Design space exploration of Convolutional Neural Networks based on Evolutionary Algorithms

Abeer Al-Hyari University of Guelph, ON, Canada
Shawki Areibi University of Guelph, ON, Canada

Abstract

This paper proposes a framework for design space exploration of Convolutional Neural Networks (CNNs) using Genetic Algorithms (GAs). CNNs have many hyperparameters that need to be tuned carefully in order to achieve favorable results when used for image classification tasks or similar vision applications. Genetic Algorithms are adopted to efficiently traverse the huge search space of CNNs hyperparameters, and generate the best architecture that fits the given task. Some of the hyperparameters that were tested include the number of convolutional and fully connected layers, the number of filters for each convolutional layer, and the number of nodes in the fully connected layers. The proposed approach was tested using MNIST dataset for handwritten digit classification and results obtained indicate that the proposed approach is able to generate a CNN architecture with validation accuracy up to 96.66% on average.

1 Introduction

Convolutional Neural Networks (CNNs) have proven their superiority in many vision tasks. The main reason behind the success of CNNs deep architecture is their ability to learn useful, rich, and invariant features through deep, nonlinear transformations. This represents an attractive property to adopt in various vision tasks, such as object detection and object tracking.

Figure 1 depicts the architecture of AlexNet that won the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) in 2012 [1]. It consists of five convolutional layers, three max-pooling layers, and three fully connected layers including a softmax layer. Despite the great success that CNNs have achieved in vision tasks, there is no single architecture that can target all type of domains. Moreover, there is an exponential relationship between the number of layers and their corresponding parameters. Thus, searching for suitable CNN architecture in a systematic way is an important step, which is referred to as design exploration. Design exploration for CNNs can be defined as selecting the hyperparameters, layers, type of layers, connectivity between layers and also the order of various layers. Hyperparameters in CNNs are the parameters that cannot be learned during the training process of the model. Instead their values are assigned and predefined by the designer of CNN prior to the training process. The most common ways to select the hyperparameters values are either based on experience or based on previous successful models. The associated models are implemented, trained, and then the model with the highest validation accuracy is selected.

In this paper, we propose a simple yet efficient approach for the design automation of CNNs. Section 2 presents a brief background of the most representative work in this domain. The methodology section provides more details about the proposed framework. Section 4 presents a summary for the results that were obtained. Conclusions and future work are presented in Section 5.

2 Background

Recently there has been lots of interest in implementing different approaches for design space exploration of CNNs. Examples of these approaches include reinforcement learning [2] [3], Bayesian optimization [4], and Genetic algorithms [5, 6, 8, 9, 10, 11, 12].

Genetic algorithms (GAs) are metaheuristic methods that were inspired by the process of natural selection. GAs rely on biology-inspired operators, such as mutation, crossover, and selection. GAs were adopted as a design exploration tool for Artificial Neural Networks (ANNs) and CNNs due to their efficiency in searching and exploring the huge solution space. The most representative work using GAs in this domain is explored in this section. NEAT (NeuroEvolution of Augmenting Topology) is considered among the earliest work in this domain [6]. NEAT addresses the evolution of both the topology and weights of ANNs. NEAT was later extended in 2009 to Hypercube-based NEAT (HyperNEAT) [7]. The main goal of this extension is to enable the effective evolution of high-dimensional ANNs by introducing indirect encoding in which the weights are generated as a function of geometry. Following the great success of CNNs in 2012, there were some attempts to extend HyperNEAT to be applicable to deep neural networks as in DeepNEAT [10], in which each gene represents a layer instead of representing a single neuron as in NEAT. The gene also contains a table of hyperparameters that may be mutated during the evolution context. Cooperative DeepNEAT (CoDeepNEAT) was next introduced to evolve two subpopulations: one is for modules and the other is for the blueprints.

