Evolution of Convolution Neural Network Architectures using Genetic Algorithm

Aadi Swadipto Mondal
Electronics and Electrical Communication Engineering
Indian Institute of Technology (IIT) Kharagpur
West Bengal, India
Email: aadismondal@iitkgp.ac.in

***Abstract*—Convolution Neural Network (CNN) has become a remarkable tool in solving many real-world problems of computer vision in recent years. However, the designing of most of CNN architectures is manual, which requires a significant trial and error methodology as CNN's performance highly relies on their architectures. Thus, it gets arduous to design a promising CNN architecture without having sufficient domain knowledge and human expertise.**

In this paper, we attempt to explore the possibility of using Genetic Algorithm to design CNN architectures automatically. The CNN architecture proposed by the Genetic Algorithm is trained from scratch using Gradient- Descent Algorithm and evaluated on a validation set at each evolutionary step. The algorithm does not require any preprocessing of the data before its execution, nor any post-processing on the evolved CNN architecture. We propose an encoding scheme for determining the layer connectivity. This scheme allows the formation of skip connections within the CNN architecture. Along with the encoding scheme, the filter dimensions, and the number of nodes in the fully-connected layer are also genetically evolved during the subsequent evolution using standard genetic operators, namely selection, mutation, and crossover. We have specified algorithms for the formation of the CNN model by combining the information about the encoding scheme, filter dimensions, and the number of nodes of the fully-connected layer in addition to the genetic operations.

The proposed algorithm is tested on the MNIST dataset for handwritten digit recognition and Fashion-MNIST dataset. Our experiments have shown that the algorithm is capable of successfully generating high-quality CNN architectures, which are less studied before.

***Keywords*—Genetic Algorithm, Convolution Neural Networks, Skip Architecture**

I. Introduction

Visual Recognition has become an essential task for computer-aided vision systems, which is majorly achieved by the application of convolution neural networks. Modern day's image recognition task has become more intricate, which requires more sophisticated CNN architectures. The first version of CNN was first introduced in 1998 as LeNet5[1] with six layers. With time, more complicated models like AlexNet[2], VGGNet[3], and ResNet[4], are developed. Although they emerge to be very efficient in several image recognition tasks, we note that their architectures are manually designed which requires experts having rich domain knowledge both in investigated data and CNN architecture design. This limits the resilience of the approach as several design parameters like filter parameters, connectivity of the layers, and the number of nodes in the hidden layers of the fully connected layers have to be managed.

In our work, we aim to propose an automatic design algorithm of the CNN architecture using Genetic Algorithm. Our algorithm accepts intervals for different parameters namely kernel dimensions, kernel numbers, and padding information for each convolution layer and the number of nodes in the hidden layers of the fully connected layers along with pooling information for each convolution layer. Even within these intervals, the search space is massive which inspires us to use the Genetic Algorithm to traverse the search space adroitly,

Deeper neural networks have greater accuracy for more complicated problems. On the contrary, increasing the number of layers leads to saturation of accuracy and then degradation, well known as the degradation problem. It turns out that sufficiently deep neural networks may fail to learn simple functions. To solve this problem, skip architectures are introduced. In our algorithm, we allow one layer to receive input from one or more previous layers and perform convolution operations on them to match up the spatial dimensions. An element-wise average is performed on all the outputs of the convolution operations to form that layer after which pooling is applied.

The remaining paper is organized as follows. Section II refers to the preliminaries. Section III illustrates the details of our algorithm. Experiments and results are noted in Section IV. Finally, the paper is concluded in Section V in addition to the scopes of future work.

II. Preliminaries

A. Convolution Neural Networks

CNN is a hierarchical model comprising of several building blocks, i.e., the convolution layer, pooling layer, merge layers, and fully connected layers. Deep neural networks try to approximate a function by application of these layers.

Convolution layers employ filters to perform convolution operation on the input feature map. Filters are 3-D matrices. In 2-dimension convolution, filters have a specific length and breadth F. The depth of the filters is equal to the number of input feature maps D. The input feature map is of dimension L and B. The stride of the convolution is taken to be one

978-1-7281-6929-3/20/$31.00 ©2020 IEEE

as generally used in literature. Then, convolution can be summarized by the following equation.

$$\overline{I}(i,j) = \sum_{z=0}^{D-1} \sum_{y=0}^{F-1} \sum_{x=0}^{F-1} I(i+x, j+y, z)\overline{K}(F-x-1, F-y-1, z) + \overline{B}(i,j)$$

From the above equation, the maximum range of $i+x$ is $L-1$, which concludes that i extends from 0 to $L-F$ and the size of the output feature maps results in $(L-F+1, B-F+1, W)$ where W denotes the number of filters.

