Designing Convolutional Neural Network Architecture Using Genetic Algorithms

Ashray Bhandare and Devinder Kaur
Department of EECS, the University of Toledo, Toledo, Ohio, USA

***Abstract*—In this paper, genetic algorithm (GA) is used to optimally determine the architecture of a convolutional neural network (CNN) that is used to classify handwritten numbers. The CNN is a class of deep feed-forward network, which have seen major success in the field of visual image analysis. During training, a good CNN architecture is capable of extracting complex features from the given training data; however, at present, there is no standard way to determine the architecture of a CNN. Domain knowledge and human expertise are required in order to design a CNN architecture. Typically architectures, The GA determine the exact architecture of a CNN by evolving the various hyper parameters of the architecture for a given application. The proposed method was tested on the MNIST dataset. The results show that the genetic algorithm is capable of generating successful CNN architectures. The proposed method performs the entire process of architecture generation without any human intervention.**

Keywords-Convolutional Neural Network; Genetic Algorithm; MNIST Data

I. INTRODUCTION

The idea that programmable computers will become intelligent was conceived over a hundred years before one was built. AI has tackled and solved many problems that are intellectually difficult for human beings but relatively straightforward for computers. Such problems are defined by a set of mathematical rules. The challenge for AI is to transform tasks which are easy and intuitive for humans into formal procedures that a computer can understand. For example, it is easy for humans to recognize a face, a piece of music even when the data is corrupted or incomplete.

With the advancements in big data, Graphical Processing Unit (GPU) technology and algorithms there has been a lot of progress in the field of Deep Learning. Deep Learning is part of machine learning techniques and allows a machine to learn with experience and data. It makes use of artificial neural networks with more than one hidden layer. By implementing more layers and more neurons within a layer, it allows the network to understand complex ideas by building upon simpler ones. For example, a deep network can build the concept of an image of a car by combining simpler concepts, such as edges, corners, contour, and object parts [1].

Convolutional Neural Network (CNN) is one such type of deep networks. Yann LeCan carried out one of the first exercises on CNN. He taught a computer system how to recognize the differences between handwritten digits [2]. When the system chose incorrectly, he would correct it until the program figured out the mathematical operation called convolution. Convolution is a specialized kind of linear operation. Unlike conventional neural networks, CNN's use this linear operation to obtain an intermediate output (feature) before

DOI: 10.21307/ijanmc-2021-024

using it as an input for the next layer. This is done in at least one of their layers. A typical CNN architecture consists of layers such as convolution layer, pooling layer, and fully connected layer. Each of these layers consists of hyper-parameters that are chosen by researchers using new theoretical insights or intuition gained from experimentation. In this paper, we achieved the following objectives:

Automate the process of CNN architecture selection.

Achieve the architecture by evolving the hyper parameters of CNN using Genetic Algorithm (GA)

Discover CNN architectures without any human intervention that perform well on a given machine-learning task.

GA is inspired by biological evolution, used to find globally optimal solutions and makes use of genetic operators such as selection, crossover, and mutation.

The goal of the proposed algorithms is to discover CNN architectures that perform well on a given machine-learning task with no human intervention. Over the course of many generations, Genetic Algorithm picks out the layers and hyper-parameters to choose from, the algorithm is left with a finite but large space of model architectures to search from. It learns through random exploration and slowly begins to exploit its findings to select higher performing models. It receives the testing accuracy as a means of comparison between architectures and ultimately selects the best architecture. The entire process called an evolutionary experiment goes on for many generations until a fully trained suitable CNN model is generated.

This paper is organized as follows. In Section 2, the dataset used for this analysis is introduced. Section 3 presents the mathematical model of CNN. GA is explained in section 4. Section 5 talks about our proposed method to generate CNN architectures using GA. In Section 6, experiments and results are presented. Conclusions are presented in section 7.

