Genetic Algorithm Optimization of CNN Architecture for Image Recognition

Author: Divyendu Narayan
Graduate Student, Electrical Engineering
email: dnarayan@umich.edu
University of Michigan-Dearborn

ABSTRACT

This project uses Convolutional Neural Networks(CNN) for face recognition. To get optimum performance, CNN needs to be evaluated with different configurations for various numbers of convolutional layers, filter size in each convolution layer, number of convolution filters in each layer, pool size (for down sampling) the images. The trial and error method for selecting configuration does not guarantee an optimal performance, requires user to continuously observe the performance trend (improvement in prediction accuracy for changes in number of layers, filter size and number of filters). This project makes use of Genetic Algorithm for altering the configuration of CNN to obtain optimal performance (accuracy of image recognition). Algorithm has been implemented using Matlab and makes use of Neural Network Toolbox (for CNN) [1]*, Image Processing Toolbox* [2] *and Global Optimization Toolbox (for Genetic Algorithm)* [3]*. Experiments have been carried over Face Database from AT&T Laboratories, Cambridge University* [4]*.*

I. INTRODUCTION

Artificial Neural Networks can be trained for solving regression and classification problems. A multi-layer perceptron (MLP) with non-linear activation function (sigmoid, hyperbolic tangent function) can approximate any non-linear function. MLP has input layer, hidden layer and output layer. Except input layer the other two layers have neurons with nonlinear activation function. In images features are often constrained in a smaller 2D region. Hence it is not useful to have a fully connected MLP (input layer connected to each pixel) for detecting objects in an image. Also, an object let's say a cat can be anywhere in the image, so a fully connected MLP need two be modified to identify object anywhere in the image. For these reasons using a MLP for detecting objects in an image is not recommended.

For recognizing objects in an image Convolutional Neural Networks are used. It is a special case of MLP in which instead of scalar weights for neurons, a 2D convolution kernel is used as weight. The convolution kernel is applied over the complete image. When the object is located at different positions in the training image a spatially invariant CNN is obtained. Details of training and application of CNN has been provided in Section II.

Several trials may be required to obtain the architecture for CNN in terms of number of convolutional layers, size of convolution kernel and number of convolution filters in the hidden layer. This project has made use of Genetic Algorithm for carrying out iterations to obtain optimal architecture for CNN. Genetic Algorithm (GA) is a heuristic search algorithm to find set of optimal parameters for a problem. It does so by defining parameters of a problem in terms of bit string (for logic problems) or a set of real numbers called chromosomes. Starting with a set of randomly generated chromosomes called initial population, GA continuously carries out crossover (combination of parent chromosomes to generate offspring chromosomes), mutation (manipulating bit or real number in offspring chromosome) and selection (using a fitness criterion to select a fraction of parent chromosomes to next generation). GA is being used for its ability to converge to global minima. This helps in eliminating the need of user to continuously monitor the CNN performance (improving prediction accuracy) while iterating through various architecture configurations. Application of GA for optimization of CNN has been described in Section III.

Section IV describes the proposed system configuration for optimization of CNN architecture using GA. Section V presents the experiments used to verify the working of proposed system. Section VI presents the conclusion obtained for this project. The Matlab code used for the system has been presented in the Appendix I.

II. CONVOLUTIONAL NEURAL NETWORK

CNN is like a normal Artificial Neural Network. However, the weights are in the form of 2D convolution kernels. A CNN can have several stages of combination of convolution layer and max pool layer (as shown in fig 1).

Finally, CNN has fully connected layer with number of outputs equal to number of classes for training data and a SoftMax layer that gives out the most probable class with related probability.

First convolution layer is responsible for finding the edges for a given image, second layer is responsible for creating shapes out of the edges extracted like semi-circle, squares. As we move along the convolution layers higher levels of abstraction are obtained. CNN creates its own design for feature extraction given the set of training images.

Following sub-sections describe activation function, convolutional layer, max pool layer, full connected layer, SoftMax layers of CNN.

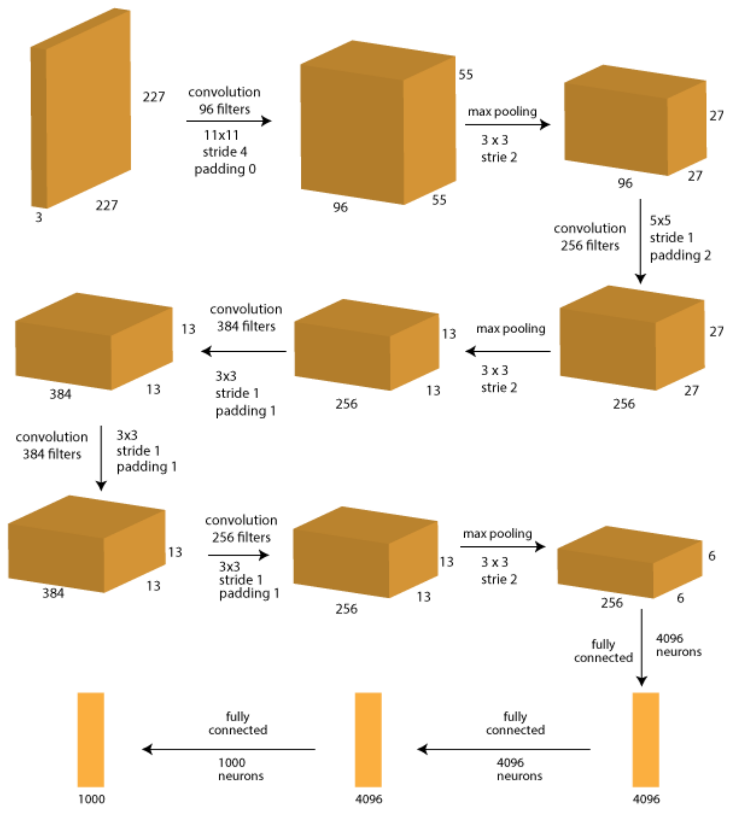


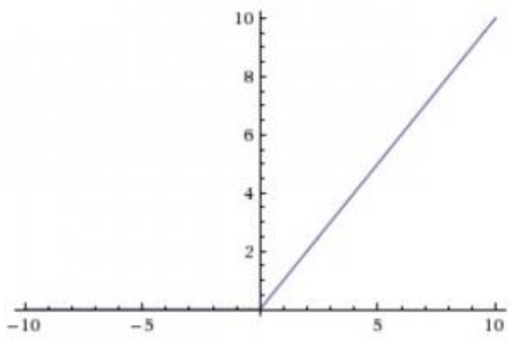
Fig 1: AlexNet, example of CNN [5] (It has several stages composed of convolution and max pool layer and finally has two fully connected layers)