Zero padding refers to the process of symmetrically adding zeros to the input feature map. This is purposefully used to keep information from the edges. The following equations summarize padding.

$$I(i,j) = 0 \quad 0 \leqslant i \leqslant L+2P-1 \quad j = 0, B+2P-1$$
$$I(i,j) = 0 \quad 0 \leqslant j \leqslant B+2P-1 \quad i = 0, L+2P-1$$

Thus, we can modify the dimensions of the output feature maps to $(L-F+2P+1, B-F+2P+1, W)$.

Pooling layers progressively reduce the number of parameters and computation in the network by reducing the spatial size of the representation. Pooling layer independently operates on each feature map. Several types of pooling layers are used in the literature. We will use max-pooling throughout the paper. Let the pool filter dimension be K. Following equations summarize pooling layers.

$$S(i,j) = max(I(K.i+x, K.j+y)) \quad 0 \leqslant x, y \leqslant K-1$$

From all the above equations, we can determine the ultimate dimensions of the output feature maps.

Output-Length = Integral part of $(L-F+2P+1)/K$
Output-Width = Integral part of $(B-F+2P+1)/K$
Output-Depth = W (no. of filters)

After each convolution operation, the feature maps are passed through a non-linear differentiable function, regularly known as the activation function. In our case, we will use Rectified-Linear Unit (ReLU)[5] $f(x) = max(0, x)$.

Also, fully-connected layers are annexed to the tail of the CNN. We will use multilayer perceptron, which is a non-linear mapping between input and output vector. The nodes are interlinked by weights after being passed through a non-linear differentiable activation function. We will use the sigmoid function[6] $\frac{1}{1+e^x}$ here. The last layer of the convolution layers is flattened into a column matrix and then taken as the input for the fully-connected layer.

$$y = \overline{W}.x + \overline{b}$$

where $\overline{W}_{(m \times n)}$ is the weight matrix, $y_{(m \times 1)}$ is the output vector, $x_{(n \times 1)}$ is the input vector and $\overline{b}_{(m \times 1)}$ is the bias vector.

In CNN, the convolution layers before the fully connected layers hold information regarding local features of the image such as edges, blobs, shapes, etc. Each convolution layer holds several filters that represent one of those features. The fully connected layer contains composite and aggregated information from all the convolution layers that are of paramount significance.

With the advent of the Graphics Processing Unit (GPU), complex neural networks can be trained, containing some millions of parameters. As we have to conduct training of several architectures during the evolution, we will perform our experiments only on smaller datasets like MNIST dataset for handwritten digit recognition, and Fashion-MNIST. Although, our algorithm is easily expandable for complex problems requiring millions of trainable parameters.

B. Skip Connection

Skip connections[7] connect two non-adjacent layers of a CNN. The skip connections were first introduced as a gate mechanism in training recurrent neural networks with long and short-term memory[8] for avoiding the Gradient Vanishing (GV) problems. GV problems refer the gradient to be very small or exploding while back propagating through the layers of the deep neural network. As most of the deep neural network algorithms are based on gradient descent algorithm, GV problems have become a significant hindrance in training deep neural networks. Adopting skip connections shorten the number of layers of back-propagation, which allows the deep neural network to have the leverage of having a higher number of layers and also getting trained on low computational resources A typical skip connection looks like in Figure 1. A very modern application of skip connections is noticed in

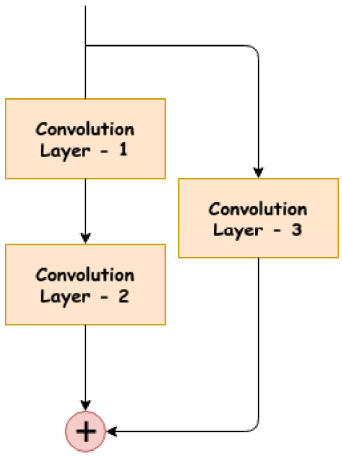

Fig. 1. Skip Connection

ResNet[4], applied as an identity layer $F(x) = f(x) + x$. Identity layers connect two non-adjacent layers without any convolution operations. The major drawback of this approach is that the two layers should have the same spatial dimensions. To overcome this hurdle, we introduce a convolution operation, so that the spatial dimensions can be matched. In another recent approach[9], layers interconnect within the same stage, which requires all the layers within the same stage to have the same spatial dimensions. Our approach also overcomes this problem.

C. Genetic Algorithm

Genetic Algorithms (GA) are a class of meta-heuristic algorithms inspired by the process of natural selection, according to Darwin's theory of evolution. They belong to the larger class of evolutionary algorithms. Genetic Algorithms work on two basic requirements:

- An encoding strategy to denote candidates in the search space of the optimization problem.
- An evaluation strategy for those candidates to compare among themselves.