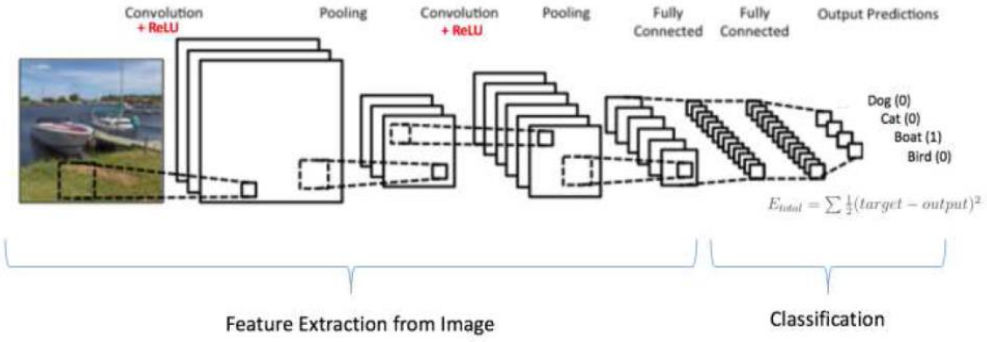

Figure 1. An example of CNN architecture [10]

II. MNIST DATASET

The MNIST Dataset is scanned images of handwritten digits and the associated labels describe which digit 0-9 is contained in each image. The “NIST” stands for National Institute of Standards and Technology, the agency that originally collected this data. The “M” stands for “modified,” since the data has been preprocessed for easier use with machine learning algorithms.

The data set consists of 50,000 labeled samples of handwritten digits which is to be used as training data. It also consists of an extra 10,000 images that are unlabeled and used as testing data.

It is one of the popular datasets as it allows researchers to study their proposed methods in a controlled environment. In our case, we will discover the best CNN architecture that classifies this data set using genetic algorithm (GA).

III. Convolutional Neural Network

In machine learning, a CNN is a type of feed-forward artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex [3]. This idea was expanded upon by a fascinating experiment by Hubel and Wiesel in 1962 where they showed that some individual neuronal cells in the brain responded (or fired) only in the presence of edges of a certain orientation [4]. For example, some neurons respond when exposed to vertical edges and some respond when shown horizontal or diagonal edges. When all these neurons are arranged in a tilled manner, they were able to produce visual perception. The idea that different neurons in the visual cortex look for different features are the inspiration behind a CNN.

In this paper, we use CNN for the task of image classification. CNN take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way [5]. In particular, the layers of a CNN have neurons arranged in three dimensions: width, height, depth. (Note that the word depth here refers to the third dimension of an image, not to the depth of a full Neural Network, which can refer to the total number of layers in a network). The neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner.

A. Layers in a CNN

A simple convolutional neural network is a sequence of layers. A CNN takes an image and passes it through a series of convolutional, nonlinear (activation), pooling (down sampling), and fully connected layers to get an output [5] [6]. This output can be a single class or a probability of classes that best describe the image. An example of a CNN architecture is shown in Figure 1. Each of these layers is explained in the following subsections.

1) Convolutional layer

A convolutional layer is a core building block of any CNN architecture. It is always the first layer of the architecture. The input of this layers will always be a three-dimensional object (eg. 400x400x3). The best way to understand convolution is to imagine a window of significantly lesser size (eg. 5x5) is being moved across all the areas of the input. In machine learning terms, this window is called a filter or neuron or kernel. Now, this filter is also an array of numbers (the numbers are called weights or parameters).

Figure 2 illustrates the process of convolution on an image of dimensions 5x5x3. The filter used to convolve over this image is of the dimensions 3x3x3. It can be noted that the depth of the filter is same as the depth of the image. The output generated after the process of convolution is calculated by performing elementwise multiplication. These multiplications across the width, height and depth of an image with the filter are all summed up in order to get a single value. Finally, the single value is averaged over the total number of values in the filter. It is computed as follows:

$$Output = \frac{\begin{array}{c}[(0*0)+(-1*0)+(1*1)+(0*0)+\\(0*0)+(-1*-1)+(0*2)+(1*0)+\\(1*2)]+[(-1*2)+(0*1)+(1*1)+\\(1*2)+(1*1)+(1*0)+(0*0)+\\(0*0)+(0*2)]+[(-1*0)+(-1*0)+\\(1*1)+(-1*1)+(0*0)+(1*0)\\+(0*1)+(0*1)+(0*1)]\end{array}}{27} = \frac{2+2-1}{27} = 0.111$$

Now, we repeat this process for every location on the input volume. Every unique location on the input volume produces a number. After sliding the filter over all the locations, we are left with a two-dimensional array which is called an activation map or feature map.

