A Grammar-based Genetic Programming Approach to Optimize Convolutional Neural Network Architectures

Jessica Barbosa Diniz[1], **Filipe R. Cordeiro**[1],
Pericles B. C. Miranda[1], **Laura A. Tomaz da Silva** [2]

[1]Departament of Computing (DC)
Federal Rural University of Pernambuco (UFRPE), Brazil

jessicadiniz@outlook.com, {filipe.rolim,pericles.miranda}@ufrpe.br

[2]School of Technology - Pontifical Catholic University of Rio Grande do Sul (PUCRS), Brazil

laura.angelica@acad.pucrs.br

Abstract. *Deep Learning is a research area under the spotlight in recent years due to its successful application to many domains, such as computer vision and image recognition. The most prominent technique derived from Deep Learning is Convolutional Neural Network, which allows the network to automatically learn representations needed for detection or classification tasks. However, Convolutional Neural Networks have some limitations, as designing these networks are not easy to master and require expertise and insight. In this work, we present the use of Genetic Algorithm associated to Grammar-based Genetic Programming to optimize Convolution Neural Network architectures. To evaluate our proposed approach, we adopted CIFAR-10 dataset to validate the evolution of the generated architectures, using the metric of accuracy to evaluate its classification performance in the test dataset. The results demonstrate that our method using Grammar-based Genetic Programming can easily produce optimized CNN architectures that are competitive and achieve high accuracy results.*

1. Introduction

The ability to acquire knowledge is one of the most prominent features of human intelligence. Machine Learning (ML) investigates how to simulate human learning to achieve computational intelligence, i.e., be able to learn and adapt to a changing environment. With the increase of computational power and amount of available data, ML is able to analyze and identify patterns in the data. One disadvantage of these algorithms is the need to preprocess the data to be able to extract features, requiring expert knowledge about the problem domain [LeCun et al. 2015]. To overcome this disadvantage, Deep Learning (DL) aims at building models that learn representations of the data at several levels of abstraction, allowing the system to learn complex functions without depending entirely on handcrafted features. DL has been applied to many domains, including computer vision [Farfade et al. 2015, Plis et al. 2014], speech recognition [Graves et al. 2013, Deng et al. 2013], and machine translation [Zhang and Zong 2015].

The popularity of this research area led to several other deep learning methods and techniques. One of them is Convolutional Neural Networks

(CNNs) [LeCun et al. 1990]. CNNs are the state-of-the-art in different computer vision tasks, such as object recognition. An example of CNN used for object recognition is VGG [Simonyan and Zisserman 2014] that can have 16 to 19 deep convolutional layers architecture, and gained notoriety on the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) in 2014. After that, several CNN architectures were proposed to a variety of tasks. Unfortunately, as architectures evolved and got deeper, more design choices for CNN architectures and hyperparameter settings were added, impacting directly on CNNs performance. As these models get more and more complex, optimizing all of the involved parameters is becoming a laborious task. To solve this problem, researchers have focused their efforts on automating the design of such deep networks.

In this article, we present a study that applies principles of genetic programming to optimize CNN topologies. A population-based evolutionary algorithm is executed with a specific grammar capable of generating valid CNNs, and all networks have their effectiveness evaluated according to the accuracy value obtained after testing the CNN. The execution of CNN training and test were made using the image classification dataset CIFAR-10.

The rest of the paper is organized as follows. Section 2 provides a background on Convolutional Neural Network and genetic programming, as well as Grammar-based Genetic Programming. Section 3 reports related work found in the literature. In section 4, we explain our approach to generate Convolutional Neural Network architectures and the technique in which the proposed algorithm is based on. The experimental environment, dataset, and experimental setup are presented in Section 5. Section 6 discuss experimental results, and, finally, Section 7 contains the conclusion and future work of the study.

2. Background

Convolutional Neural Networks (CNNs) are a particular type of the Artificial Neural Networks (ANNs), specialized in processing data in the form of multiple arrays such as images (2D), audio, video, and volumetric data (3D) [Goodfellow et al. 2016]. CNNs have been successfully employed in several applications, achieving state-of-the-art results in image-based tasks such as object recognition, detection, and semantic segmentation [Krizhevsky et al. 2012, Long et al. 2015].

Despite the popularity of CNNs, its design involves a large number of choices, where most of them are made by an expert of the domain. Such choices impact heavily on the training and performance of CNNs, e.g., decisions need to be made concerning the number of layers, type of layers, and parametrization of multiple receptive fields that are part of it, as the number of filters, stride, or filter sizes.