Image Preprocessing

All images need to be converted to same size and normalized (mean value of image intensity subtracted from each pixel).

Activation Function

CNN makes use of ReLU activation function function as its gradient does not have saturation in positive region (does not reduce learning rate as more layers are added), involves simple thresholding, has faster convergence rate compared to tanh or sigmoid function.



[5]

Fig 2: ReLU (rectified linear unit) Activation function, f(x) = max (0, x) [5]

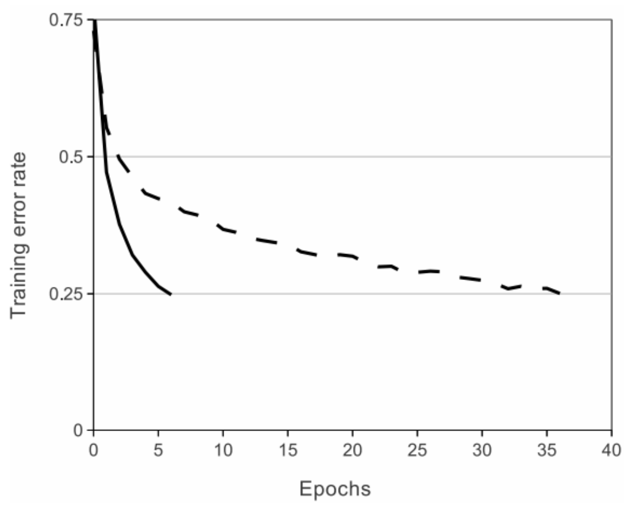


Fig 3: CNN Training Error Rate with ReLU (solid line) vs tanh (dashed line) [6]

Since convergence rate for ReLU is higher compared to tanh for large data sets it is preferred activation function.

Weight Initialization

For less than 5 hidden layers, weights can be initialized to have random values with zero mean and small standard deviation [5].

Convolution Layer

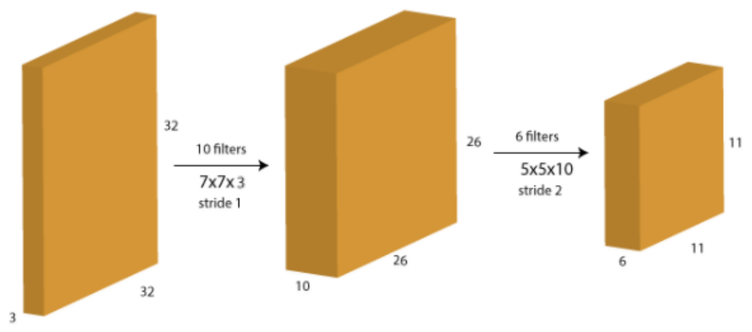


Fig 4: Representation of convolution operation [5]

Consider image in form of cuboid (32x32x3) over which a filter in the form of cuboid (7x7x3) is run. The resulting image is the weighted sum of the image with filter weights as the filter is moved over the image. The resulting image is 26x26x10 dimensions (boundary value of 2 discarded from each side of original image). When a single filter is used then the depth of the resulting image is 1. In given case, the number of filters is 10 hence the depth is 10.

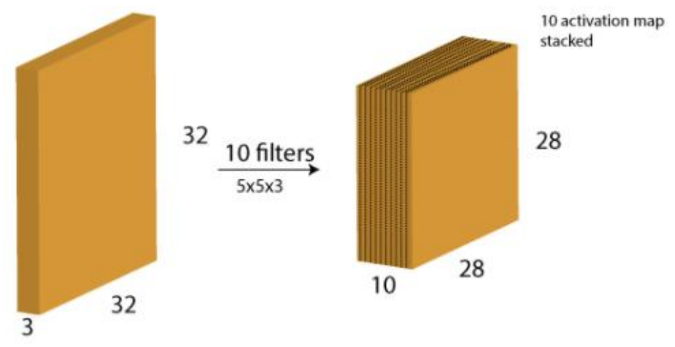


Fig 5: Representation of convolution operation when multiple filters are used [5]

In the above operation since 10 filters are being used the output has a depth of 10.

In order that the resulting image has the same size as original image padding is used. For example, a padding of 2 for above image would result into 32x32x10 output.

Max Pooling Layer

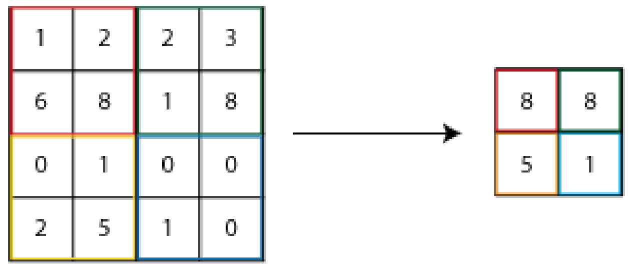


Fig 6: working of Max Pooling Layer [5]

It is used for down sampling image at each stage. When a max pooling kernel is applied over a region, it extracts the max pixel value and places it in the resulting image as shown above.

Fully Connected Layer

Fully Connected Layer is like a normal neural network where each pixel of the image is input to Neural Network.