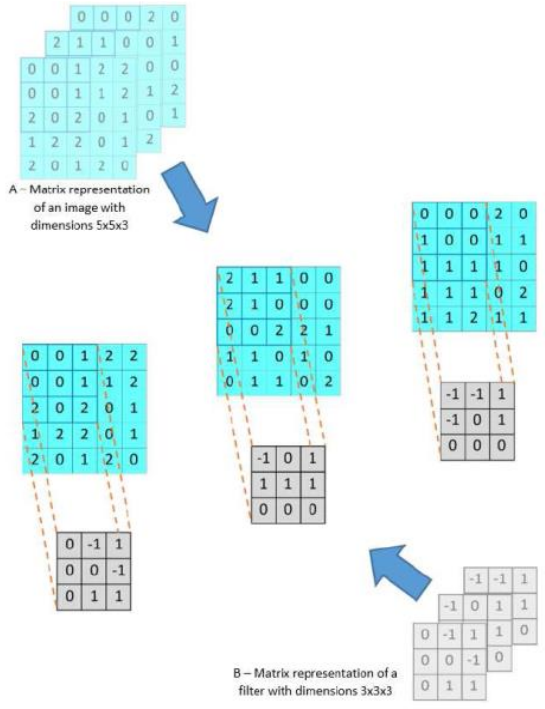

Figure 2. The process of convolution on the left-hand corner of an image

The size of this feature map is governed by three parameters.

These parameters need to be decided before convolution is performed. They are as follows:

- **Depth:** The number of filters that convolve over the same image to get different two-dimensional feature maps corresponds to the depth of the output after the convolutional layer.
- **Stride:** The step by which a filter is moved across an image, is called stride.
- **Zero-padding:** The process of padding zeros around an input image is called as zero-padding. This is done so that we can slide the filter to the bordering elements of the input image matrix. This parameter allows us to control the size of the generated feature map.

2) Non Linearity (Activation Layer)

Most of the real world datasets are nonlinear in nature. The process of convolution (element-wise matrix multiplication and addition) is a linear operation. Due to this, there are no nonlinear properties in our network. In order to introduce nonlinearity in our network, the output feature maps obtained after the process of convolution in the convolutional layer is passed through a nonlinear function.

One of the most popular nonlinear operation is the ReLU operation. Other nonlinear functions such as tanh or sigmoid can also be used instead of ReLU, but ReLU has been found to perform better in most situations.

3) Pooling Layer

Pooling is a way to take large images and shrink them down while preserving the most important information in them. It is common to periodically insert a Pooling layer in-between successive Convolutional layers in a CNN architecture. Maxpooling is one the most popular types of pooling. Maxpooling consists of stepping a small filter across an image and taking the maximum value from the filter at each step. The stride by which the filter steps across the image is usually the same as that of filter size.

The process of pooling has two main advantages. The first is that the amount of parameters or weights is reduced by 75%, thus lessening the computation cost. The second is that it will control overfitting as it reduces the number of parameters and computations in the network.

4) Fully Connected Layer

The Fully Connected layer is a traditional Multi-Layer Perceptron layer that uses a softmax activation function in the output layer. The softmax activation function generates the outputs in the range of 0 and 1. The output of the softmax function is equivalent to a categorical probability distribution, it tells you the probability that any of the classes are true.

B. Training using Backpropagation

As we have seen from the subsections above, the convolution and pooling layers act as Feature Extractors from the input image in a CNN architecture while a fully connected layer acts as a classifier generating probabilities for the different classes. This is the forward propagation step.

Now we train the dataset using traditional back-propagation until it has converged and we obtain high classification accuracy. This classification accuracy will be used as a fitness value when the architecture is passed through the GA tuner. This is further explained in the following sections.

IV. GENETIC ALGORITHM

Genetic Algorithms were invented to mimic some of the processes observed in natural evolution. Many people, biologists included, are astonished that life at the level of complexity that we observe could have evolved in the relatively short time suggested by the fossil record [7]. The idea with GA is to use this power of evolution to solve optimization problems. Genetic Algorithms (GAs) are adaptive heuristic search algorithm based on the evolutionary ideas of natural selection and genetics [8]. As such, they represent an intelligent exploitation of a random search used to solve optimization problems. Although randomized, Gas are by no means random, instead, they exploit historical information to direct the search into the region of better performance within the search space.