Due to factors earlier mentioned, one alternative to improve decisions made on CNNs architectures and its performance is to exploit Genetic Programming to automatically generate CNNs.

Genetic Programming (GP) is an important sub-area of Evolutionary Computation (EC) described in the early 1980s, but clearly defined and established by the research of Koza [Koza 1992]. GP is a technique that aims at automating the construction of computer programs to perform a previous specified task from a high-level statement of a problem. Inspired by biological evolution and its mechanisms, GP follows Darwin's theory of evolution "survival of the fittest" and "genetic propagation of characteristics" principles. To

evolve programs, GP uses nature-inspired genetic operations, such as crossover, mutation, reproduction, gene duplication and gene deletion. GP may also employ developmental processes in which an embryo is transformed into a fully developed structure.

To exemplify this process, GP randomly constructs an initial population of programs, or individuals (making the correlation with natural evolution), using functions and terminals appropriated to the problem domain. Each program is then measured to evaluate how well it solves the given problem (fitness of the program). Next, programs are selected for reproduction following the principle of Darwin's survival of the fittest. Selection methods for reproduction are numerous, but for this work, we briefly explain tournament selection [Hingee and Hutter 2008]. This method is a variant of rank-based selection methods, in which individuals are randomly selected and then ranked according to their relative fitness value, selecting the fittest among them for reproduction. Genetic operations of crossover and mutation are then used to generate new offspring programs from individuals selected on the current population. The crossover operator creates new individuals, that are also programs, using two parental programs, and the mutation operator generates new offspring by altering the individual genotype, constituting the new population. Each new individuals have their fitness measured, and the entire process is repeated for many generations. The last generation, usually, is designated as the result produced by GP and formed by the best programs.

Over the years, various GP methods emerged departing from the original premise that candidate solutions are computer programs represented by tree structures. Our interest here is in Grammar-based Genetic Programming (GGP) [O'Neill and Ryan 2001, O'Neil and Ryan 2003].

Grammars play an important role structuring representations in computer science, being broadly employed to syntactically limit symbolic expressions [Hopcroft 2008]. Their application range from the definition of valid expressions, enforcement of type restrictions to the description of constraints of a computer language. GGP uses formal grammar, usually in Backus-Naur Form (BNF) (notation for expressing the grammar of a language in the form of production rules) to restrict the search space, incorporating the domain knowledge of the problem.

GGP explores the biological process of gene expression, introducing the concept of genotype to phenotype mapping. In general, each generated individual has a variable-length linear genotype, or chromosome, consisting of integer codons (i.e., integer or binary lists), to which genetic operators, such as mutation and crossover, are applied. The genotype is then mapped to phenotype, a program in the specified language by the context-free grammar, to evaluate the individual's fitness. GGP does not use trees for individual representation, but as a temporary structure in the course of mapping. That is to say that instead of operating solely on solution trees, as in standard GP, GGP allows search operators to act on genotypes, partially derived phenotypes, or fully-formed phenotypic derivation trees.

3. Related Work

Many work have been proposed in the literature for the evolution of artificial neural networks in the last years, some of them using Grammar-based Genetic Programming. Tsoulos et al. [Tsoulos et al. 2008] propose the optimization of network topology and

parameters such as input vectors, weights and bias using BNF grammar. The study focused on evolving only one-hidden-layer Feed-Forward Neural Networks and was tested on classification and data fitting problems.

Ahmadizar et al. [Ahmadizar et al. 2015] present a technique composed by a combination of grammatical evolution applied on network topology evolution and genetic algorithm on the search of weights and bias. This approach also evolves only one hidden-layer neural networks. The fitness is based on both classification performance and a proposed adaptive penalty term.

Miikkulainen et al. [Miikkulainen et al. 2017] use the concept of NeuroEvolution (NE) to evolve deep neural network architectures. In their approach, genetic algorithms evolved the hyperparameters, topologies, and components of Convolutional Neural Networks and were able to extend their work to Long Short Term Memory (LSTM) architecture. A downside to their study is the massive amount of available computational resources required.