A simple example would be solving the Knapsack problem[10] where it aims at finding the items to include where each item has a particular weight and cost so that the total weight is less than a specific value and the total cost is maximum for the included items. GA does not perform an analytic search, thus providing a near optimal solution. The near-optimal solution is achieved using much less computational resources. Hence, this algorithm is extensively used where the search space is massive, or there is no specific greedy approach to find the optimal solution. To the best of our knowledge, both the factors apply to our problem.

Genetic Algorithm allows the evolution of individuals using five genetic operators namely - Initialization, Evaluation, Crossover, Selection and Mutation.

1) Initialization: Initialization is the process of randomly selecting some candidates in the search space, usually distributed over the entire search space (for enclosed search spaces). The extent of the search space generally decides the number of candidates. In our algorithm, we take it as an input from the instruction set. To ensure the uniform distribution of candidates on the search space, we use Gaussian distribution.

2) Evaluation: Evaluation is the process of recognizing how close a candidate (chromosome) is to the optimal solution. Each chromosome is evaluated using a fitness function. Here, our chromosome denotes the architecture of the CNN. The fitness value is the accuracy of the CNN on a specific validation set after being trained using back-propagation algorithm.

3) Crossover: Crossover is the process of combining genetic information from parents chosen from the current population to produce more enriched population set. New off-springs are produced by exchanging or varying genetic information from the selected set of parents. There are many types of crossover algorithms. Multi-Point crossover[11] is one such algorithm, used in our approach, where alternating segments of the parents are swapped to produce new off-springs.

4) Selection: Selection is the process of eliminating weaker individuals in each evolution step. In each epoch, some candidates are discarded according to the evaluation score provided by the fitness function. Since, our problem is a maximization problem, where we aim to maximize the accuracy of the CNN architecture on the validation set, we will discard chromosomes with low accuracy in each evolution step.

5) Mutation: Mutation denotes the random alteration of some parts of the chromosome within the boundaries of the search space to produce required variation within the population set and maintain diversity. In mutation, the solution may differ entirely from the previous solution. Generally, the percentage of mutation is meagre so as not to revamp the population set much.

In many cases, we use a strategy known as Elitist strategy[12] where we copy a small proportion of the fittest individuals without any alteration. It ensures that the GA does not waste time in re-discovering previously discarded partial solutions. This enriches it's performance dramatically. Candidate solutions remain eligible for selection as parents in the next evolution step when preserved through elitism.

III. The Proposed Algorithm

This section presents the framework of the proposed algorithm. Our first objective is to provide an encoding scheme to represent the CNN architectures. Next, we provide algorithms for the genetic operators which allow efficient traversal of the search space.

We only encode chromosomes to denote the architecture of the convolution neural networks (convolution layers and fully connected layers). Combining the information from the encoding strategy and layer dimension sections of the chromosome, we design the CNN architecture and evaluate it on the validation set. The accuracy is noted and used as the fitness value of the chromosome.

The genetic algorithm traverses the search space to maximize the accuracy of the predicted CNN on the validation set. The intervals provided in the instruction set defines the boundaries of the search space.

A. Encoding Strategy

Each gene consists of three parts:

- Structure encoding information for each convolution layer
- Convolution information for each convolution layer
  - Number of filters
  - Kernel/Filter dimension
  - Padding dimension
- Number of nodes in each dense layer (except the first and last layer)

First, the layer dimensions are calculated using Algorithm 2. The formation of the layer connection and convolution filters follows it, using Algorithm 1. At last, the fully connected layers are formed.

1) Structure Encoding of Convolution Layers: We present a binary encoding strategy to determine the connectivity among the convolution layers. A series of binary numbers denotes the presence of a connection between two convolution layers. A convolution layer/block consists of several convolution operations on the output feature maps of the previous layers followed by the averaging of those convolution outputs and pooling on them. The input convolution layer is considered as the input image without any modifications. The i^{th} convolution layer can accept input from any of it's i previous layers. For instance, the third convolution layer can accept input from the input, first and second convolution layers. Thus, an i-bit binary

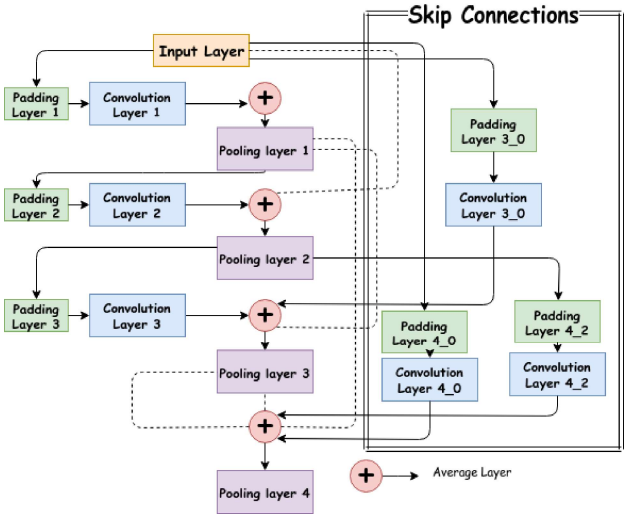

Fig. 2. A typical encoding procedure for 4 layer gene 1-01-101-1010. Here, the first layer takes input from the input layer. The second layer has the first bit 0. So, a dotted line signifies the possible connection. Second bit is 1 thus it takes input from the first layer. Similarly the third layer takes input from the input and second layers. For the fourth layer, note that, it does not take input from the immediately previous third layer. It only takes input from the input and second layers as the encoding for the 4^{th} layer is 1010.