A. GA Overview

Gas simulates the survival of the fittest among individuals over a consecutive generation for solving a problem. Each generation consists of a population of character strings that are analogous to the chromosome that we see in our DNA. Each individual represents a point in a search space and a possible solution.

The individuals in the population are then made to go through a process of evolution based on the following foundations [8]:

- Individuals in a population compete for resources and mates.
- Those individuals most successful in each ‘competition’ will produce more offspring than those individuals that perform poorly.
- Genes from good individuals propagate throughout the population so that two good parents will sometimes produce offspring that are better than either parent.
- Thus each successive generation will become more suited to their environment.

A population of individuals is maintained within a search space for a GA, each representing a possible solution to a given problem. Each individual is coded as a finite length vector of components, or variables, in terms of some alphabet, usually the binary alphabet {0,1}. These individuals are analogous to chromosomes and the

variables are analogous to genes. Thus a chromosome (solution) is composed of several genes (variables). A fitness score is assigned to each solution representing the abilities of an individual to compete. The individual with the abilities of an individual to compete. The individual with the optimal (or generally near optimal) fitness score is sought. The GA aims to use selective breeding of the solutions to produce offspring better than the parents do by combining information from the chromosomes. Figure 3 illustrates a population of chromosomes. In this figure, each chromosome consists of 8 genes.

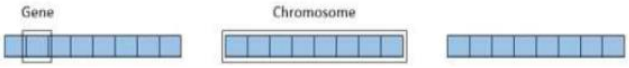

Figure 3. Population of three chromosomes

Parents are selected to mate, on the basis of their fitness, producing offspring via a reproductive plan. Consequently, solutions with higher fitness are given more opportunities to reproduce so that offspring inherit characteristics from each parent. As parents mate and produce offspring, room must be made for the new arrivals since the population is kept at a static size. Individuals in the population die and are replaced by the new solutions, eventually creating a new generation once all mating opportunities in the old population have been exhausted. In this way, it is hoped that over successive generations better solutions will thrive while the least fit solutions die out [7].

New generations of solutions are produced containing, on average, more good genes than a typical solution in a previous generation. Each successive generation will contain better 'partial solutions' than previous generations. Eventually, once the population has converged and is not producing offspring noticeably different from those in previous generations, the algorithm itself is said to have converged to a set of solutions to the problem at hand.

B. Genetic Operators

A genetic operator is an operator used in genetic algorithms to guide the algorithm towards a solution to a given problem. There are three main types of operators

- Selection
- Crossover
- mutation

These operators must work in conjunction with one another in order for the algorithm to be successful. Genetic operators are used to creating and maintaining genetic diversity (mutation operator), combine existing solutions (also known as chromosomes) into new solutions (crossover) and select between solutions (selection).

1) Selection

Selection operators give preference to better solutions (chromosomes), allowing them to pass on their 'genes' to the next generation of the algorithm. The best solutions are determined using some form of objective function (also known as a 'fitness function' in genetic algorithms), before being passed to the crossover operator [8]. Different methods for choosing the best solutions to exist, for example, Roulette wheel selection and tournament selection; different methods may choose different solutions as being 'best'. The selection operator may also simply pass the best solutions from the current generation directly to the next generation; this is known as elitism or elitist selection.

2) Crossover

Crossover is the process of taking more than one parent solutions (chromosomes) and producing a child solution from them. By recombining portions of good solutions, the

genetic algorithm is more likely to create a better solution. As with selection, there are a number of different methods for combining the parent solutions, for example, single-point crossover and two-point crossover [8].

Figure 4 and Figure 5 illustrate the concept of single-point and two-point crossover respectively.

