Optimization of CNN Architecture using Genetic Algorithm for Image Classification

ABSTRACT

This project uses Convolution Neural Networks (CNN) for the task of Image Classification. To achieve best accuracy the CNN architecture must be modelled with various numbers of architectures with varying number of filters, kernel size, number of layers etc. To avoid the tedious trial and error method, this project uses Genetic Algorithm to obtain optimum CNN architecture. The project is implemented using TensorFlow and Keras over Face Database from AT&T Laboratories, Cambridge University [2].

I. INTRODUCTION

Feed forward Neural Networks can be used to solve any kind of regression or classification problems but lacks in the field of computer vision as the number of parameters to optimize is very high in fully connected layers also ANN's cannot identify the objects in a given image due to these reasons ANN's is not recommended for identifying object in an image. The usage of Convolution Neural Networks in the field of Image classification has achieved remarkable success in recent years. Automating the design of CNN's is required to help some users having limited domain knowledge to finetune the architecture for achieving desired performance and accuracy. Usage of different evolutionary methods such as Genetic Algorithms helps in simplifying, automating the architecture of CNN's and also to improve their performance [1]. Section II explains the general principle of Convolution Neural Networks, Section III presents working of Genetic Algorithm in a great detail. Section IV shows the working methodology of proposed Genetic CNN and we wrap up with the results and conclusions in Section V. The TensorFlow code used for this project will be attached in Appendix I.

II. CONVOLUTION NEURAL NETWORKS

The principle behind CNN is same as Artificial Neural Network except CNN's benefits from weight sharing between convolution layers as this reduces parameters required to optimize during back propagation. Figure 1. shows the famous Alex Net CNN configuration with several combinations of convolution and max pooling layer. Finally, CNN output is given to a ANN network with the last layer being a SoftMax activation to obtain predicted probabilities of output classes. Each convolution layer in CNN has many filters which captures certain features of image such as edges so that at the end of training each layer specializes in capturing specific feature characteristics of given images. The following subsections explains convolution layer, activation functions, full connected layers, max pooling layers of CNN.

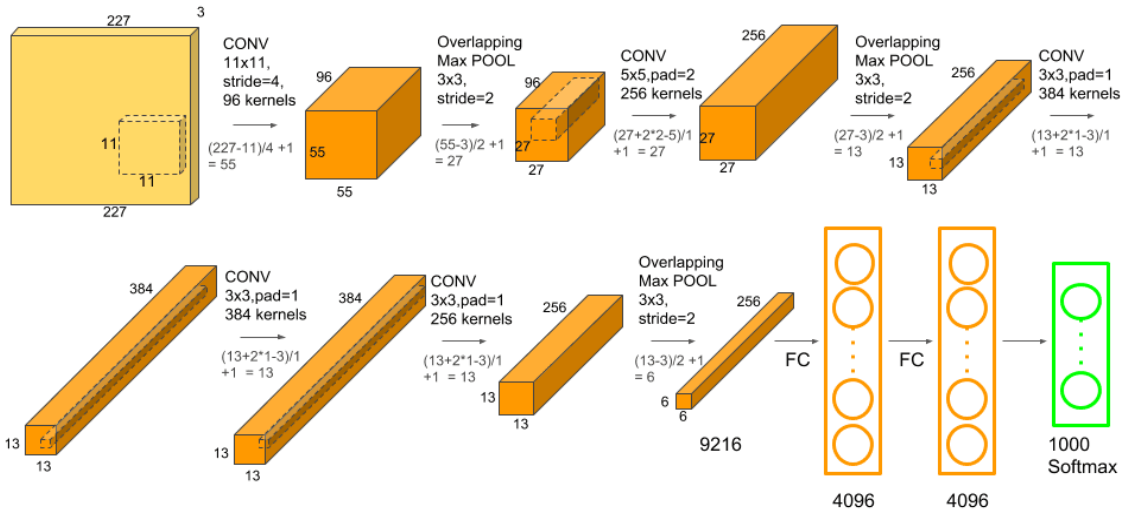

Figure 1. Alex Net, a configuration of CNN [3]

COVOLUTION LAYER

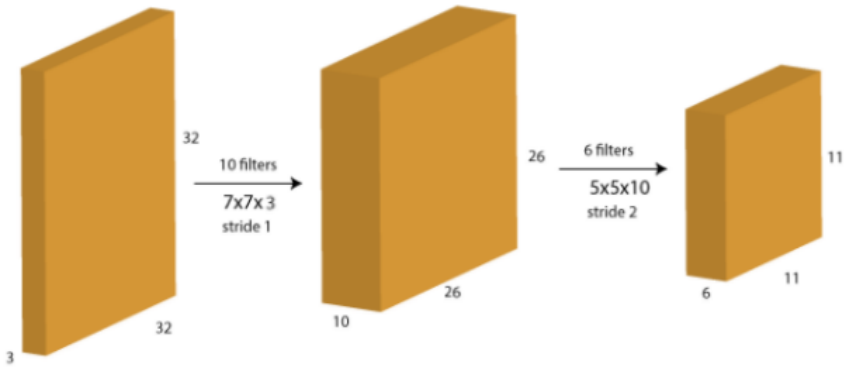

Figure 2. Convolution Operation of CNN [3]