Multi-node Evolutionary Neural Networks for the Deep Learning (MENNDL) framework is proposed to address the optimization of CNN hyperparameters using GAs [8]. The hyperparameters that were optimized are the filter size and the number of filters for each convolutional layer, which was restricted to three layers. The resulting network is fully-trained and no further post-processing is required. Tirumala et al. [9] studied the reduction of training time of Deep Neural Networks (DNNs) by adopting evolutionary approaches. The training time of DNN was accelerated over the regular approach when using MNIST dataset. GAs thus provide a promising start for the deep learning by evolving optimized DNNs instead of adopting heuristic random initial architecture. Real et al. [5] examined the feasibility of using GAs to generate fully-trained CNNs that do not need any further interventions from humans. A novel and intuitive set of mutation parameters was introduced to mimic the actions that human designers are following in designing CNNs. Using GAs as a tool to efficiently traverse the huge search space related to CNN design was investigated by Xie and Yuille [11]. However, their proposed approach was based on a constrained case that assumed a CNN consisted of only three convolutional layers. Cartesian Genetic Programming (CGP) was used to automate CNN design for image classification task [12]. It enables the generation of variable-length networks and skip connections. Thus, it enables the generation of highly competitive CNN architectures that can compete with the state-of-the-art networks.

3 Methodology

In this section we introduce the chromosome representation used in this work followed by proposed framework.

3.1 Proposed chromosome structure

Figure 2 depicts the chromosome structure. It consists of two subparts: the first part is for the convolutional layer parameters, while the second part is for the fully connected layer parameters. The proposed method assumes that the maximum number of convolutional layers is six, while the maximum number of fully connected layers is two including the softmax layer. The chromosome of each convolutional layer is of fixed-length, which consists of nine genes, and five genes for the fully connected layer. Therefore, each chromosome (individual) in the population consists of $(9 \times 6 + 5 \times 1 = 59)$ genes. Table 1 lists the possible values that each gene can have for both the convolutional and the fully connected layers. An active gene determines whether this layer will appear in the phenotype of this chromosome or not. Batch normalization gene has a binary value that reflects whether there is normalization or not. Batch normalization is considered to be one of the essential steps in any image processing or image related application since it has a positive impact on the performance and the robustness of the CNN.

Dropout rate is considered since it prevents overfitting of the CNN. The possible percentage of dropout are between 10% to 50%. The pooling active gene determines the presence or absence of a pooling layer. The pooling type gene determines the type of pooling; the two possible options are either max pooling or average pooling. Lastly for the pooling size, the only available option is (2x2) region. For the chromosome part that is responsible for the fully connected layer encoding, the first gene determines whether

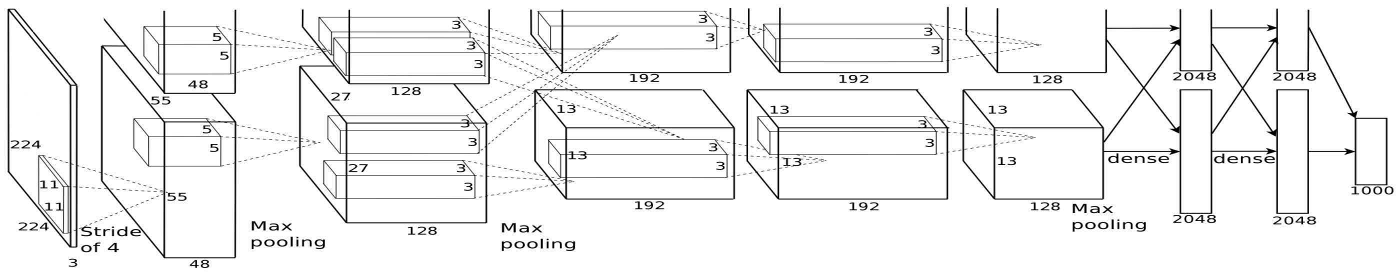

Fig. 1: AlexNet [1]