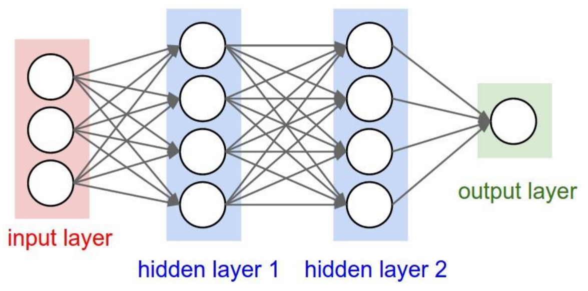


Fig 7: Fully connected Neural Network [7]

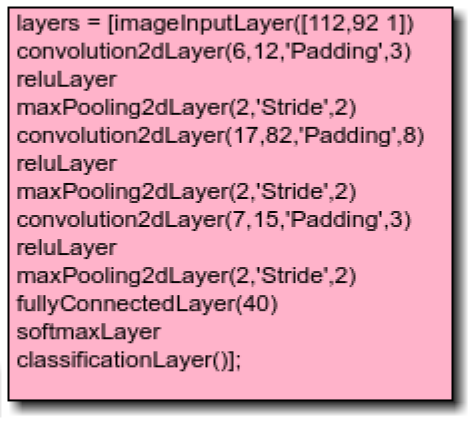
SoftMax Layer

Given a vector 'n' SoftMax transfer function is defined as follows:

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

It gives the probability of a class.

CNN Layer Configuration in Matlab



```
layers = [imageInputLayer([112,92 1])
convolution2dLayer(6,12,'Padding',3)
reluLayer
maxPooling2dLayer(2,'Stride',2)
convolution2dLayer(17,82,'Padding',8)
reluLayer
maxPooling2dLayer(2,'Stride',2)
convolution2dLayer(7,15,'Padding',3)
reluLayer
maxPooling2dLayer(2,'Stride',2)
fullyConnectedLayer(40)
softmaxLayer
classificationLayer()];
```

Fig 8: Layers configuration for CNN in Matlab

Configuration of the CNN layers can be interpreted as follows:

(i) input layer: size same size as image size (112,92,1).
(ii) first convolution layer: consists of 12 convolution filters of size 6, padding of 3 pixels around input image. max pooling layer (size 2x2 and stride of 2), activation function ReLU
(iii) second convolution layer: consists of 82 convolution filters of size 17, padding of 8 used around input image, max pooling layer (size 2x2 and stride 2), activation function ReLU
(iv) third convolution layer: consists of 15 convolution filters of size 7, padding of 3 pixels used around input image, max pooling layer (size 2x2 and stride 2), activation function ReLU.
(v) fully connected layer: has 40 outputs corresponding to 40 object classes.
(vi) SoftMax layer: provides probability of the object
(v) Classification layer: provides the object class

III. GENETIC ALGORITHM

Genetic Algorithm is heuristic search algorithm. It uses biological evolution analogy of crossing over fittest chromosomes to generate offsprings. A fixed fraction of previous population (having higher fitness) and offsprings (produced from crossover) are evaluated against a fitness function for their suitability. A fixed fraction of population that ranks low in fitness are eliminated and again the fittest chromosomes are crossed over to obtain offsprings. This method is iterative. A stopping criterion needs to be mentioned in terms of number of iterations or threshold value of fitness, else the GA optimization could go forever with little or no optimization.

Following steps are being followed as part of GA optimization:

Encoding Optimization problem as Chromosome

Number of Filter				Size of Filter				Cost Value
Layer 1	Layer 2	Layer 3	Layer 4	Layer 1	Layer 2	Layer 3	Layer 4	

Fig 9: Chromosome Definition. Chromosome has two vectors corresponding to Number of Filter and Size of filter.

Given the number of layers for CNN, GA obtains the optimal values for "number of filters" and "size of filter" in each layer. Chromosome is defined in terms of two vectors of length equal to number of layers. These two vectors contain the values for number of filters and size of filter. Also, the cost value corresponding to each setting (number of filters and size of filter) is obtained and updated.

Define Fitness function

For each setting obtained from chromosome, a CNN is generated and then trained using the training data. The CNN thus obtained is tested with the complete set of training data. The classification accuracy is calculated in terms of percentage and then subtracted from 100 to obtain the cost value. Lower the cost value, higher the fitness.

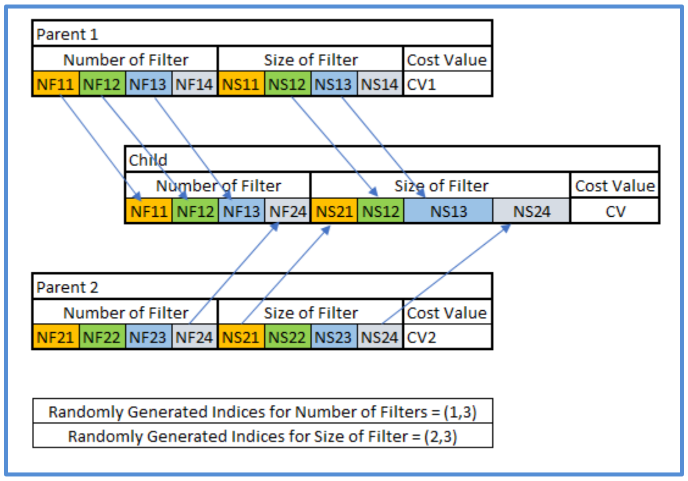
Define Initial Population

Initial population is a set of randomly generated chromosomes. Depending on the "maximum limit for the number of filters" and "size of filter", a random integer generator generates initial population. The size of initial population is defined by a parameter.