In the above convolution operation, each pixel of the input image of dimensions (32 x 32 x 3) is multiplied with filters of kernel size (7 x 7 x 3) to get a single output layer as 10 different filters were used above therefore producing an output of dimension (26 x 26 x 10) note that width and height changes as no padding is used here.

ACTIVATION FUNCTION

To introduce non linearity in the model, the output values of convolution layers are passed through activation functions. In this project ReLU activation function is used other famous functions include tanh for recurrent neural networks and sigmoid for ANN. The figure below shows the graph of ReLU .

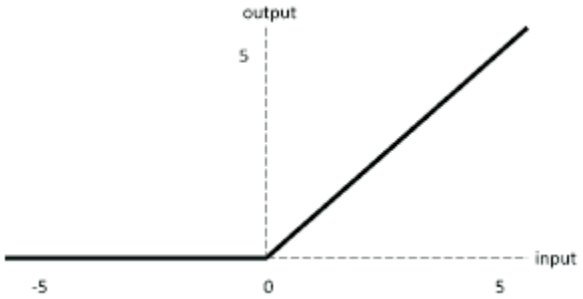

Figure 3. ReLU Activation function, f(x) = max(0, x)

MAX POOLING LAYER

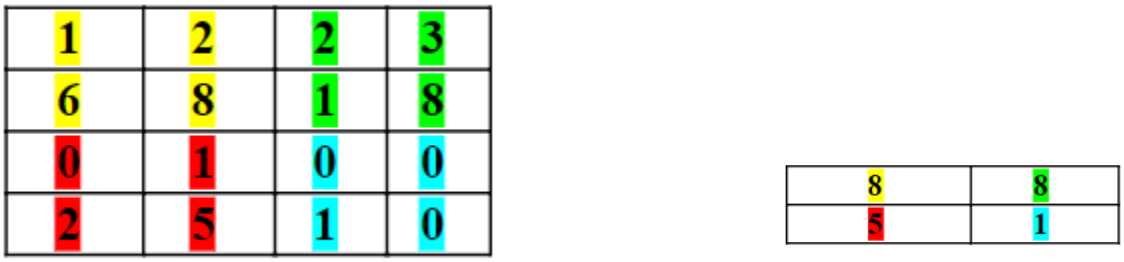

Figure 4. Max Pooling Layer Working

To reduce or down sample the width and height of input layers, max pooling is used with specific pool size and strides. It extracts the maximum pixel value and substitutes over the entire kernel.

FULLY CONNECTED LAYER

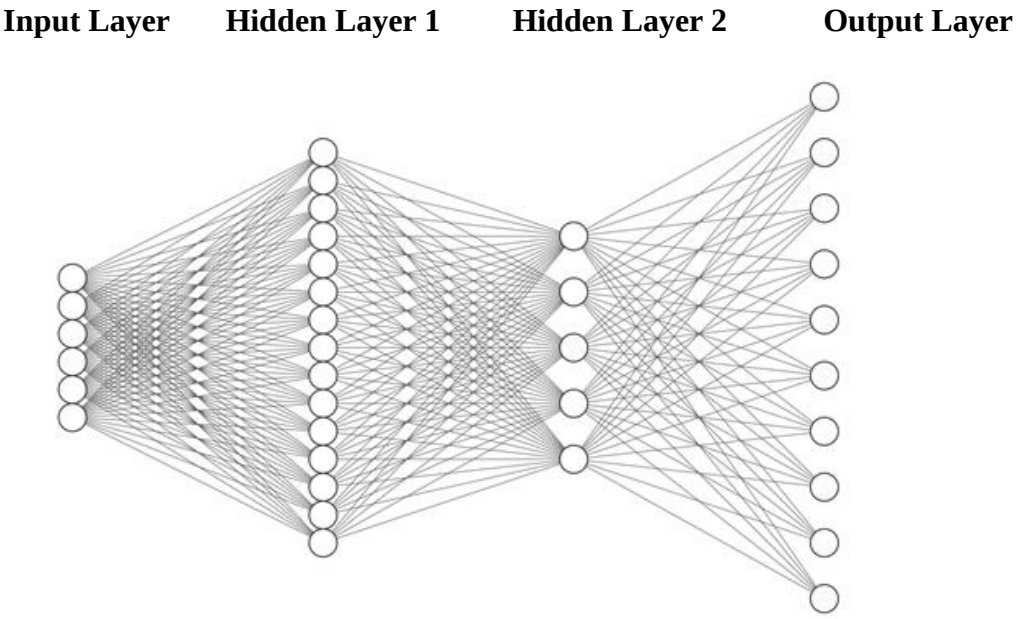

Figure 5. Feed Forward Artificial Neural Network

In the Feed Forward Network all the pixel outputs from last layer of CNN is flattened and are sent as input to obtain output classification.