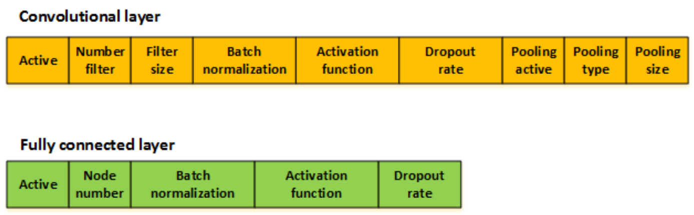

Fig. 2: Proposed chromosome structure

Convolutional layer	
Gene	**Values**
Active	{yes, no}
Number of filters	{ 8, 16, 32, 64, 128, 256}
Filter size	{(1x1), (3x3), (5x5), (7x7)}
Batch normalization	{yes, no}
Activation function	{Relu, Sigmoid}
Dropout rate	{10%, 20%, 30%, 40%, 50%}
Pooling active	{yes, no}
Pooling type	{max pooling, average pooling}
Pooling size	{(2x2)}
Fully connected layer	
Gene	**Values**
Active	{yes, no}
Number of nodes	{16, 32, 64, 128, 256, 512, 1024}
Batch normalization	{yes, no}
Activation function	{Relu, Sigmoid}
Dropout rate	{10%, 20%, 30%, 40%, 50%}

Table 1: The possible values for the genes of the proposed chromosome structure

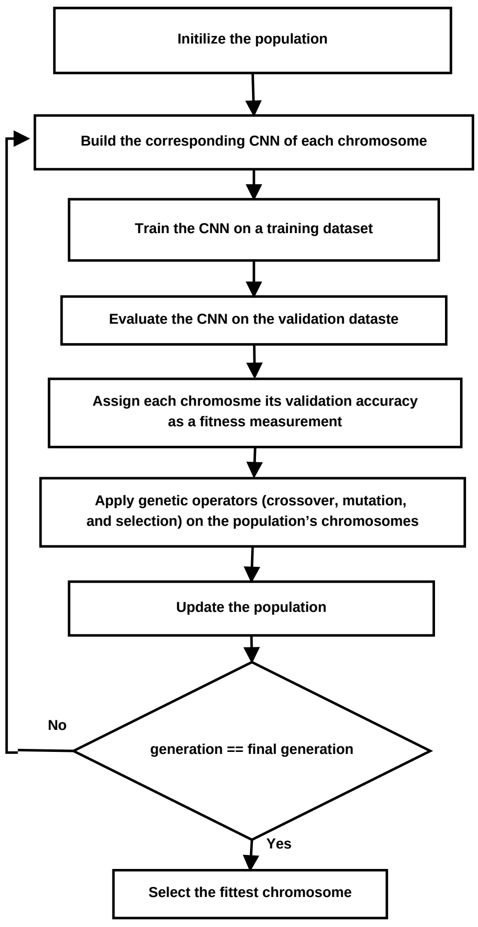
Initilize the population
Build the corresponding CNN of each chromosome
Train the CNN on a training dataset
Evaluate the CNN on the validation dataste
Assign each chromosme its validation accuracy as a fitness measurement
Apply genetic operators (crossover, mutation, and selection) on the population's chromosomes
Update the population
generation == final generation
No
Yes
Select the fittest chromosome

Fig. 3: Flowchart of the proposed approach

the layer is active or not. The second gene decides the number of nodes, the options are: 16, 32, 64, 128, or 256. Also, for the activation function it is either Relu or sigmoid. The last gene of the chromosome determines the dropout rate for the fully connected layer.

3.2 Proposed framework

Figure 3 presents a flow of the proposed approach. Firstly, the chromosomes of the initial population are initialized randomly so that each gene holds one value of the possible values that it might take according to Table 1. The validation accuracy of each CNN network is assigned as a fitness measurement to each chromosome. The genetic operators in the form of crossover, mutation, and selection are then applied to the current population. The current population is updated and a test of convergence is applied. The process of building the new CNNs, evaluating, and evolving these CNNs is repeated until the final generation is reached. The best chromosome with its corresponding CNN architecture is saved for further analysis.