Recently, Assunção et al. [Assunção et al. 2018] proposed DENSER, an approach to evolve deep neural networks, applied to Convolutional Neural Networks. They combine principles of Genetic Algorithms (GA) with Grammatical Evolution (GE) to directly evolve a sequential list of layers, encapsulating the parameter values in a position of the GA genotype, to facilitate the use of genetic operators. Therefore, being able to reapply this method for different network structures and domains, only changing its grammar. In their work, the method outperformed previous evolutionary approaches to CNNs generations, created CNNs with state-of-the-art performance using less prior knowledge, and evolved CNNs with novel topologies than those designed by hand.

4. Proposed Approach

The proposed approach uses principles of a standard Genetic Algorithm (GA) in combination with Grammar-based Genetic Programming (GGP) technique. In this process, a population of CNN architectures is generated, where each CNN architecture is considered an individual, and it is evaluated to produce a fitness value.

For the generation of individuals, we propose the adoption of a quite simple grammar. Its definition is shown in Grammar 1. It represents a context-free grammar, in the BNF format, where the tag $< FINAL\_EXP >$ indicates the final expression for the CNN. Fc, pool, and conv refer to fully connected layer, pooling layer, and convolution layer respectively. Regardless of the number of layers, all convolution layers have ReLU activation [Glorot et al. 2011]. For the pooling layer, it is used Max Pooling method [Scherer et al. 2010]. The values for the variables N, K, and M determine the number of layers that should be used. The values were chosen based on empirical experiments and values found in the literature. We restrict the value of M and N to 3 because of the computational cost to evaluate several architectures. However, these values can be expanded and adapted to solve complex problems through the generation of deeper architectures.

Figure 1 shows the flowchart for the construction of a CNN architecture based on the grammar. The grammar allows the generation of CNNs with multiple convolution layers followed or not by a pooling layer. A Fully Connected layer is also optional.

$\langle FINAL\_EXP \rangle$ ::= $\langle EXP\_2 \rangle \langle FC \rangle$

$\langle EXP\_2 \rangle$ ::= ($\langle EXP\_1 \rangle$*$\langle M \rangle$)

$\langle EXP\_1 \rangle$ ::= ($\langle CONV \rangle \langle POOL \rangle$)

$\langle FC \rangle$ ::= fc*$\langle K \rangle$

$\langle POOL \rangle$ ::= pool
| ϵ

$\langle CONV \rangle$ ::= (conv*$\langle N \rangle$)

$\langle N \rangle$ ::= 1
| 2
| 3

$\langle K \rangle$ ::= 0
| 1
| 2

$\langle M \rangle$ ::= 1
| 2
| 3

Grammar 1. BNF grammar applied in the study.

CNN architectures composed by more than one convolution layer blocks (a block is here determined by the tag $< CONV >$ in the grammar), which also have pooling layer, have pooling applied to all of their convolution layer blocks. An example of architecture that the grammar can produce is this genotype: (((conv*1)pool)*2)fc*2; the algorithm is capable of converting the function to a CNN with two blocks of one convolution layer and a pooling layer, followed by two Fully Connected layers. Figure 2 shows the architecture created to represent this configuration. Figure

5. Experimental Environment

In this section, we describe the dataset used in our experiments, the details for automatically generating CNNs and evolving its architecture based on GP.

5.1. Dataset

The CIFAR-10 dataset [Krizhevsky and Hinton 2009] consists of $60,000$ images evenly distributed among 10 classes used for object classification. The classes are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck, containing $6,000$ 32×32 RGB color images in each of them. Images are not necessarily centered, i.e., objects may appear in different poses. The Background is different in each image.

5.2. Experimental Setup

The code was implemented using Python programming language. To run GGP, we choose to use PonyGE2 framework [Fenton et al. 2017]. The Keras open source library [Chollet et al. 2015] was used in the development and execution of the deep neural networks.

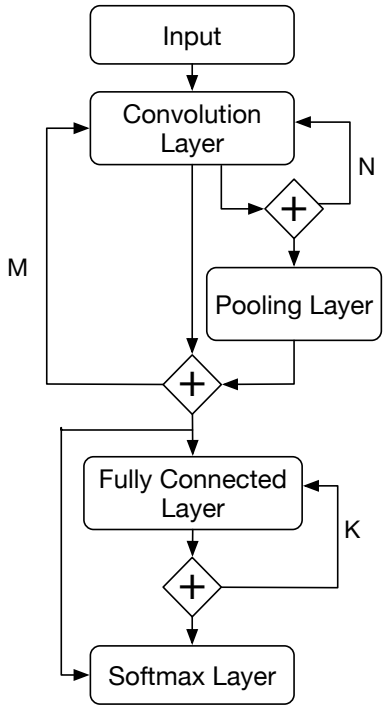

Figure 1. Flowchart of possibilities for CNN topologies using the proposed grammar.