Algorithm 1 Layer Connection Algorithm

for l in layer-bit-info **do**
  for b in l **do**
    if b is 1 **then**
      if input layer dimension = current layer dimension **then**
        Connect them as identity layer
      else
        Add the padding specified in the chromosome
        Find the kernel size $F = L - \overline{L} + 1 + 2P$
        while K is non-positive **do**
          $P = P + 1$
          $F = L - \overline{L} + 1 + 2P$
        end while
        Set depth $D =$ Depth of the Input Layer
      end if
    end if
  end for
  Place the padding P on the input feature map.
  Perform the convolution with kernel (F,F,D).
  Average the outputs of all convolution operations.
  Pool as specified in the instruction set.
end for

code encodes its information of connectivity. Among i bits, at least one has to be positive to ensure that each convolution layer takes input from minimal one layer.

For n convolution layers, the total number of bits is $1+2+3+..+n = \frac{n(n+1)}{2}$. When there is a positive bit, the convolution layer dimensions are compared. The layer from where input is taken, dimensions are considered after pooling (except for the input layer), but not for the current layer. If dimensions mismatch, padding, as specified in the chromosome, is applied, followed by the convolution operation. For a match in the dimensions, that layer is connected as an identity layer without any convolution operation. The filter dimension is calculated from the formula $L - \overline{L} + 1 + 2P$ where L is the input feature map dimension, $\overline{L}$ is the output feature map dimension and P is the padding dimension. If the filter dimensions emerge out to be negative, padding is increased to ensure it is positive, All the convolution output feature maps or identity mappings are element-wise averaged and then pooling, as specified in the instruction set, is performed. We denote the series of $\frac{n(n+1)}{2}$ bits as layer-bit-info. An example is in Figure 2. The input layer is denoted as the 0^{th} layer. The detailed algorithm is given in Algorithm 1.

For example, let the encoding for the third convolution layer is 101. The algorithm checks if the third layer dimension before pooling matches with the input layer. If not, padding and convolution operations are performed as specified above. Otherwise, it is added as an identity layer. Similarly, the same is performed for the second layer (the dimension of the second layer here is after the pooling operations). The first layer is skipped as its corresponding bit is negative. This ensures that there are convolution operations between two adjacent layers until and unless they have the same spatial dimensions.

2) Convolution Layer Information Encoding: In this section, we provide the convolution information encoding scheme and the layer dimension calculation algorithm. Each layer has an interval supplied for each of the three quantities in the instruction set.

- Number of filters
- Kernel/Filter dimension
- Padding dimension

Each chromosome consists of three parameters for every convolution layer. This part of a chromosome is a 2-D matrix of size $n \times 3$ where each row corresponds to one convolution layer consisting of n convolution layers. The first element denotes the number of filters, the second is the filter dimension, and the third is the padding dimension. A typical example would be :

$$\begin{pmatrix} 59 & 4 & 1 \\ 39 & 5 & 0 \\ 15 & 3 & 0 \end{pmatrix}$$

This signifies that the CNN has three convolution layers with the following details:

- First layer has 59 filters of dimension 4 each and a padding of 1 is applied to the layers on which convolution is performed.
- Second layer has 39 filters of dimension 5 each and no padding is applied.
- Third layer has 15 filters of dimension 3 each and no padding is applied.

To calculate layer dimensions, we first take the previous layer dimensions (L,B,D). Each layer dimension is (integral part of $\frac{(L-F+1+2P)}{K}$, integral part of $\frac{(B-F+1+2P)}{K}$, W) where

Algorithm 2 Layer Dimension Calculation Algorithm.