SOFTMAX LAYER

To get probabilities of output classes, values of output layer are passed to a softmax function defined as follows:

$$SoftMax(n) = exp(n)/sum(exp(n))$$

III. GENETIC ALGORITHM

GA is an heuristic search algorithm which is inspired from biological evolution analogy of crossing over fittest chromosomes to generate off springs. Genetic Algorithms work by applying "random" changes to current solutions in order to create new ones. To select the best parameters, a fitness function is used and solutions representing the higher fitness value is chosen. The figure below explains steps involved in implementing Genetic Algorithms [1].

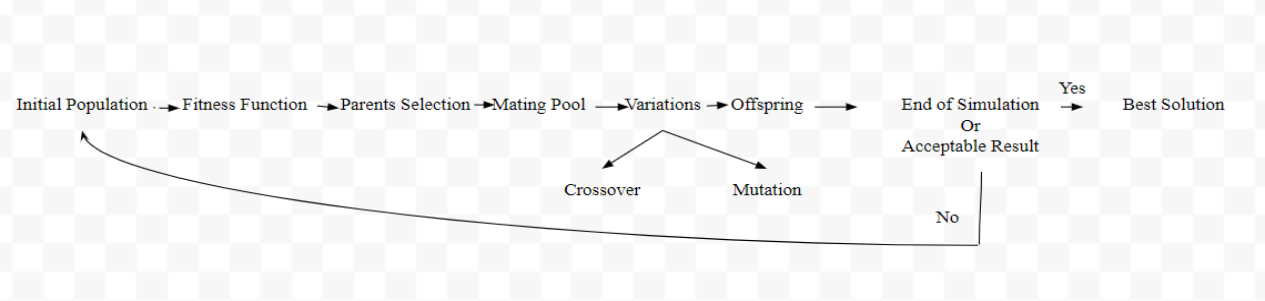

Figure 6. Steps in Genetic Algorithm

INITIAL POPULATION

The initial population is a set of randomly generated chromosomes. Given the number of filters for each layer and also size of kernels for convolution, the NumPy random generator gives us the initial population as shown in the below table.

Individual	Number of Filters			Size of Filters			Fitness (%)
	Layer 1	Layer 2	Layer 3	Layer 1	Layer 2	Layer 3	
1	45	48	65	16	11	19	78
2	68	68	10	4	18	15	50.75
3	84	22	37	8	1	2	94
4	88	71	89	10	1	11	90
5	89	13	59	4	12	19	90.5
6	66	40	88	3	1	1	95
7	47	89	82	5	6	7	93.25
8	38	26	78	9	18	16	58
9	73	10	21	5	10	11	89
10	81	70	80	2	2	8	94.5

Table 1. Randomly Generated Initial Population

FITNESS FUNCTION

For each individual of randomly generated initial population, train the CNN using training data and calculate accuracy of CNN architecture using the test dataset. Here accuracy multiplied by 100 is used as the fitness function to be optimized or maximized. The parents in the next step are chosen based on this fitness values.

PARENTS SELECTION

At each step the top individual of the population having highest fitness values is selected, this process is continued till the required number of parents is obtained for crossover and mutation in the next steps. The parents for this initial population is shown in below table.

Individual	Number of Filters			Size of Filters			Fitness (%)
	Layer 1	Layer 2	Layer 3	Layer 1	Layer 2	Layer 3	
1	66	40	88	3	1	1	95
2	81	70	80	2	2	8	94.5
3	84	22	37	8	1	2	94
4	47	89	82	5	6	7	93.25
5	89	13	59	4	12	19	90.5

Table 2. Selected Parents from Initial Population

CROSSOVER FUNCTION

Two top chromosomes of the parent population is chosen to create a new individual by applying crossover. Here a fixed cross over is used, randomly selected cross over can also be used. It works by selected half of values from parent I and the rest from parent II. The process is as shown in below figure and child population is presented in table 3