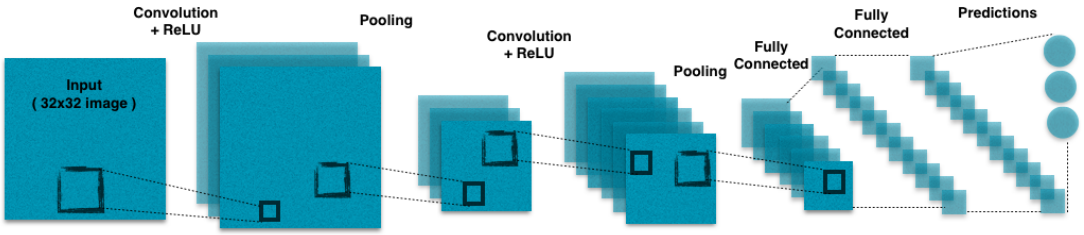

Figure 2. Architecture of a CNN for the following genotype: (((conv*1)pool)*2)fc*2.

5.3. Methodology

Applying the principles of a standard Genetic Algorithm (GA) in combination with Grammar-based Genetic Programming (GGP) technique, a population of CNN architectures was generated. Every CNN was considered an individual in a population and, as so, it was evaluated to produce a fitness value.

This study was performed using a population of 50 individuals and 30 generations. The individuals had their fitness value calculated according to the accuracy they obtained when testing the CNN against the CIFAR-10 dataset. For this research, greater accuracy leads to greater fitness.

The parameters applied in the execution of the genetic algorithm are summarized in Table 1.

The selection process used is the tournament selection technique, in which two random individuals are chosen from the population, and only the one with the best fitness is selected. After the selection, the crossover step is performed by randomly picking

Table 1. Experimental parameters.

Parameter	Value
Number of generations	30
Population size	50
Crossover rate	75%
Mutation	Int Flip Per Codon
Mutation rate	1/Genome length
Tournament size	2
Elite size	1

Table 2. Default CNN hyperparameters

Hyperparameters	Values
Kernel size	3×3
# of filters	Starts with 32; duplicate after every 2 convolutions
Stride	1
Max pooling shape	2×2
Learning rate	0.01
# of epochs	70
# of Neurons FC layer	256

two individuals and swapping their genetic material by applying the variable one-point technique (also known as Single Point Crossover) [Soni and Kumar 2014]. A probability of 75% is used to determine if the crossover must occur in the pair of individuals. After the crossover, every pair of parents produces a pair of children. Mutation is operated on every individual in the child population after crossover is applied. For this process, Int Flip Per Codon [Fenton et al. 2017] mutation is operated on the genomes and randomly mutates every individual codon with a given probability. Every population is replaced by a newly generated child population.

To perform the analysis of the network, the dataset was split into three sets of images, the first one is the training images, having a total of $50k$ images; then we have the other $10k$ images split in validation and test set; validation has 80% of the $10k$, and test have the 20% left. The first set was used during the training of the network, the second for the validation step while the third was used for the tests. All CNNs were trained for 70 epochs and with a batch size equals to 128.

Regardless of the network topology, all CNN generated based on the grammar were executed applying the same hyperparameters. Table 2 shows the hyperparameters shared among all individuals.

Table 3. Three best results from GGP execution.

CNN representation	Accuracy (%)
(((conv*2)pool)*3)fc*1	79.35
(((conv*2)pool)*3)fc*2	79.3
(((conv*2)pool)*3)fc*0	78.8

6. Results

To evaluate the quality of the generated CNNs, we used the accuracy achieved during classification of a test sample set. This metric measures the percentage of the images correctly classified as their real class. During its execution, GGP algorithm evolved its population until it reached a level from where applying mutation in the individuals was only providing slightly better accuracies. Figure 3 shows a graph with the evolution of individual accuracies over the generations.