	Number of Filter			Size of Filters			Cost Value
Individual	Layer 1	Layer 2	Layer 3	Layer 1	Layer 2	Layer 3	(% Error)
1	96	81	4	15	10	19	97.5
2	23	27	9	9	3	13	4.166666667
3	81	70	77	7	17	9	94.16666667
4	83	63	15	2	1	2	1.666666667
5	46	12	3	16	8	15	97.5
6	46	46	48	10	17	9	97.5
7	91	4	78	3	13	1	4.166666667
8	54	1	96	19	16	19	97.5
9	15	16	19	13	19	20	97.5
10	72	74	91	9	5	4	5
11	12	82	15	6	17	7	5.833333333

Fig 10: Example of randomly generated initial population (parameters used for generation: number of layers = 3, maximum value for number of filters = 100, maximum size of filter = 20, initial population size = 11)

Define Crossover function
Two parent chromosomes are crossed over to generate the offspring chromosome. When chromosomes are defined in terms of bit strings then the offspring is generated using bit mask. However, in present case where the chromosome is composed of integers a cross over function needs to be defined.



Parent 1
Number of Filter
Size of Filter
Cost Value
NF11 NF12 NF13 NF14 NS11 NS12 NS13 NS14 CV1
Child
Number of Filter
Size of Filter
Cost Value
NF11 NF12 NF13 NF24 NS21 NS12 NS13 NS24 CV
Parent 2
Number of Filter
Size of Filter
Cost Value
NF21 NF22 NF23 NF24 NS21 NS22 NS23 NS24 CV2
Randomly Generated Indices for Number of Filters = (1,3)
Randomly Generated Indices for Size of Filter = (2,3)

Fig 11: Crossover operation

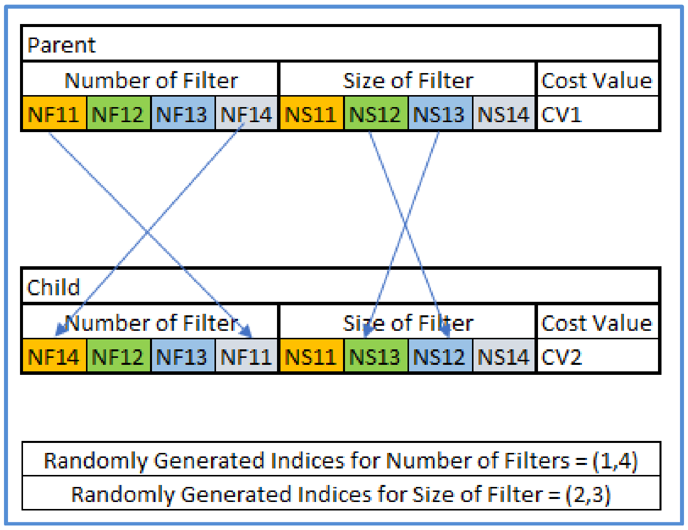
Parent 1: vector for "number of filters" has elements NF11, NF12, NF13, NF14, vector for "size of filter" has elements NS11, NS12, NS13, NS14

Parent 2: vector for "number of filters" has elements NF21, NF22, NF23, NF24, vector for "size of filter" has elements NS21, NS22, NS23, NS24.

Random index generator generates indices for cross over operation (in this case indices for number of filter are [1,3], indices for size of filter are [2,3]).
Using indices, the cross over operation generates the child chromosome. The direction of arrow in Fig 10, from parent to child shows the elements inherited by child from parents 1 and 2.

In this case random index generator has been used, however fixed index cross over can also be used. Author has not explored performance comparison for these two crossover methods.

Define Mutation function



Parent
Number of Filter
Size of Filter
Cost Value
NF11 NF12 NF13 NF14 NS11 NS12 NS13 NS14 CV1
Child
Number of Filter
Size of Filter
Cost Value
NF14 NF12 NF13 NF11 NS11 NS13 NS12 NS14 CV2
Randomly Generated Indices for Number of Filters = (1,4)
Randomly Generated Indices for Size of Filter = (2,3)

Fig 12: Mutation Operation

Parent has vector for “number of filters” as NF11, NF12, NF13, NF14; vector for “size of filters” as NS11, NS12, NS13, NS14.
Random Index Generator generates indices for mutation (mutation indices for number of filters generated are [1,4] and for size of filters as [2,3]). Using the indices, the values in the parent chromosome are interchanged to produce Child chromosome.

Generation

Each iteration in GA is called a generation. It involves elimination of least fit chromosomes, selecting the fittest chromosomes, performing crossover and mutation to generate the offsprings.

Population

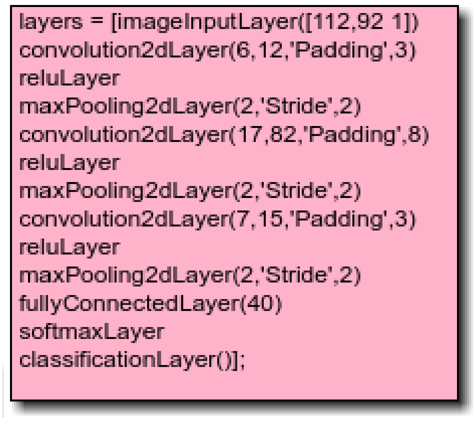
Population is a set of individual chromosomes. It consists of parent chromosomes carried over to next generation, offsprings generated as a result of crossover and mutation.

IV. SYSTEM ARCHITECTURE

The complete system for optimization for CNN using Genetic Algorithm consists of following:

(1) Genetic Algorithm block: Fig 13, block (1)
it creates a set of chromosomes that contain the configuration of CNN. This block depends on number of layers of CNN, maximum number of filters in each layer, maximum size of filter in each layer and size of population to generate a set of chromosomes.

