*, abs/1808.03818, 2018.

[13] TinyDNN team. Tinydnn. `https://github.com/tiny-dnn/tiny-dnn`.