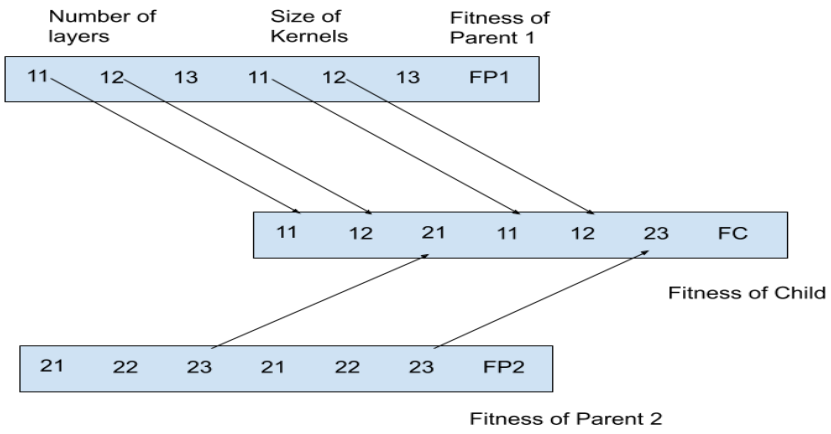

Figure 7. Crossover Operation

Individual	Number of Filters			Size of Filters			Fitness (%)
	Layer 1	Layer 2	Layer 3	Layer 1	Layer 2	Layer 3	
6	66	40	80	3	1	8	-
7	81	70	37	2	2	2	-
8	84	22	82	8	1	7	-
9	47	89	59	5	6	19	-
10	89	13	88	4	12	1	-

Table 3. Child Population

MUTATION FUNCTION

The mutation function adds some newness to population by introducing new values in the individuals. It works by selecting random layer for each individual and adding or subtracting a randomly generated number from the current value. This helps genetic algorithm to try new parameters rather than continuing with the same initial population. The table below shows mutated child population.

Individual	Number of Filters			Size of Filters			Fitness (%)
	Layer 1	Layer 2	Layer 3	Layer 1	Layer 2	Layer 3	
6	66	**42**	80	3	1	**11**	-
7	**84**	70	37	2	**5**	2	-
8	84	**27**	82	**11**	1	7	-
9	**51**	89	59	**8**	6	19	-
10	**94**	13	88	**5**	12	1	-

Table 4. Mutated Child Population

GENERATION

The obtained parents and child populations are combined to create a new population on which fitness of all the individuals are calculated and then the whole process is repeated until the maximum fitness value of the population stabilizes or converges.

IV. METHODOLOGY

Here we discuss the complete steps involved in optimization of CNN architecture using Genetic Algorithm.

DATA SET

The dataset used for training and testing is obtained from Face Data of AT&T Laboratories at Cambridge University [2]. Since, we had no access to high computing GPU's a small dataset is chosen but the procedure remains same for any complex dataset. The face dataset contains 10 images of 40 different people with different facial expressions (glasses, smiling, eyes closed etc.). The figure below shows random faces with class labels from dataset.