(2) Generate CNN Configuration: Fig 13, block (2)
Depending on individual for a generation (that contains configuration for CNN), this block generates the configuration script(Matlab) for CNN as shown below:



```
layers = [imageInputLayer([112,92 1])
convolution2dLayer(6,12,'Padding',3)
reluLayer
maxPooling2dLayer(2,'Stride',2)
convolution2dLayer(17,82,'Padding',8)
reluLayer
maxPooling2dLayer(2,'Stride',2)
convolution2dLayer(7,15,'Padding',3)
reluLayer
maxPooling2dLayer(2,'Stride',2)
fullyConnectedLayer(40)
softmaxLayer
classificationLayer()];
```

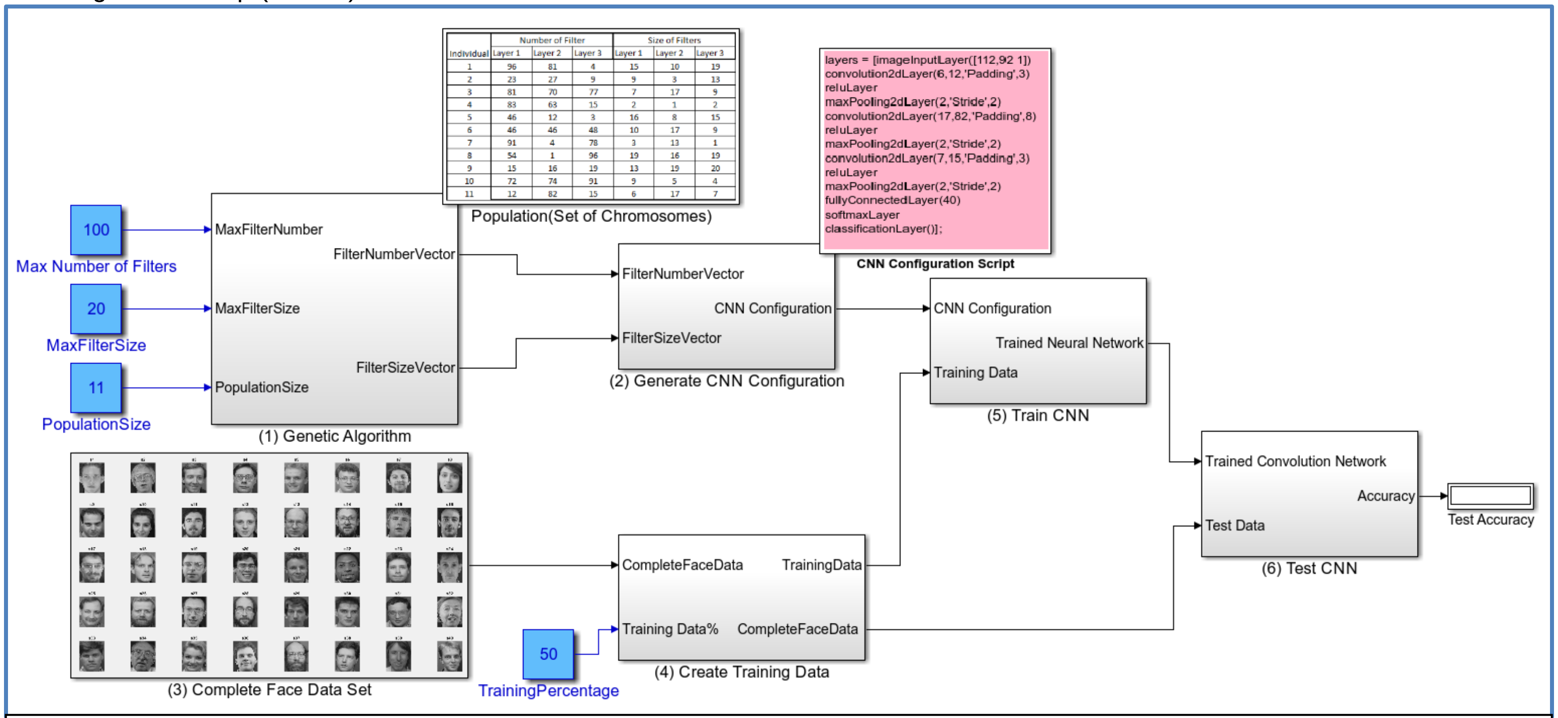
Fig 14: Generated CNN Configuration Script(Matlab)

(3) Complete Face Data Set: Fig 13, block (3)
It is a collection of faces for 40 people with 10 faces for each [4].

(4) Create Training Data: Fig 13, block (4)
The face data set received is divided into training and test data set based on the parameter “training percentage”. This block also randomizes the selection of faces for training and testing.

(5) Train CNN: Fig 13, block (5)
This block is responsible for creating CNN based on the configuration inputs and training data set. This block contains some fixed parameters such as learning rate (0.001), learning method (stochastic gradient descent), maximum epochs (25) and batch size (20).

(6) Test CNN: Fig 13, block (6)
This block receives the complete face data set and the trained CNN network and provides the classification accuracy percentage as output.



Individual	Number of Filter			Size of Filters		
	Layer 1	Layer 2	Layer 3	Layer 1	Layer 2	Layer 3
1	96	81	4	15	10	19
2	23	27	9	9	3	13
3	81	70	77	7	17	9
4	83	63	15	2	1	2
5	46	12	3	16	8	15
6	46	46	48	10	17	9
7	91	4	78	3	13	1
8	54	1	96	19	16	19
9	15	16	19	13	19	20
10	72	74	91	9	5	4
11	12	82	15	6	17	7

```
layers = [imageInputLayer([112,92 1])
convolution2dLayer(6,12,'Padding',3)
reluLayer
maxPooling2dLayer(2,'Stride',2)
convolution2dLayer(17,82,'Padding',8)
reluLayer
maxPooling2dLayer(2,'Stride',2)
convolution2dLayer(7,15,'Padding',3)
reluLayer
maxPooling2dLayer(2,'Stride',2)
fullyConnectedLayer(40)
softmaxLayer
classificationLayer()];
```

Fig 13: System Architecture for GA Optimization of CNN

V. EXPERIMENTS AND RESULTS

Data Set

The data set used for training and testing CNN were obtained from Face Data Set of AT&T Laboratories at Cambridge University [4]. The data set consists of 10 face images each for 40 different people. These images were obtained in different lighting conditions against a dark background. The images were taken with different facial expressions (smiling/not smiling/eyes open/closed) and facial details (glasses/no glasses).