4 Results

4.1 Experimental setup

The proposed framework was written in Python 3.5.2 and was run on a Unix (Ubuntu 16.04) operating system. Each evolved CNN architecture was implemented using Keras deep learning framework [13]. The experiments were run on NVIDIA GeForce GTX TITAN Black GPU. The MNIST dataset is a handwritten digit recognition task [14] that is commonly used for training various image processing systems. It consists of 60000 images for training, and 10000 images that are used for validation purposes. The images are of size (28x28) pixels in grayscale.

4.2 Parameters tuning

GAs have many related parameters that need to be configured. In order to find out the best parameters combination, several experiments were conducted using the MNIST dataset and the obtained results are discussed in this section. Firstly, to have a baseline for parameters tuning experiments of GAs and to prove the convergence of the proposed approach, baseline settings were set. Table 2 lists the parameter values that were used for the baseline case.

Figure 4 depicts the improvement of average validation accuracy over generations. It is clear that there is an increasing trend in the validation accuracy when evolving from one generation to the next. Fine tuning for a certain parameter is tested in each of the following experiments to end up with the best parameter configurations of the GAs approach for CNN design exploration. All the parameters are set up according to Table 2 except the parameter that is under testing.

Table 3 lists the various parameters and their values or methods that were tested in addition to the average validation accuracy

Parameter	Value/Method
Population size	12
Number of Generations	10
Crossover rate	95%
Crossover method	One point crossover
Mutation rate	0.1%
Selection method	Roulette wheel selection
Replacement policy	Best individuals (elitism)

Table 2: Parameter settings for the baseline case

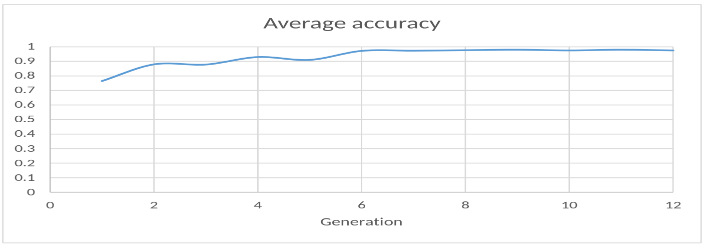

Fig. 4: Average validation accuracy over generations

Parameter	Value/Method	Average validation accuracy
Population size	**8**	**96.99%**
	15	87.12%
	20	90.86%
Crossover rate	80%	90.3%
	85%	85.5%
	90%	87.5%
	98%	**96.6%**
Mutation rate	**0.1%**	**90.92%**
	5%	77.6%
	10%	84.22%
Crossover method	**One-point**	**85.39%**
	Two-point	84.57%
Selection method	**Roulette wheel selection**	**92.12%**
	Random selection	86.09%
	Tournament selection	70.85%
Replacement policy	**Best individuals**	**94.27%**
	All	82.99%

Table 3: Parameter tuning results

achieved for each case. In the first experiment, various population sizes were tested, starting from two individuals up to ten in addition to 15 and 20 individuals. The best overall average accuracy of 96.99% is achieved with population size of eight chromosomes. Crossover is a critical operator in GAs since it simulates the exchange of genetic material between mating parents. The crossover rate determines the probability of randomly mated parents. The rates that were tested are: 80%, 85%, 90%, and 98%, and the best achieved average validation accuracy is 96.6% with a crossover rate of 98%. The mutation rate decides the rate at which genes are randomly mutated or flipped after performing crossover. The genes that are prone to mutation are the active genes since they will result in either adding one whole layer to the network or discard a layer. The mutation rates that were tested are: 0.1%, 5%, and 10%. From Table 3, it is clear that 0.1% can be picked as an optimal mutation rate since it has the highest average validation accuracy.