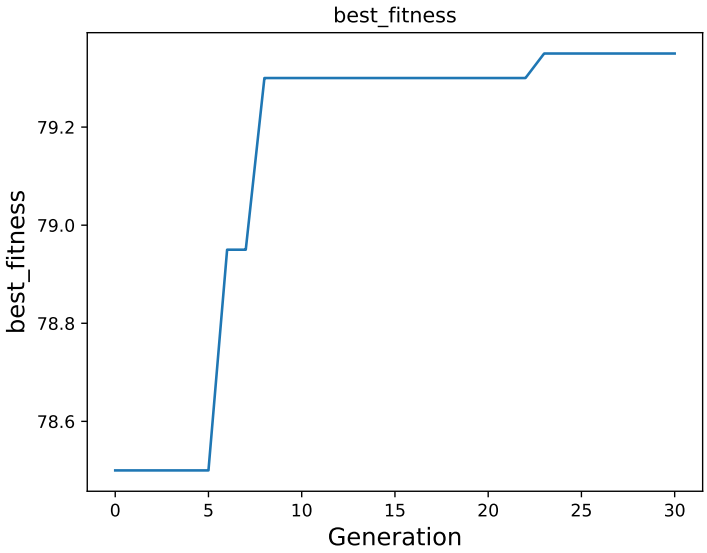

Figure 3. Best fitness evolution during GGP execution.

For this algorithm, the best accuracy value obtained was 79.35%. The three best-achieved accuracies were 79.35%, 79.3%, and 78.8%. Table 3 presents the individual genotypes that represent the best CNNs as well as their accuracy. It is possible to notice that the higher results were reached by CNN consisting of three sets of two convolution layers followed by a pooling layer. Also, the presence of fully connected layer did not have a big impact on the final result. The last point to highlight is the fact that our grammar allows the creation of CNNs with blocks of three convolution layers, followed or not by pooling layers; however, those networks were not among the best results. Therefore, a deeper architecture not always results in better accuracy. Because of this, it is important to use a grammar-based GP approach to find the best architecture.

After running GGP algorithm, the three CNNs that scored the highest accuracy were selected to have some improvements applied on their network. Two modifications

were chosen: the first one was the addition of a dropout layer after every pooling layer if the neural network has it. This helps to provide more control over the overfitting. The second was the addition of a batch normalization after every convolution layer. This aims to increase the overall accuracy and accelerate the learning. The three best CNNs were executed with all possible combination of parameters applied to the improvement. For this process, we chose to execute every CNN three times for each configuration and consider as the result the average accuracy of the three runs.

The best CNN with the addition of both dropout and normalization achieved the best accuracy on this study, 85.77%. We also noticed that this configuration hit the greatest accuracy for all three CNNs solutions. Figure 4 shows the result reached by the three best neural networks from GGP. As it is shown in the chart, after adding dropout and normalization layers, the results increased.

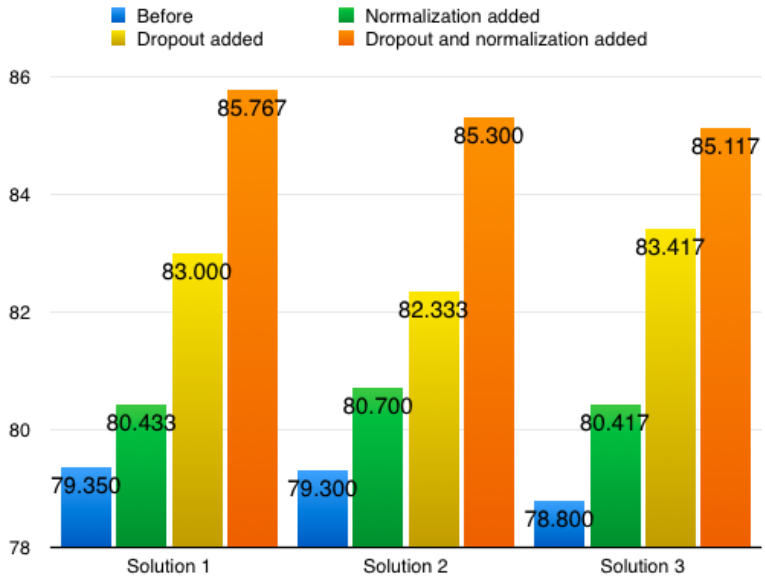

Figure 4. Result comparison of the three best CNNs (here called Solution 1, 2 and 3, respectively) from GP before and after adding droupout and normalization layers.

Table 4 shows the comparison of the proposed method with the DENSER technique [Assunção et al. 2018], which is a state of the art approach to build CNN architectures. Results of DENSER shown in Table 4 are the values related to the original work, before using data augmentation and learning decay optimization, to be in the same conditions as the proposed approach. From the results, it can be observed that the proposed approach has competitive results when compared with DENSER approach, but with architecture using a lower number of layers and number of parameters. It means that our proposed technique was capable of evolving in a way that with less computational resources (considering that less parameters and layers result in less computational processing required [Zhou et al. 2016]), achieved results quite equivalents.