Fig 15: Face Database (40 different people, 10 images each). [4]

GA optimization algorithm for CNN is allowed to run for 2 and 3 convolutional layers CNN. The limits were specified for the maximum number of filters in each layer and maximum size of filter. Maximum number of generations allowed for GA is specified.

Experiment 1

Settings: Number of convolutional layers = 2; maximum number of filters = 100; maximum size of filter = 20; maximum number of generations for GA = 10

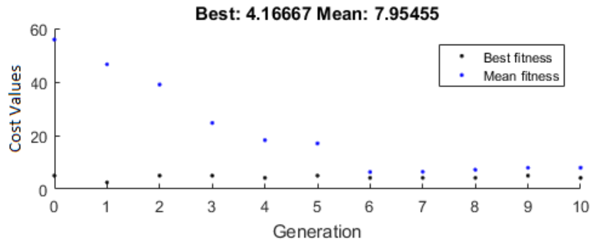
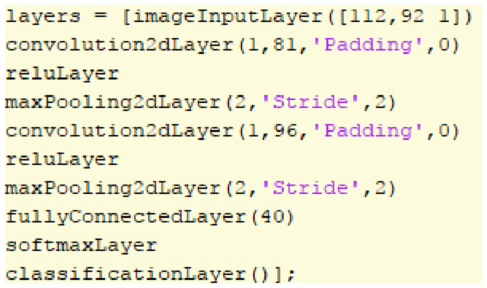


Fig 16: Cost value with Generation for 2 convolutional layer CNN.

It can be observed that the cost value (classification error) reduced with increase in number of generation and saturated after 6 generations (no improvement) The configuration obtained for CNN is as follows:



```
layers = [imageInputLayer([112,92 1])
convolution2dLayer(1,81,'Padding',0)
reluLayer
maxPooling2dLayer(2,'Stride',2)
convolution2dLayer(1,96,'Padding',0)
reluLayer
maxPooling2dLayer(2,'Stride',2)
fullyConnectedLayer(40)
softmaxLayer
classificationLayer()];
```

Fig 17: CNN configuration obtained using GA (number of filters for first and second layer are 81 and 96 respectively, size of filters for first and second layer are 1 and 1 respectively)

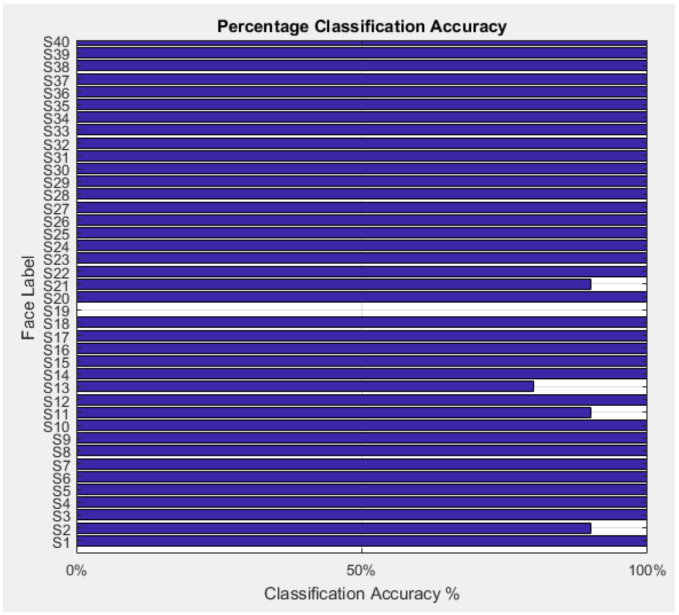


Fig 18: Percentage classification accuracy for each face label.

Classification accuracy over complete data set is 96.25%

Experiment 2

Settings: Number of convolutional layers = 3; maximum number of filters = 100; maximum size of filter = 20; maximum number of generations for GA = 10

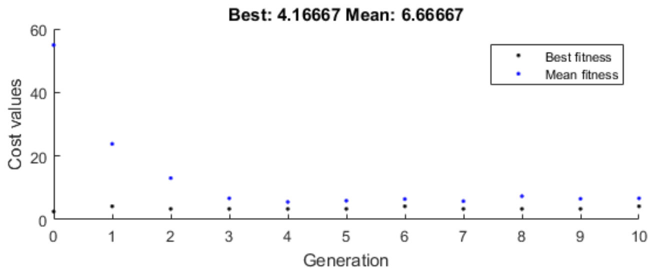
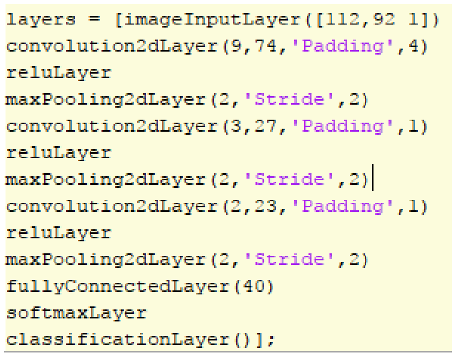


Fig 19: Cost values with Generations for 3 convolutional layer CNN

It can be observed that the cost value (classification error) reduced with increase in number of generations and saturated after 4 generations (no improvement).

The configuration obtained for CNN is as follows:



```
layers = [imageInputLayer([112,92 1])
convolution2dLayer(9,74,'Padding',4)
reluLayer
maxPooling2dLayer(2,'Stride',2)
convolution2dLayer(3,27,'Padding',1)
reluLayer
maxPooling2dLayer(2,'Stride',2)
convolution2dLayer(2,23,'Padding',1)
reluLayer
maxPooling2dLayer(2,'Stride',2)
fullyConnectedLayer(40)
softmaxLayer
classificationLayer()];
```

Fig 20: CNN configuration obtained using GA (number of filters for first, second and third layer are 74, 27 and 23 respectively, size of filter for first, second and third layer are 9, 3 and 2 respectively).