Pooling_instruction $\rightarrow$ *max-pool filter size for* i^{th} *layer*
Arr $= []$ *layer sizes before pooling*
ArrP $= []$ *layer sizes after pooling*
$n =$ no. of convolution layers
Arr $=$ Arr $+$ Input Dimension
ArrP $=$ ArrP $+$ Input Dimension
for $i = 0, 1, 2, 3, ..., n-1$ **do**
  $W, F, P =$ chromosome[i] *convolution layer information*
  $K =$ Pooling_instruction[i]
  Index $-1 \rightarrow$ *last element of the array*
  $L =$ ArrP[-1][length]
  $B =$ ArrP[-1][breadth]
  $L^{'} = Int(\frac{(L-F+1+2P)}{K})$
  $B^{'} = Int(\frac{(B-F+1+2P)}{K})$
  Arr $=$ Arr $+(L-F+1+2P, B-F+1+2P, W)$
  ArrP $=$ ArrP $+(L^{'}, B^{'}, W)$
end for

F is the filter dimension, L is the previous layer dimension, P is the padding, W is the number of filters, and K is the stride and dimension of max-pool filter (same here). A detailed algorithm is given in Algorithm 2. For each convolution layer, the bias and weights are randomly initialised using Gaussian estimation. The activation function used here is ReLU $f(x) = max(0, x)$[5].

3) Dense Layer Encoding: This section deals with the encoding procedure for the dense (fully-connected layers) using a row matrix with size $m \times 1$.

Two more layers exist where the first layer is the input dense layer, obtained from flattening of the output feature map of the last convolution layer. The last layer contains the number of nodes in the output layer, like for a classification problem, the number of classes. We use sigmoid activation function[6] $\frac{1}{1+e^x}$ except for the last layer. For classification problems, we use softmax activation[13] $\frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}$ where C is number of classes, z_i is the activation for the i^{th} class. For regression problems, we use linear activation $f(x) = x$. Each element in the matrix consists of numbers within the intervals given in the instruction set.

An example chromosome section is $\begin{pmatrix}65 & 32\end{pmatrix}^T$. This denotes that the hidden layers contain 65 and 32 nodes respectively. A detailed analysis is given in Figure 3.

B. Genetic Operations

In this section, we present the algorithms for the genetic operations, namely crossover, selection, mutation, and gene correction.

1) Crossover: Crossover is the genetic operator to produce new enriched population form the current population by alternating genetic information from the existing parent chromosomes.

Let the population size be N. We take $Int(\frac{N}{2})$ chromosomes randomly as first set of parents, and other half as

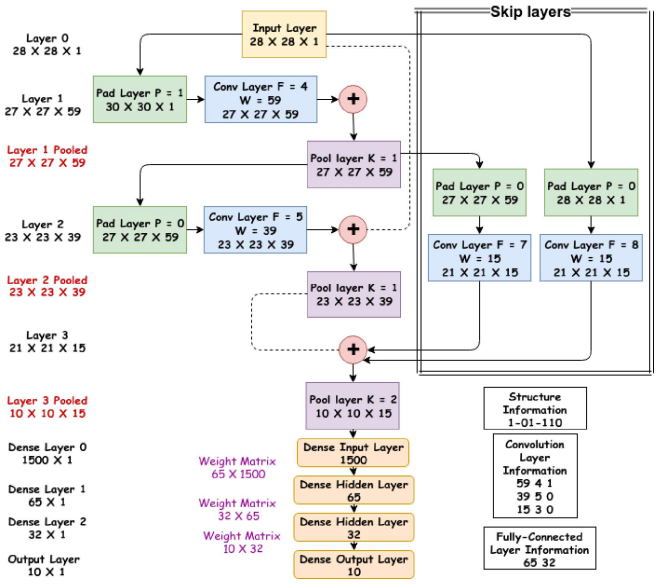

Fig. 3. A complete encoding procedure for the gene given in the image. The dotted lines provide possible connections but not present as their corresponding bit is 0. The convolution layer information matrix and the dense layer information matrix are also given. This is a 3 convolution layer and 2 hidden dense layer convolution neural network.

second set of parents. In case of odd N, we ignore any one of the chromosomes in the population. Then, crossover occurs for every pair from the first and second set of parents. Each chromosome consists of three main sections:

- Structure information of convolution layers.
- Filter number, dimension and padding information of convolution layers.
- Number of nodes in the hidden layers of the fully-connected layers.

For the structure encoding section, XOR operation is performed between the corresponding bits. The reason behind choosing XOR operator is that given two bits, the output will be 1 or 0 is with equal probability, which helps in keeping sufficient variations. Other operators like AND or OR tend to saturate at a particular bit (0 for AND and 1 for OR).

Example : 1-11-110 $\oplus$ 1-10-100 $\longmapsto$ 0-01-010.

For the convolution information, we take alternating information from the parents. The first parent gives its information of number of filters and padding sizes. The second parent gives its information of filter size and dense/fully-connected layer information. An example is provided in Figure 4.