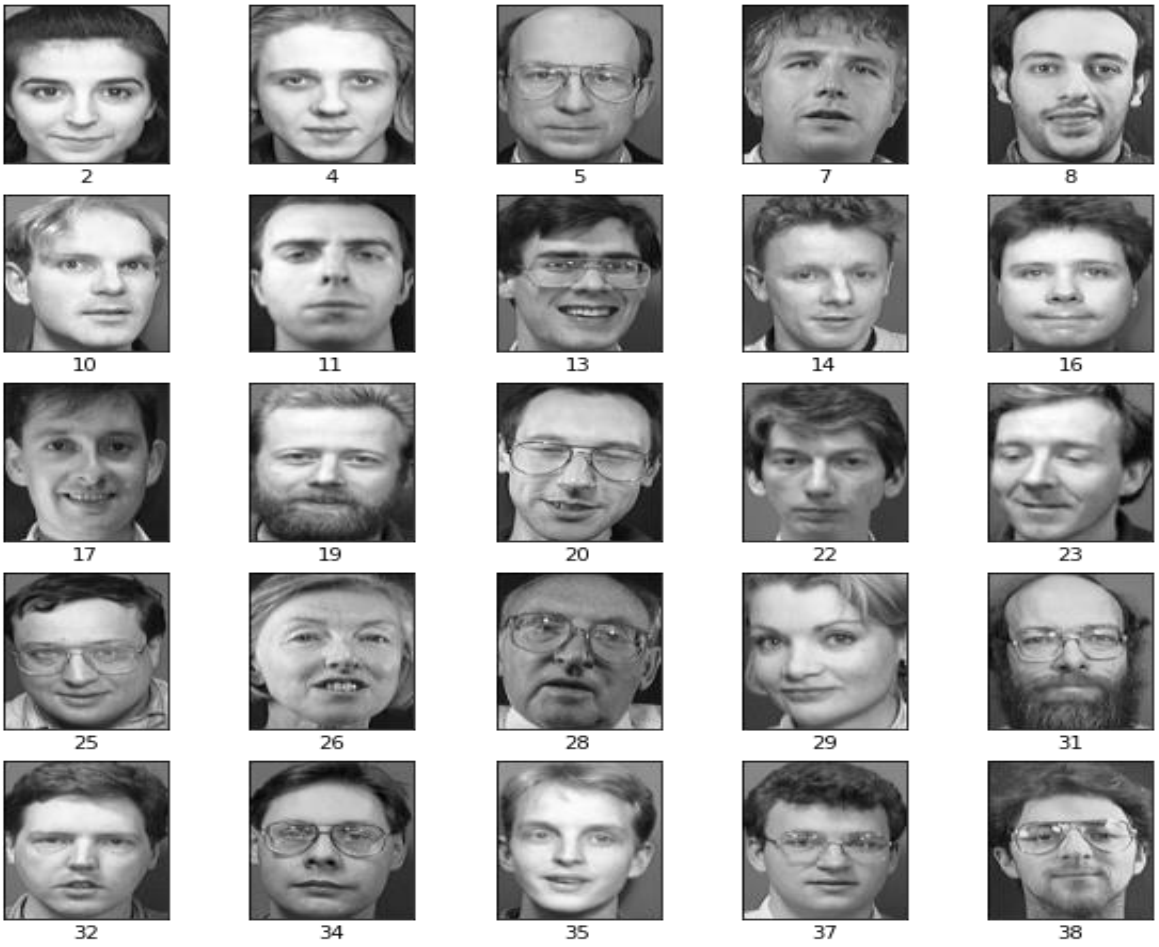

Figure 8. Face Dataset (40 people, 10 images each)

CNN CONFIGURATION

The CNN to be optimized consists of 3 layers with max pooling layers in between and output of conv layer is connected to fully connected neural network which gives predicted probabilities over 40 different classes. The figure below describes a sample CNN architecture(**?** denotes the batch size given at compilation).

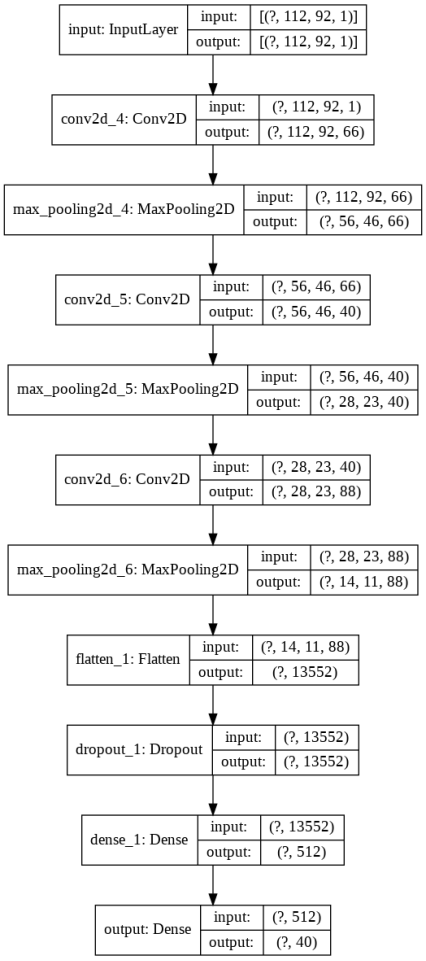

Figure 9. Sample CNN Architecture to be optimized

VGG16 CONFIGURATION

This genetic CNN architecture is compared with pretrained complex VGG16 network on ImageNet challenge, the convolution layer weights were kept constant only the fully connected layers are trained on the face dataset. The figure below shows layers and number of parameters involved in VGG16 architecture.