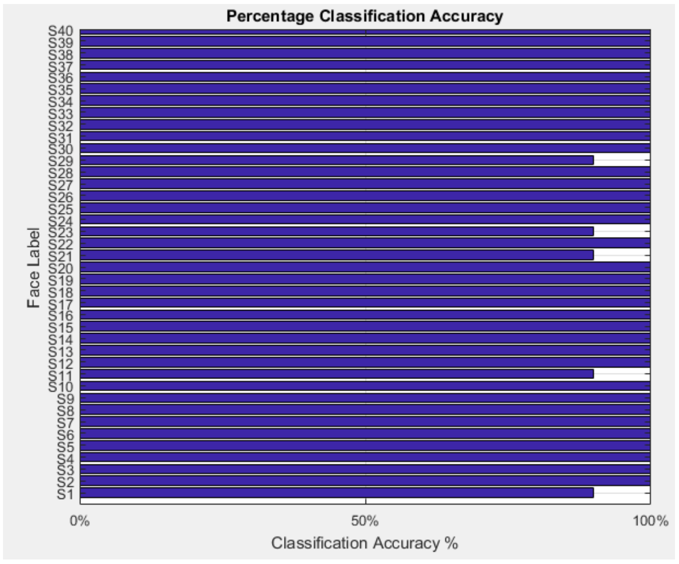


Fig 21: Percentage Accuracy for each face label

Classification accuracy over complete data set is 98.75%.

VI. CONCLUSION

In conclusion, this project has demonstrated application of Genetic Algorithm(GA) for searching optimal parameter setting for Convolutional Neural Network. It was observed that as the generations for GA increase the classification error converges to zero. However, after certain number of generations there is no improvement.

This project has created an automated setup for finding optimal setting for CNN. The user is required to plan for the number of convolutional layers then specify the maximum number of filter, maximum size of filter for convolutional layer and maximum number of generations for GA and let the system run. GA presents the setting for CNN once it has completed the running for generations specified.

Parameters obtained for CNN show sufficiently good classification accuracy.

ACKNOWLEDGMENTS

This project has been taken as a course requirement for ECE-588: Robot Vision as part of Master's program in Electrical Engineering at University of Michigan-Dearborn. Author would like to thank Prof Larry Sieh, for all the informative and detailed classes taken as part of this course. Author would also like to thank Prof Yi Lu Murphey for class on Intelligent System. Several concepts learnt for Genetic Algorithm and Neural Networks in that class have been used in this project.

REFERENCES

1. Neural-Network-Toolbox: https://www.mathworks.com/help/nnet/convolutional-neural-networks.html
2. Image-Processing-Toolbox: https://www.mathworks.com/products/image.html
3. Global-Optimization-Toolbox: https://www.mathworks.com/products/global-optimization.html
4. Face Database, AT&T Laboratories, Cambridge University: http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html
5. Aarshay Jain, Deep Learning for Computer Vision – Introduction to Convolution Neural Networks https://www.analyticsvidhya.com/blog/2016/04/deep-learning-computer-vision-introduction-convolution-neural-networks
6. A. Krizhevsky, I. Sutskever, G. E. Hinton, "ImageNet classification with deep convolutional neural networks", Advances in neural information processing systems pp. 1097-1105 2012
7. Convolutional Neural Network for Visual Recognition, http://cs231n.github.io/convolutional-networks/

APPENDIX

(1) GA_ConvLearn.m
This file is responsible for creating function handle to create population, crossover, fitness functions of GA and invoking Genetic Algorithm function of Matlab

```
convLayers = 4;
convParams = 2*convLayers;

maxNumFilters = 100;
maxFilterSize = 20;

populationRange = max(maxNumFilters,maxFilterSize);

ga_call_create_population = @(NVARS,FitnessFcn,options)
create_initial_population...
    (NVARS,FitnessFcn,options,maxNumFilters,maxFilterSize);

ga_call_crossover_population =
@(parents,options,NVARS,FitnessFcn,thisScore,thisPopulation) ...

crossover_population(parents,options,NVARS,FitnessFcn,thisScore,thisPopulation,conv
Layers);

call_ga_fitness = @(x)
ga_fitness(x,convLayers,trainFaceData,testFaceData,imgLength,imgWidth);

%% Genetic Algorithm Options Setup
% First, we will create an options container to indicate a custom data type
% and the population range.
options = optimoptions(@ga, 'PopulationType', 'custom','InitialPopulationRange',
...
                            [1;populationRange]);
%%
% We choose the custom creation, crossover, mutation, and plot functions
% that we have created, as well as setting some stopping conditions.
options = optimoptions(options,'CreationFcn',ga_call_create_population, ...
                        'CrossoverFcn',ga_call_crossover_population, ...
                        'MutationFcn',@mutate_population, ...
                        'MaxGenerations',10,'PopulationSize',11, ...
                        'MaxStallGenerations',100,'UseVectorized',true);
options = optimoptions(options,'PlotFcn', {  @gaplotbestf @gaplotscores });
options = optimoptions(options,'UseParallel', true);
%%
% Finally, we call the genetic algorithm with our problem information.

numberOfVariables = convParams;
[x,fval,reason,output] = ...
    ga(call_ga_fitness,numberOfVariables,[],[],[],[],[],[],[],options)
%%
%save the final configuration for CNN obtained
save('finalConfig.mat','x');
```