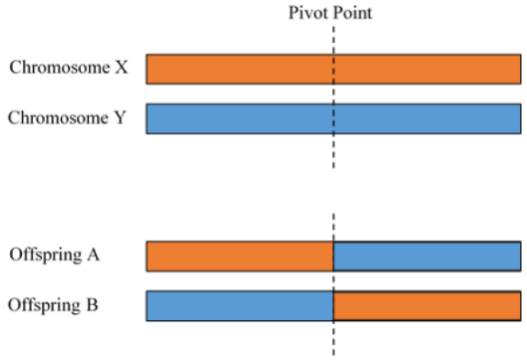

Figure 4. Single-point crossover

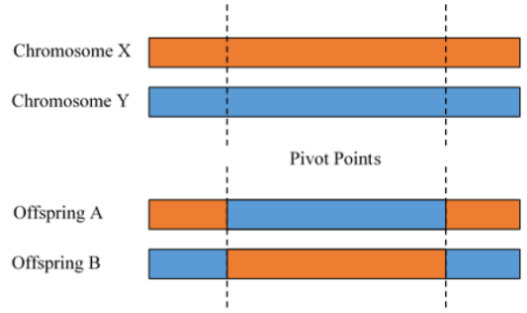

Figure 5. Two-point crossover

3) Mutation

The mutation operator encourages genetic diversity amongst chromosomes and attempts to prevent the genetic algorithm converging to a local minimum by stopping the chromosomes becoming too close to one another. In mutating the current pool of chromosomes, a given chromosome may change entirely from the previous chromosome. By mutating the chromosomes, a genetic algorithm can reach an improved solution solely through the mutation operator [8]. Again, different methods of mutation may be used; these range from a simple bit mutation (flipping random bits in a binary string chromosome with some low probability) to more complex mutation methods, which may replace genes in the chromosomes with random values chosen from the uniform distribution or the Gaussian distribution.

V. PROPOSED METHOD

Before training a deep neural network, it is necessary to determine the architecture of that network which is in turn done by choosing the hyper parameters associated with each layer of the network. Usually, the hyper parameters are determined by human intuition, experience or trial, and error.

Table 1, shows all the hyper-parameters associated with the deep neural network and their ranges which are used for experiments in this paper. The focus of this paper is to use genetic algorithm to automatically determine these hyper-parameters for best performance.

TABLE 1. THE VARIOUS HYPER PARAMETERS IN CNN WITH THEIR RANGES

Hyper parameter	Range
No. of Epoch	(0 - 127)
Batch Size	(0 - 256)
No. of Convolution Layers	(0 - 8)
No. of Filters at each Convo layer	(0 - 64)
Convo Filter Size at each Convo layer	(0 - 8)
Activations used at each Convo layer	(sigmoid, tanh, relu, linear)
Maxpool layer after each Convo layer	(true, false)
Maxpool Pool Size for each Maxpool layer	(0 - 8)
No. of Feed-Forward Hidden Layers	(0 - 8)
No. of Feed-Forward Hidden Neurons at each layer	(0 - 64)
Activations used at each Feed-Forward layer	(sigmoid, tanh, softmax, relu)
Optimizer	(Adagrad, Adadelta, RMS, SGD)

When tuning the hyper-parameters of a CNN architecture using genetic algorithm, the most crucial step is the problem representation. In other words, the problem should be formulated in such a way that it is suitable for the genetic algorithm. The variables involved in this tuning process are the various CNN hyper-parameters. Hyper-parameters, which are tuned to their optimal values, will generate the best CNN architecture and provide the highest classification and prediction accuracy for the given MNIST dataset.

For the purposes of experimentation, a direct-encoding representation of the hyper-parameters is performed. Here the value of hyper-parameters is extracted from a GA chromosome, which is in the binary format. Figure 6 shows a random GA architecture in the binary format.

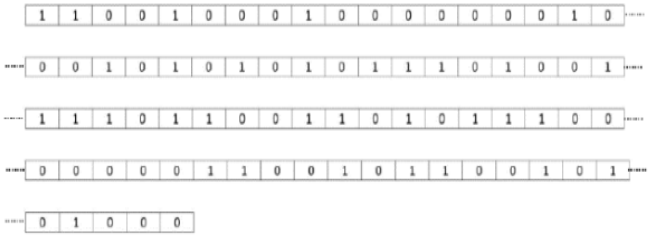

Figure 6. Representation of a GA chromosome

Figure 7 shows an illustration of the problem encoding process. The figure represents a random CNN architecture is obtained from a GA chromosome. Each hyper-parameter is shown with its corresponding binary interpretation.