The crossover operator is responsible for generating new offspring from the genetic material of the mated parents. In general, there are mainly two methods for performing crossover in GAs. The first is one-point crossover in which a random gene is selected and the genes after that point are swapped between mated parents. In the two-point crossover, two points are selected randomly and the genes between those two points are swapped between the mated parents. In the fourth experiment validation accuracy of both methods were measured and the results are shown in Table 3. One-point crossover has a slight improved performance over the two-point crossover. The selection operator in GAs is responsible for determining the individuals that are used to act as parents and generate the next generation of offspring. Usually, the individuals with the highest fitness values are selected. The methods that were examined in the fifth experiment are: roulette wheel selection, random selection, and tournament selection. In roulette wheel selection, the fittest parents are selected, while in random selection the parents are selected randomly. Lastly, selecting an individual in tournament selection depends on the fitness of the individual and the size of the sample to select from it. Table 3 shows that roulette wheel selection is the favored strategy. Further, some individuals have to be replaced after generating the new offspring by the crossover operator in order to maintain the population size. The two main replacement policies that were examined in the sixth experiment are: the best, and replace all population. The best (elitism) replacement policy chooses the best individuals from both parents and evolved offspring. By contrast in the replace all policy, the new evolved offspring are replacing their parents. The best policy has a higher validation accuracy than replace all policy.

5 Conclusion

In this paper we proposed a design exploration framework for CNNs which is promising and achieves high accuracy on the MNIST dataset. There are many directions that can further extend this proposed approach, including (i) modifying the chromosome structure to introduce tuning additional hyperparameters of the CNNs, such as stride, pooling filter size, etc, (ii) studying the diversity of the population at the inception of the evolution process and after evolving each generation, and (iii) applying this approach on more complicated dataset, such as CIFAR-10, Street View House Number (SVHN), and CIFAR-100.

References

[1] Krizhevsky, A. Sutskever, I. and Hinton, G. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* (2012).

[2] Baker, B. Gupta, O. Naik, N. and Raskar, R. Designing Neural Network Architectures using Reinforcement Learning. *arXiv:1611.02167* (2016).

[3] Zoph, B. and Le, Q. Neural Architecture Search with Reinforcement Learning. *arXiv:1611.01578 [cs]* (2016).

[4] Snoek, J. Larochelle, H. and Adams, R. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*(2012).

[5] Real, E. Moore, S. Selle, A. Saxena, S. Suematsu, Y. Le, Q. and Kurakin, A. Large-scale evolution of image classifiers. *arXiv:1703.01041* (2017).

[6] Stanley, K. and Miikkulainen, R. Evolving Neural Networks Through Augmenting Topologies. *Evol. Comput.* (2002).

[7] Stanley, K. D'Ambrosio, D. and Gauci, J. A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks. *Artificial Life* (2009).

[8] Young, S. Rose, D. Karnowski, T. Lim, S. and Patton, R. Optimizing deep learning hyper-parameters through an evolutionary algorithm. *Proc. ACM* (2015).

[9] Tirumala, S. Ali, S. and Ramesh C. Evolving deep neural networks: A new prospect. *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)* (2016).

[10] Miikkulainen, R., Liang, J. Meyerson, E. Rawal, A. Fink, D. Francon, O. Raju, B. Shahrzad, H. Navruzyan, A. Duffy, N. and Hodjat, B. Evolving Deep Neural Networks. *arXiv:1703.00548 [cs]* (2017).

[11] Xie, L. and Yuille, A. Genetic CNN. *arXiv:1703.01513 [cs]* (2017).

[12] Suganuma, M., Shirakawa, S. and Nagao, T. A Genetic Programming Approach to Designing Convolutional Neural Network Architectures. *arXiv:1704.00764 [cs]* (2017).

[13] Chollet, F. et al. Keras.`https://github.com/fchollet/keras` (2015).

[14] LeCun, Y. The mnist database of handwritten digits. `http://yann.lecun.com/exdb/mnist/` (1998).