Using the above algorithm, we make half the number of population of off-springs. These off-springs replace the weaker

$$\begin{pmatrix}63 & 4 & 1\\ 59 & 3 & 1\\ 62 & 7 & 0\end{pmatrix}\begin{pmatrix}64\\ 29\end{pmatrix} and \begin{pmatrix}67 & 2 & 0\\ 64 & 4 & 0\\ 59 & 6 & 1\end{pmatrix}\begin{pmatrix}68\\ 32\end{pmatrix} \longmapsto \begin{pmatrix}63 & 2 & 1\\ 59 & 4 & 1\\ 62 & 6 & 0\end{pmatrix}\begin{pmatrix}68\\ 32\end{pmatrix}$$

Fig. 4. Crossover between two parent gene sections denoting the convolution and dense layer information of the Convolution Neural Network.

Algorithm 3 Crossover Algorithm

```
P ← Population Set
OP ← Off-spring Set
for i = 0, 2, 4, .. do
    P1 ← P[i] Parent1
    P2 ← P[i + 1] Parent2
    P ← New off-spring
    Operator ':' → Columns are copied
    P[:, no. of filters] = P1[:, no. of filters]
    P[:, filter sizes] = P2[:, filter sizes]
    P[:, padding values] = P1[:, padding values]
    P[no. of nodes] = P2[no. of nodes]
    P[structure] = P1[structure] XOR P2[structure]
    OP = OP + P
end for
```

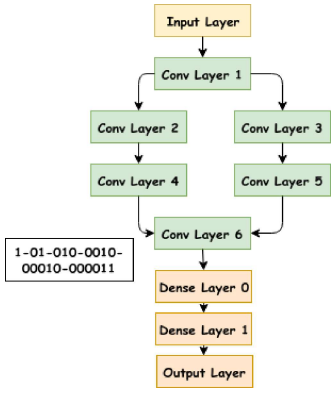

Fig. 5. A bi-directional layer configuration formed using the encoding schema described in this paper. Note here the pooling, average and padding operations are hidden. Only the convolution layers are shown with the given encoding structure. The number of dense layers is taken to be 1.

chromosomes by applying the algorithm explained later. The crossover algorithm is explained in Algorithm 3.

2) Selection and Mutation: The selection algorithm is based on the median of the fitness values (accuracy of the CNN on the validation set). CNN's having fitness values lower than the median are eliminated from the current population and are replaced by the new off-springs created from the crossover algorithm. As a result, the average accuracy of the population increases.

A particular fraction of the population is mutated. Usually, this number is conventionally meagre ($\approx$.2) and is kept constant throughout the process. We change 50% of the values present in this fraction of population randomly. A chromosome specifically contains $n\times3+m\times1$ layer information and $\frac{n(n+1)}{2}$ structure information where n is the number of convolution layers and m is the number of fully-connected layers. For layer information, we choose randomly 50% of positions and replace it with a new random value within the intervals given in the instruction set. For structure information, we choose 50% of the positions in the bit-string and flip the bits at that position.

3) Gene Correction: A glitch occurs when a layer takes no input from its previous layers but delivers input to its subsequent layers resulting in an invalid structure information. An example is 1-00-101. According to this, the second layer provides input to the third layer, but itself accepts no input from its previous layers.

To avoid this, if we find that in a particular chromosome, for a specific convolution layer, all the corresponding bits are zero, we reinitialize it entirely with a new bit string for that layer in a random fashion so that at least 1 bit is one. For this example, the bit string for the second layer will be 11 or 01 or 10. Thus, the new bit-string will be 1-11-101 or 1-01-101 or 1-10-101. This ensures that the second layer takes input from at least one of the previous layers.

A second kind of invalid chromosomes occurs when layer sizes becomes negative due to size-reduction of the layers after subsequent convolution operations. We are going to simply ignore those chromosomes during evaluation and mark them as invalid.

C. Population Initialization and Genetic Operator Sequence

This section deals with the formulation of the instruction set for population initialization and step-wise execution of the algorithms mentioned earlier for every GA iteration. The instruction set mainly comprises of :

- Number of convolution and fully-connected layers
- Maximum range of convolution layer parameters
- Minimum range of convolution layer parameters
- Pooling kernel dimension for each convolution layer
- Maximum range of dense layer parameters
- Minimum range of dense layer parameters
- Number of epochs for the genetic algorithm and back-propagation algorithm
- Size of the population set

The intervals mentioned in the previous sections of the paper, are provided in the instruction set. For all the parameters, there exists a particular interval in which the gene can take values (both limits inclusive). This essentially bounds the search space, and the GA does not unnecessarily go on finding all values.For the pooling layers, we denote the pool filter size as an exponential of 2 as 2^x like 1,2,4,8,....

During population initialization, the instruction set is read, and for each section of the chromosome, except for the structure part, a random value is chosen within the interval. For example, if the interval for the number of filters in the first convolution layer is from 64 to 32, then a random value within that interval is chosen.

The encoding section is formed using random bits for each convolution layer. There is a constraint that in each layer, at least one bit should be positive. To handle this, for each i^{th} layer, there are i bits present, and only $i-1$ bits are randomly chosen. Then positive bit is inserted randomly at any position among those $i-1$ bits. This forms a random string with i bits, and also ensures that at least one bit is positive so that no invalid structures are formed during initialization.