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 112, 92, 3)]      0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 112, 92, 64)       1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 112, 92, 64)       36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 56, 46, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 56, 46, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 56, 46, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 28, 23, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 28, 23, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 28, 23, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 28, 23, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 14, 11, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 14, 11, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 14, 11, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 14, 11, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 7, 5, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 7, 5, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 7, 5, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 7, 5, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 3, 2, 512)         0
_________________________________________________________________
flatten_7 (Flatten)          (None, 3072)              0
_________________________________________________________________
dense_17 (Dense)             (None, 256)               786688
_________________________________________________________________
dense_18 (Dense)             (None, 40)                10280
=================================================================
Total params: 15,511,656
Trainable params: 796,968
Non-trainable params: 14,714,688
```

Figure 10. VGG16 Architecture

TRAIN AND TEST CNN

The dataset is split into training and testing datasets. First both CNN's are tested on training dataset with maximum epochs of 20 and batch size of 32. At the end of generations the individual with maximum fitness population is selected and tested on test dataset to obtain final accuracy.

V. RESULTS AND DISCUSSION

The Genetic Algorithm is run with the following parameters.

Individuals Per Population	10
Number of Generations	10
Maximum No. Of Filters	100
Maximum size of Kernels	20
No. Of Parents and Children	5 each

Table 5. Hyper Parameters of Genetic Algorithm

Due to various randomness involved in the initialization of TensorFlow tensors during training, maximum fitness accuracy on test data kept oscillating around **97.5 %** as shown in below figure.

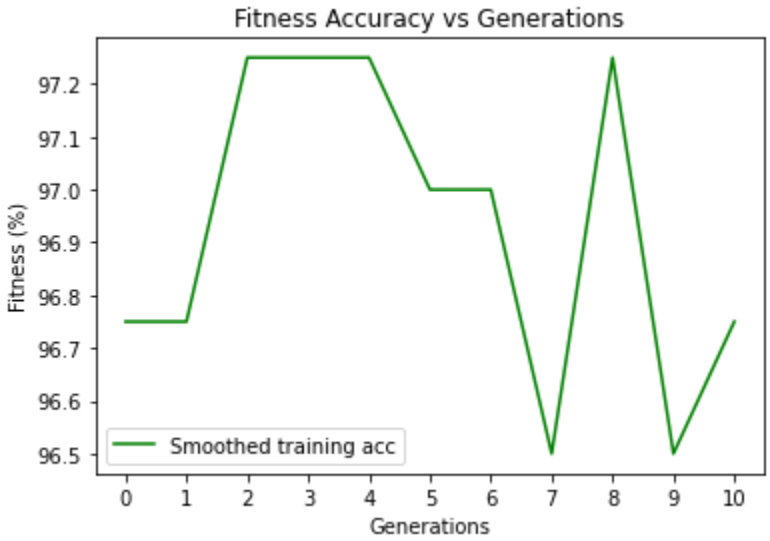

Figure 11. Fitness vs No. Of Generations

After 10 generations the best architecture obtained is:

Individual	**Number of Filters**			**Size of Filters**			**Fitness (%)**
	Layer 1	**Layer 2**	**Layer 3**	**Layer 1**	**Layer 2**	**Layer 3**	
Best	66	40	88	3	1	1	97.5

It is to be noted that even having size of kernels a maximum of 20 at the end we obtained the best kernels of size 3,1,1 which is the common size in all the famous CNN architectures such as AlexNet, VGG16, ResNet etc,. The plot below shows Percentage classification accuracy of best genetic CNN architecture for each face label.

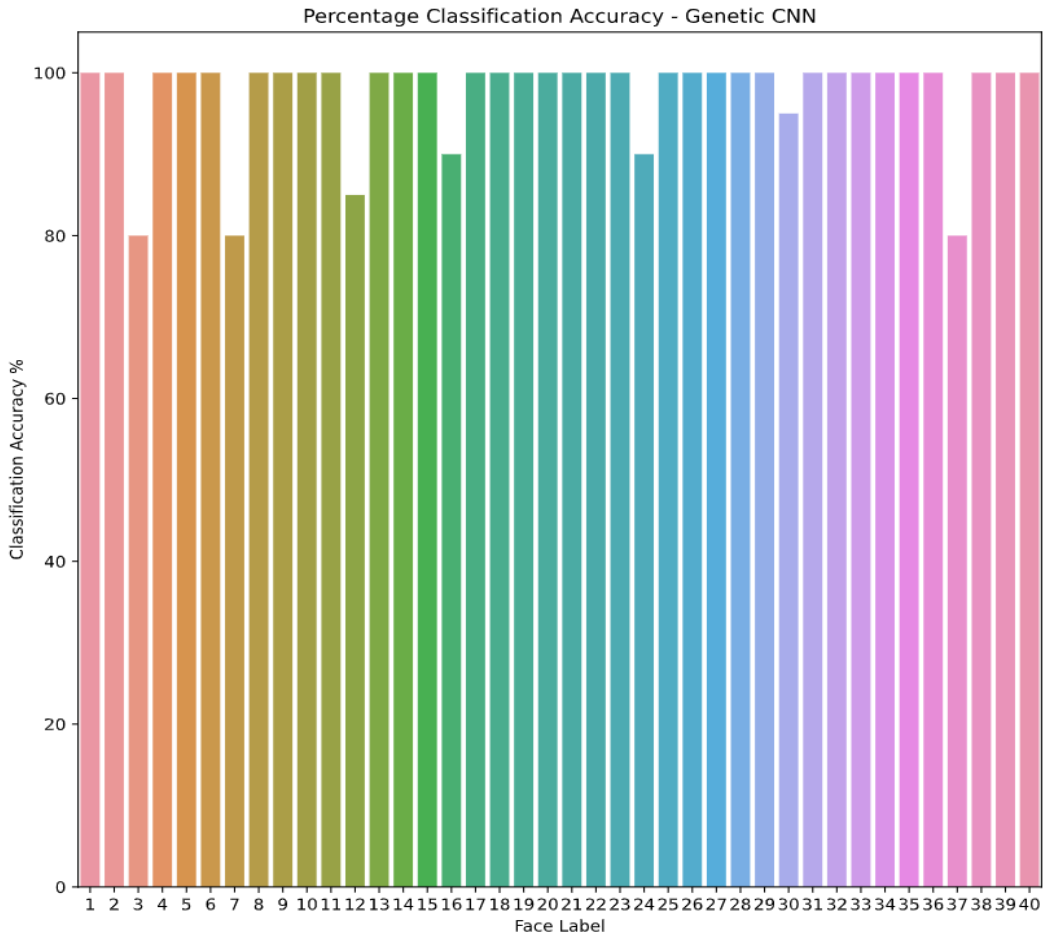

Figure 12. Percentage Classification Accuracy for each Face label

The VGG16 model achieves an accuracy of **100 %** on test data. The below heat map of original class vs predicted class describes 100% accuracy of the model.

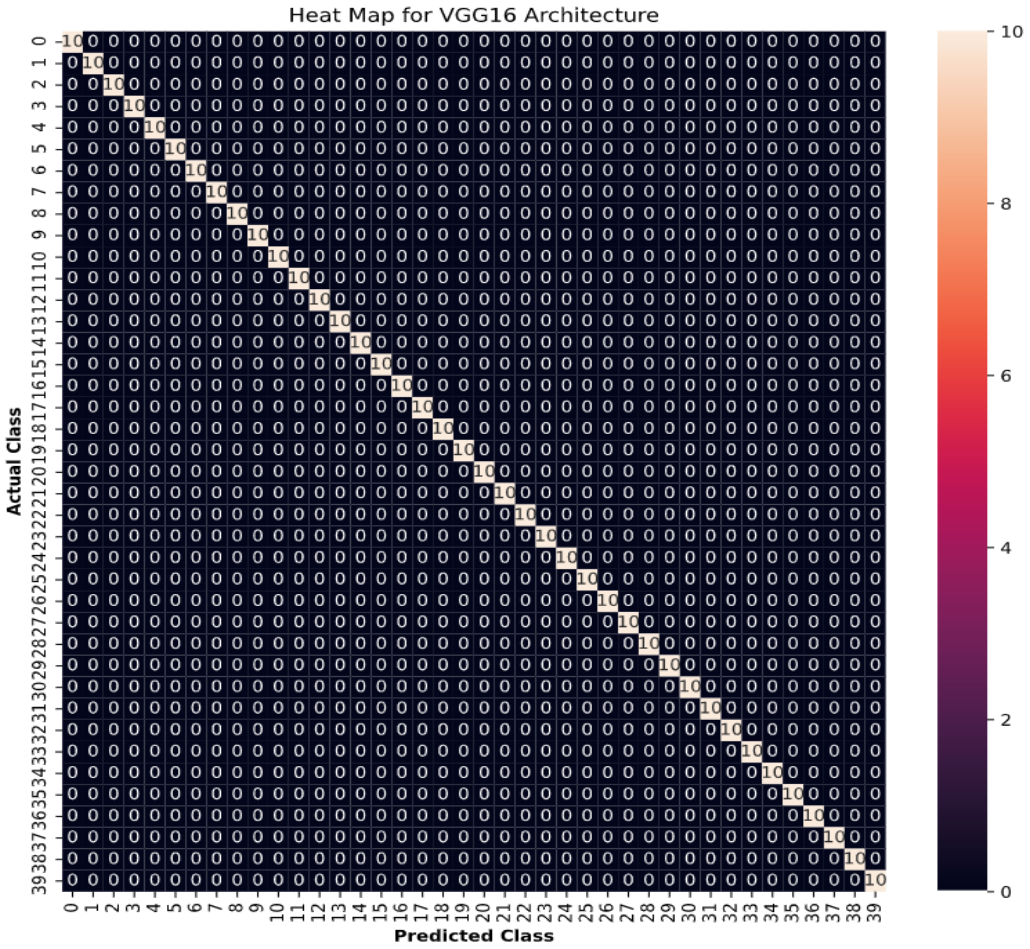

Figure 13. Heat Map for VGG16 Architecture

VI. CONCLUSION

In conclusion, this project demonstrates the application of Genetic Algorithm for finding best CNN architecture in the field of computer vision. This greatly helps people who are not well aware of complex available architectures without losing much accuracy and saves time when compared to following regular trial and error method. The applications of GA can be used in many different fields where search of parameters is required to maximize fitness function.

VII. REFERENCES

1. Yanan Sun, Bing Xue, Mengjie Zhang, “Automatically Designing CNN Architecture Using Genetic Algorithm for Image Classification”. IEEE Transactions on Cybernetics, 2020.

2. Face Database, AT&T Laboratories, Cambridge University: http://www.cl.cam.ac.uk/research/dtg/attarchive/face database.html
3. Munneb ul Hassan, https://neurohive.io/en/popular-networks/alexnet-imagenet-classification-with-deep-convolutional-neural-networks/

APPENDIX I

1. CNN ARCHITECTURE