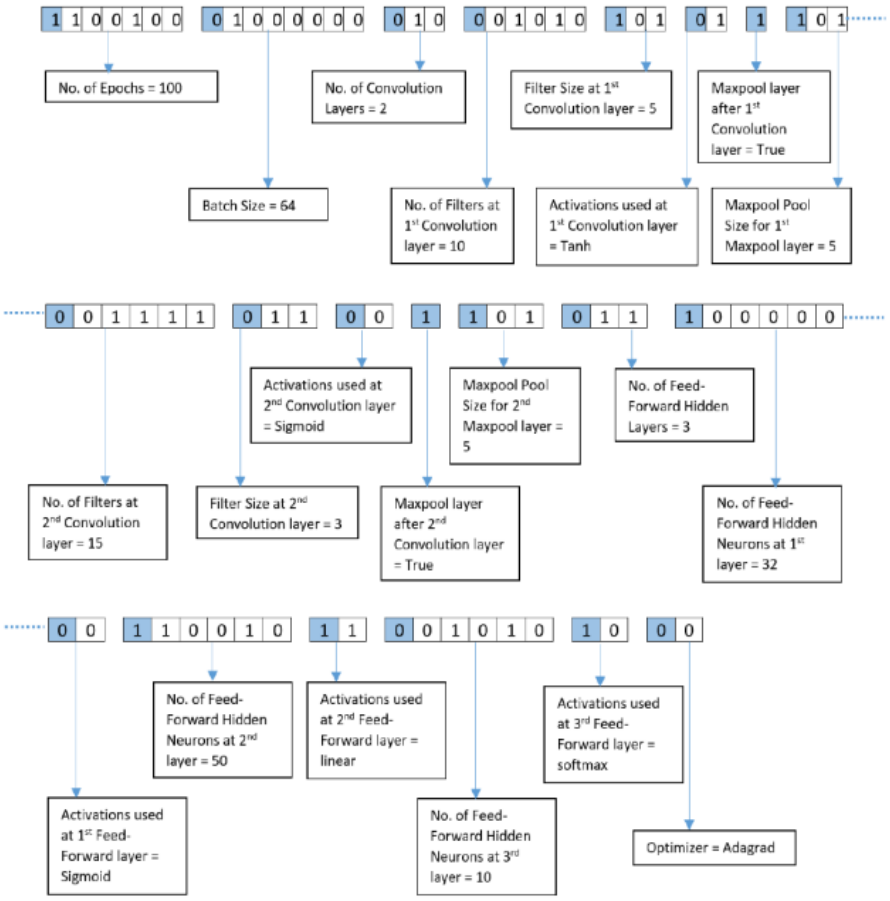

Figure 7. Representation of the hyper-parameters in binary format

A. *Fitness Evaluation*

In this study, a population size of 10 chromosomes (CNN architecture) is chosen. The fitness of each chromosome is evaluated using a fitness function.

The fitness function used in this study is the classification accuracy, which determines the number of correctly classified digits in the MNIST dataset. In section 3 of this paper, it is seen how a CNN architecture can be trained to obtain its best classification accuracy. This classification accuracy (ranges between 0 and 1) is the fitness value of a particular CNN architecture.

VI. EVALUATION

In this section, the proposed GA hyper-parameters tuning technique is used to obtain the best CNN architecture for the MNIST dataset.

In this paper tournament selection, single-point crossover and multiple point mutation are the genetic operations performed on the chromosomes. The parameters of the chosen genetic operations are listed in Table 2.

TABLE 2. PARAMETERS OF THE GENETIC OPERATIONS

Parameters	Value
Tournament selection size	2
Crossover Probability	50%
Mutation probability	80%
Genes Mutated	10%

A. *Experimental Setup*

The Genetic Algorithm based CNN architecture generator has been implemented in python 2.7. Tests were carried on an AWS instance running Ubuntu Server 16.04 LTS.

This instance is powered by an AWS-Specific version of Intel's Broadwell processor, running at 2.7 GHz. It incorporates up to 8 NVIDIA Tesla K80 Accelerators, each running a pair of NVIDLA GK210 GPUs. Each GPU provides 12 GiB of memory (accessible via 240 GB/second of memory bandwidth), and 2,496 parallel processing cores [9].

The Genetic algorithm tuner was implemented with the MNIST dataset with 50,000 images as its training set and another 10,000 images as its testing set. The aim of the tuner was to generate a CNN architecture with the best architecture such that the classification accuracy of the testing set is very high.