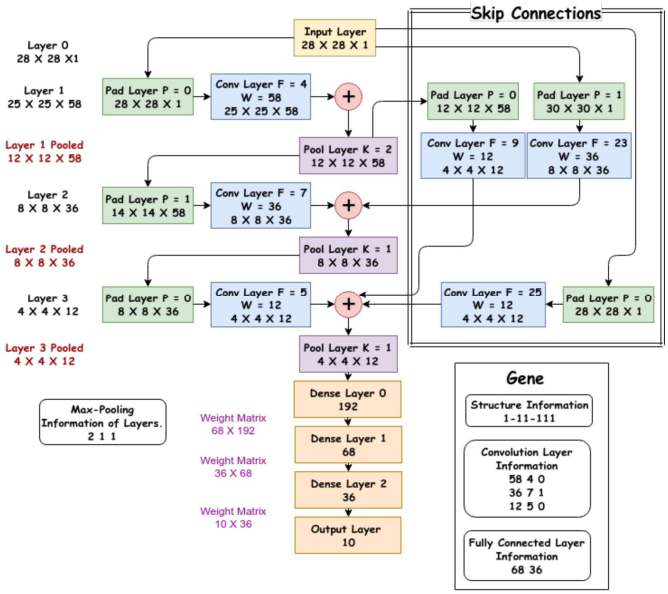

Fig. 6. A 3 convolution and 2 fully-connected layer structure formed when pooling is used in the first layer. We can see here that in the skip connections have very large convolution filters of size 23 and 25 which are not conventionally used in the literature. Although it opens up a new field of exploration where such filters are used.

After the completion of population initialization, the serial execution of the above mentioned algorithms for each GA iteration are as listed below :

1) Calculate the layer dimensions using Algorithm 2 by decoding the convolution and dense layer information for all genes in the population set as referred in Section III-A2 and Section III-A3.
2) Calculate the kernel dimensions for each convolution operations using Algorithm 1 by decoding the structure information of convolution layers for all genes in the population set as referred in Section III-A1.
3) Build and evaluate the model for every gene. Find their accuracy on the validation set.
4) Construct the new set of off-springs using Algorithm 3 as referred in Section III-B1.
5) Replace all those genes in the population having accuracy less than the median accuracy of current population set as referred in Section III-B2.
6) Perform mutation followed by gene correction as explained in Section III-B2 and Section III-B3 to get the new population set ready for the next iteration.

D. Formation of large-sized convolution filters

A significant observation is that we can form extensive convolution filters if we allow pooling operations on the earlier layers. An example is the formation of convolution filters with size 23 and 25, as shown in Figure 6. This occurs as pooling operations abruptly decreases the layer sizes to half. So while taking input from initial layers, such large convolution filters need to be formed to match up the layer dimensions. This opens up a new domain that allows us to explore filters of large dimensions, not conventionally used in literature.

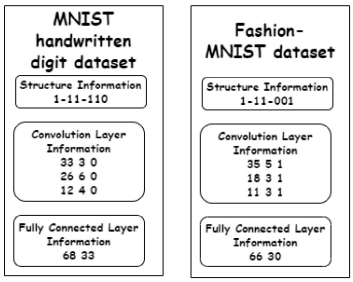

Fig. 7. Final Genes

IV. Experiments and Results

The algorithm is implemented using keras module[14] in python3.5 using an instance of Google Colab. Google Colab, for an instance, has the following specifications:

- GPU : 1xTesla K80 , having 2496 CUDA cores, 12GB GDDR5 VRAM
- CPU : Xeon Processors @2.2Ghz (1core, 2 thread)
- RAM : 12.6 GB available
- Disk : 32 GB available

We perform our training on two datasets, namely MNIST handwritten digit dataset and Fashion-MNIST dataset. Each of these consists of 28×28 black and white images of digits and clothing items. There are ten classes for each dataset representing digits and clothing items, respectively. We have used 50,000 images from the train set for training each neural network architecture and 5,000 images from the test set for validation of each model. The latter half of the test set, consisting of 5,000 images, is used for testing the best model from the algorithm. 10 GA iterations are used for the MNIST handwritten digit dataset and 15 GA iterations for the Fashion-MNIST dataset trained for 5 and 8 hours respectively on average. The population size is 10 and pooling is allowed only in the last convolution layer. The number of epochs for the back-propagation algorithm is 5, and it is performed twice. The best set of weights are noted for validating on the validation set.

The loss function used here is the softmax cross-entropy loss. The optimizer used here is Adam optimizer with learning-rate .001, $\beta_1 = .9$, $\beta_2 = .999$, $\epsilon = 1.0 \times 10^{-8}$. The learning rate is kept constant throughout the 5 epochs. The mutation rate for the GA is kept constant at .2 for all iterations.