```
class CNN(Sequential):

      def __init__(self,nfilters,sfilters):

            super().__init__()

            tensorflow.random.set_seed(0)
            self.add(Conv2D(nfilters[0],kernel_size=(sfilters[0],sfilte
            rs[0]),padding='same',activation='relu',input_shape=(112,92
            ,1)))  self.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
            self.add(Conv2D(nfilters[1],kernel_size=(sfilters[1],sfilte
            rs[1]),padding='same',activation='relu'))
            self.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
            self.add(Conv2D(nfilters[2],kernel_size=(sfilters[2],sfilte
            rs[2]),padding='same',activation='relu'))
            self.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
            self.add(Flatten())                    self.add(Dropout(0.3))
            self.add(Dense(512,activation='relu'))
            self.add(Dense(40,activation='softmax'))
            self.compile(optimizer='adam',loss='categorical_crossentrop
            y',metrics=['accuracy'])
```

2. GENETIC ALGORITHM

```
class Genetic:

 def __init__(self,pop_size,nlayers,max_nfilters,max_sfilters):

       self.pop_size = pop_size

       self.nlayers = nlayers

       self.max_nfilters = max_nfilters

       self.max_sfilters = max_sfilters
```

```
        self.max_acc = 0
        self.best_arch = np.zeros((1,6))
        self.gen_acc = []
    def generate_population(self):
        np.random.seed(0)
        pop_nlayers =
        np.random.randint(1,self.max_nfilters,(self.pop_size,self.nlayers))
        pop_sfilters =
        np.random.randint(1,self.max_sfilters,(self.pop_size,self.nlayers))
        pop_total = np.concatenate((pop_nlayers,pop_sfilters),axis=1) return
        pop_total
def select_parents(self,pop,nparents,fitness):
        parents = np.zeros((nparents,pop.shape[1]))
        for i in range(nparents):
        best = np.argmax(fitness)
        parents[i] = pop[best]
        fitness[best] = -99999
        return parents
    def crossover(self,parents):
        nchild = self.pop_size - parents.shape[0]
        nparents = parents.shape[0]
        child = np.zeros((nchild,parents.shape[1]))
        for i in range(nchild):
                first = i % nparents
                second = (i+1) % nparents
                child[i,:2] = parents[first][:2]
                child[i,2] = parents[second][2]
```

```
            child[i,3:5] = parents[first][3:5]
            child[i,5] = parents[second][5]
        return child
    def mutation(self,child):
        for i in range(child.shape[0]):
            val = np.random.randint(1,6)
            ind = np.random.randint(1,4) - 1
        if child[i][ind] + val > 100:
            child[i][ind] -= val
        else:
            child[i][ind] += val
            val = np.random.randint(1,4)
            ind = np.random.randint(4,7) - 1
        if child[i][ind] + val > 20:
            child[i][ind] -= val
        else:
            child[i][ind] += val
        return child
    def fitness(self,pop,X,Y,epochs):
        pop_acc = []
        for i in range(pop.shape[0]):
            nfilters = pop[i][0:3]
            sfilters = pop[i][3:]
            model = CNN(nfilters,sfilters)
            H = model.fit(X,Y,batch_size=32,epochs=epochs)
            acc = H.history['accuracy']
```

```
            pop_acc.append(max(acc)*100)
        if max(pop_acc) > self.max_acc:
            self.max_acc = max(pop_acc)
            self.best_arch = pop[np.argmax(pop_acc)]
        self.gen_acc.append(max(pop_acc))
        return pop_acc
    def smooth_curve(self,factor,gen):
        smoothed_points = []
        for point in self.gen_acc:
            if smoothed_points:
                prev = smoothed_points[-1]
                smoothed_points.append(prev*factor + point * (1-factor))
            else:
                smoothed_points.append(point)
        plt.plot(range(gen+1),smoothed_points,'g',label='Smoothed      training
      acc')
        plt.xticks(np.arange(gen+1))
        plt.legend()
        plt.title('Fitness Accuracy vs Generations')
        plt.xlabel('Generations')
        plt.ylabel('Fitness (%)')
        plt.show()
```