VII. RESULTS

Genetic algorithm with 10 chromosomes generated randomly was executed 10 times, each time with randomly chosen chromosomes. It was noted that each run of GA tuner gave the best chromosome which represented a CNN architecture with an accuracy of greater than 90%. This can be seen in Table 3. However, the best of the 10 runs gave an accuracy of 99.2%. This particular architecture was chosen.

TABLE 3. HIGHEST FITNESS VALUES OBTAINED DURING EACH OF THE 10 EXPERIMENTS

Exp. No.	Highest Fitness Value
1	0.984499992943
2	0.973899998105
3	0.988800008184
4	0.991900001359
5	0.947799991965
6	0.949000005102
7	0.983099997652
8	0.979799999475
9	0.956399999567
10	0.972350000068

The generated output after GA tuning in shown in Figure 8

```
all time best score: 0.991900001359
- code -
--------------------
{'batch_size': 1,
 'convo_activations': ['relu', 'tanh', 'sigmoid', 'tanh'],
 'convo_dropouts': [0.41, None, 0.45, 0.15],
 'dense_activations': ['softmax',
                       'sigmoid',
                       'relu',
                       'softmax',
                       'sigmoid',
                       'softmax',
                       'sigmoid'],
 'dense_dropouts': [0.27, 0.37, 0.26, 0.4, None, 0.06, 0.34],
 'dense_hidden_neurons': [21, 46, 0, 5, 52, 26],
 'loss': 'categorical_crossentropy',
 'maxpools': [True, False, True, True],
 'nb_conv': [5, 1, 5, 3],
 'nb_convo_layers': 4,
 'nb_dense_layers': 7,
 'nb_epoch': 102,
 'nb_filters': [0, 27, 13, 62],
 'optimizer': 'sgd',
 'pool_sizes': [4, None, 7, 1]}
score: 0.991900001359
--------------------
```

Figure 8. Generated CNN architecture after GA tuning

Figure 8 shows the best values for hyper-parameters shown in Table 2 reached by the algorithm.

VIII. CONCLUSION

In this paper, we carried out an experiment to see if genetic algorithm can be used to automatically generate good CNN architectures without any human intervention. The basic techniques of the Gas are designed to simulate processes in natural systems necessary for evolution and; especially those that follow the principles first laid down by Charles Darwin-"survival of the fittest."

Our simulation results for finding the best CNN architecture using GA to tune the hyper-parameters lead to the generation of architecture which yielded an accuracy rate of more than 90% for the classification of the MNIST dataset. The best of the 10 runs gave an accuracy of 99.2%.

Therefore, it can be concluded that GA have the potential of generating successful CNN architectures automatically.

REFERENCES

[1] Y. B. a. A. C. Ian Goodfellow, Deep Learning, 2016.

[2] Y. J. L. D. B. B. D. J. S. G. H. P. G. I. H. D. H. R. E. a. H. W. LeCun, "Handwritten digit recognition:Applications," IEEE Communications Magazine, pp. 41-46, 1989.

[3] M. Matusugu, M. Katsuhiko, M. Yusuke and K. Yuji, "Subject independent facial expression recognition with robust face detection using a convolutional neural network," Neural Networks, vol. 16, pp. 555-559, 2003.

[4] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," The Journal of Physiology, vol. 195, pp. 218-243, 1968.

[5] A. karpathy, "CS23In Convolutional Neural Networks for Visual Recognition," 2016. [Online]. Available: http://cs23In. github.io/convolutional-networks/.

[6] ujjwalkarn, "An Intuitive Explanation of Convolutional Neural Networks," 11 August 2016. [Online]. Available:https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/.

[7] "Genetic Algorithms," [Online]. Available: https://www.doc.ic.ac.uk/~nd/surprise_96/journal/voll/hmw/article.html#top.

[8] L. N. d. Castro, "Genetic Algorithms," in Fundamentals of Natural Computing; Basic Concepts, Algorithms, and Applications, Chapman and Hall/CRC, 2006, pp. 88-91.

[9] J. Barr, "New P2 Instance Type for Amazon EC2-Up to 16 GPUs," 29 September 2016 [Online]. Available: https://aws.amazon.com/blogs/aws/new-p2-instance-type-for-amazon-ec2-up-to-16-gpus/.

[10] A. Gibiansky, "Convolutional Neural Networks," 24 February 2014. [Online]. Available: http://andrew.gibiansky.com/blog/machine-learning/convolutional-neural-networks/.