After the execution of the algorithm, we find that the best accuracy for the MNIST handwritten digit dataset is 98.54% and 88.58% for the Fashion-MNIST dataset, when the best model is evaluated on the test set. The fittest gene found are present in Figure 7.

A detailed graph for accuracy vs GA iterations is given in Figure 8. We compare our algorithm with other state-of-the-art algorithms, as listed in Table I. The maximum accuracy for these CNN architectures is noted after five epochs, except

TABLE I
COMPARISON WITH OTHER ALGORITHMS ON FASHION-MNIST

Fashion-MNIST dataset	
Model Name	*Maximum Accuracy (%)*
VGG16 Base model [15]	89.60
VGG16 H–CNN model [15]	85.42
VGG19 Base model [15]	89.54
VGG19 H–CNN model [15]	85.63
Spiking Neural Networks [16]	82.21

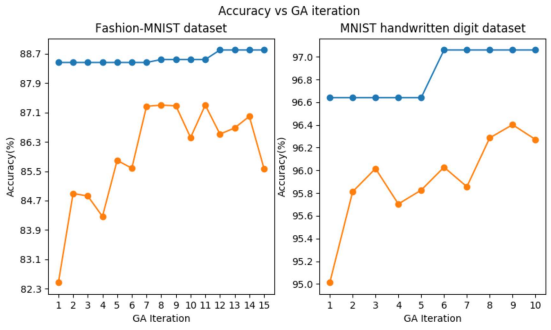

Fig. 8. Variation of accuracy with GA iterations.The blue line denotes the variation of maximum accuracy and the orange line denotes the variation of average accuracy.

for the last one. As Genetic Algorithm is a meta-heuristic algorithm, we have executed it 15 times independently and have recorded an average F1-score of 88.43% for the Fashion-MNIST dataset.

V. CONCLUSIONS AND FUTURE SCOPE

In this paper, we have explored the possibility of using Genetic Algorithm to find the best possible CNN architecture without human intervention. We have proposed an encoding method to determine layer connectivity along with other strategies to evolve the layer dimensions. We have then tested our algorithm on two familiar datasets, namely MNIST dataset for handwritten digit recognition and Fashion-MNIST dataset. Our experiments have shown that they accomplish good results.

Despite the well-accomplished results, we find that our algorithm has good scopes of improvement. Our algorithm does not incorporate the evolution of max-pooling filter sizes and up-convolution or up-sampling operations. It would also be quite interesting to incorporate Genetic Algorithm to allow the simultaneous evolution and training of CNNs instead of using GA to predict network structures only. I acknowledge Professor Dipankar Dasgupta and Professor Shamik Sural for my work.

REFERENCES

[1] Y. B. Y. LeCun, L. Bottou and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE, November 1998*, 1998.

[2] I. S. A. Krizhevsky and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, 2012.

[3] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *Proceedings of the 32nd International Conference on Machine Learning*, 2014.

[4] S. R. J. S. K. He, X. Zhang, "Identity mappings in deep residual networks," *European Conference on Computer Vision*, 2016.

[5] T. C. M. L. B. Xu, N. Wang, "Empirical evaluation of rectified activations in convolutional network," *arXiv 1505.00853v2*, 2015.

[6] A. G. C.E. Nwankpa, W. Ijomah and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *arXiv:1811.03378v1*, 2018.

[7] K. G. R. K. Srivastava and J. Schmidhuber, "Training very deep networks," *Advances in Neural Information Processing Systems, Montral, Canada*, 2015.

[8] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation, vol. 9, no. 8*, 1997.

[9] L. Xie and A. Yuille, "Genetic cnn," *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[10] J. B. P.C. Chu, "A genetic algorithm for the multidimensional knapsack problem," *Journal of Heuristics*, 1998.

[11] K. D. Jong and W. Spears, "A formal analysis of the role of multi-point crossover in genetic algorithms," *Annals of Mathematics and Artificial Intelligence*, 1992.

[12] S. A. K. Deb, A. Pratap and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, VOL. 6, NO. 2*, 2002.

[13] R. A. Dunne and N. A. Campbell, "On the pairing of the softmax activation and cross-entropy penalty functions and the derivation of the softmax activation function," *Conference on Neural Networks, Melbourne*, 1997.

[14] F. Chollet *et al.*, "Keras." https://keras.io, 2015.

[15] S. K.-s. Seo, Yiana, "Hierarchical convolutional neural networks for fashion image classification," *Expert Systems with Applications*, 2019.

[16] M. W. K. N. A. M. P. Opiełka, J.T. Starczewski, "Application of spiking neural networks to fashion classification," *International Conference on Artificial Intelligence and Soft Computing*, 2019.