7. Conclusion and Future Work

This work proposes the use of Genetic Algorithm with Grammar-based Genetic Programming to create and optimize Convolution Neural Network architectures. Every generated

Table 4. Comparison of the best results obtained by different methods on the CIFAR-10 dataset.

Method	Accuracy (%)	# Parameters	# Layers
DENSER [Assunção et al. 2018]	88.41	10.81×10^6	18
Proposed	85.77	6.61×10^5	10

CNN was trained and tested on an image classification problem using CIFAR-10 dataset, and the quality of the architecture was measured by the accuracy obtained when executing the classification for the test dataset.

Results showed that we were able to produce architectures for CNNs that gradually evolved achieving higher accuracies for the given dataset. The architecture was evolved during the execution of the genetic algorithm, and the evolution analysis indicates that the use of grammar can produce promising results. In comparison to the complex grammar presented on DENSER [Assunção et al. 2018], our rather simple grammar was able to reach competitive accuracy to their best original architecture. We should also highlight the fact that our technique obtained its best results utilizing less number of parameters and layers than DENSER.

Future work involves the maturation of the grammar to include CNN variables not covered in this experiment, as well as, layers here studied, dropout and normalization, but not included in the grammar.

References

Ahmadizar, F., Soltanian, K., AkhlaghianTab, F., and Tsoulos, I. (2015). Artificial neural network development by means of a novel combination of grammatical evolution and genetic algorithm. *Engineering Applications of Artificial Intelligence*, 39:1–13.

Assunção, F., Lourenço, N., Machado, P., and Ribeiro, B. (2018). Evolving the topology of large scale deep neural networks. In *European Conference on Genetic Programming*, pages 19–34. Springer.

Chollet, F. et al. (2015). Keras. `https://keras.io`.

Deng, L., Hinton, G., and Kingsbury, B. (2013). New types of deep neural network learning for speech recognition and related applications: An overview. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8599–8603. IEEE.

Farfade, S. S., Saberian, M. J., and Li, L.-J. (2015). Multi-view face detection using deep convolutional neural networks. In *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*, pages 643–650. ACM.

Fenton, M., McDermott, J., Fagan, D., Forstenlechner, S., Hemberg, E., and O'Neill, M. (2017). Ponyge2: Grammatical evolution in python. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1194–1201. ACM.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. (2016). *Deep learning*, volume 1. MIT press Cambridge.

Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE.

Hingee, K. and Hutter, M. (2008). Equivalence of probabilistic tournament and polynomial ranking selection. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 564–571. IEEE.

Hopcroft, J. E. (2008). *Introduction to automata theory, languages, and computation*. Pearson Education India.

Koza, J. R. (1992). *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT Press.

Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436.

LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404.

Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440.

Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., et al. (2017). Evolving deep neural networks. *arXiv preprint arXiv:1703.00548*.

O'Neill, M. and Ryan, C. (2001). Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358.

O'Neil, M. and Ryan, C. (2003). Grammatical evolution. In *Grammatical Evolution*, pages 33–47. Springer.

Plis, S. M., Hjelm, D. R., Salakhutdinov, R., Allen, E. A., Bockholt, H. J., Long, J. D., Johnson, H. J., Paulsen, J. S., Turner, J. A., and Calhoun, V. D. (2014). Deep learning for neuroimaging: a validation study. *Frontiers in neuroscience*, 8:229.

Scherer, D., Müller, A., and Behnke, S. (2010). Evaluation of pooling operations in convolutional architectures for object recognition. In *Artificial Neural Networks–ICANN 2010*, pages 92–101. Springer.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Soni, N. and Kumar, T. (2014). Study of various crossover operators in genetic algorithms. *International Journal of Computer Science and Information Technologies*, 5(6):7235–7238.

Tsoulos, I., Gavrilis, D., and Glavas, E. (2008). Neural network construction and training using grammatical evolution. *Neurocomputing*, 72(1-3):269–277.

Zhang, J. and Zong, C. (2015). Deep neural networks in machine translation: An overview. *IEEE Intelligent Systems*, 30(5):16–25.

Zhou, H., Alvarez, J. M., and Porikli, F. (2016). Less is more: Towards compact cnns. In *European Conference on Computer Vision*, pages 662–677. Springer.