(2) create_initial_population.m
This function is responsible for creating initial population. It uses random integer generation function to generate initial population within the limits specified for number of filters and size of filters.

```
function pop =
create_initial_population(NVARS,FitnessFcn,options,maxNumFilters,maxFilterSize)
% Create the initial population for GA.
% Create Cell of populationSize
% each cell is an individual containing an array
% first two elements of array are number of filters in each layer
% next two elements are the filter size used in each layer

totalPopulationSize = options.PopulationSize;
n = NVARS/2;
pop = cell(totalPopulationSize,1);
disp('Starting Generation of Initial Population');
for i = 1:totalPopulationSize
    pop{i} = [randi([1 maxNumFilters],[1 n]) randi([1 maxFilterSize],[1 n])];
end
save('initialPop.mat','pop');
disp('Generation of Initial Population Completed');
```

(3) crossover_population.m
argument “parents” gives the indices of the population to be considered for cross over. “thisPopulation” argument gives the value of population. Crossover function using random index generator gets the indices for cross over between two parents to produce “xoverKids”.

```
function xoverKids  =
crossover_population(parents,options,NVARS,FitnessFcn,thisScore,thisPopulation,convLaye
rs)
persistent Generation;
if(isempty(Generation))
    Generation = 0;
end
Generation = Generation + 1;
dispString = strcat('Creating Cost values for Generation : ',string(Generation));
disp(dispString);

nKids = length(parents)/2;
xoverKids = cell(nKids,1);
index = 1;

for i=1:nKids

    dispString = strcat('Creating Cost values for Generation : ',string(Generation), '
,Individual:',string(i));
    disp(dispString);

    parent = thisPopulation{parents(index)};
    parent = [fliplr(parent(1:convLayers)) fliplr(parent(convLayers+1:2*convLayers))];

    % each individual is going to have first

    p_numFilters = randi([1 convLayers],[1 2]);
    p1_numFilters = min(p_numFilters);
    p2_numFilters = max(p_numFilters);

    child = thisPopulation{parents(index+1)};

    if(p1_numFilters == p2_numFilters)
        %do nothing
    else
        child(p1_numFilters:p2_numFilters) = parent(p1_numFilters:p2_numFilters);
end
    p_sizeFilters = randi([convLayers+1 2*convLayers],[1 2]);
    p1_sizeFilters = min(p_sizeFilters);
    p2_sizeFilters = max(p_sizeFilters);

    if(p1_numFilters == p2_numFilters)
        %do nothing
    else
        child(p1_sizeFilters:p2_sizeFilters) = parent(p1_sizeFilters:p2_sizeFilters);
    end

    xoverKids{i} = child;
    index = index + 2;
end
disp('End of crossover permutation');
```

(4) mutate_population.m

argument “parents” gives the indices of the population to be considered for cross over. “thisPopulation” argument gives the value of population. Using random number generator this function finds the indices of the parents to be used for mutation and then interchanges them to obtain “mutationChildren”.

```
function mutationChildren = mutate_population(parents ,options,NVARS,FitnessFcn, state,
thisScore,thisPopulation,mutationRate)

convLayers = 3;
mutationChildren = cell(length(parents),1);
for i=1:length(parents)
    parent = thisPopulation{parents(i)};
    child = parent;
    p_numFilter = randi([1 convLayers],[1 2]);

    child(p_numFilter(1)) = parent(p_numFilter(2));
    child(p_numFilter(2)) = parent(p_numFilter(1));

    p_sizeFilter = randi([convLayers+1 2*convLayers],[1 2]);

    child(p_sizeFilter(1)) = parent(p_sizeFilter(2));
    child(p_sizeFilter(2)) = parent(p_sizeFilter(1));

    mutationChildren{i} = child;
end
disp('End of Mutation Function');
save('GA_Population.mat','thisPopulation','thisScore');
```

(5) ga_fitness.m

This function is responsible for calculating the cost value for individual chromosomes of population. argument “x” contains the population. “convLayers” contains the number of convolution layers in CNN. “trainFaceData” and “testFaceData” contain the training and test face images data. “imgLength” and “imgWidth” are the length and width respectively of image.
This function iterates through each chromosome of population “x” and using the configuration of CNN contained in it, trains CNN and then tests it. The it subtracts the accuracy value from 100 to obtain the error which is returned to the argument “scores”.

```
function scores =
ga_fitness(x,convLayers,trainFaceData,testFaceData,imgLength,imgWidth)

persistent genNumber;
if(isempty(genNumber))
    genNumber = 1;
end
scores = zeros(size(x,1),1);

for i=1:size(x,1)
    %defining the layers
    convConfig = x{i};

genConvnetConfig(convLayers,convConfig(1:convLayers),convConfig(convLayers+1:2*convLay
ers),imgLength,imgWidth);
    run('tempScript.m');

    %specify the training options
    options = trainingOptions('sgdm','MaxEpochs',25,'MiniBatchSize',20,...
        'InitialLearnRate',0.001,'verbose',1);

    %train the network using training data
    faceConvnet = trainNetwork(trainFaceData,layers,options);

    YTest = classify(faceConvnet,testFaceData);
    TTest = testFaceData.Labels;

    %calculate accuracy
    accuracy = 100*sum(YTest == TTest)/numel(TTest);
    scores(i) = 100 - accuracy;

    dispString = strcat('======Generation : ',string(genNumber),'========Individual :
',string(i),'============');
    disp(dispString);
    dispAccuracy = strcat('Accuracy : ', string(accuracy));
    disp(dispAccuracy);
    filterNumbersString = strcat('Filter Numbers = ',
string(convConfig(1:convLayers)));
    filterSizesString = strcat('Filter Sizes = ',
string(convConfig(convLayers+1:2*convLayers)));
    disp(filterNumbersString);
    disp(filterSizesString);

end
save('initialPop.mat','scores','-append');
genNumber = genNumber + 1;
```

(6) plotTestResults.m

This function loads the trained convolution network and then applies the face data base to it and then obtains the accuracy corresponding to each label of image.

```
%%
%find categories in faceData.Labels
close all;clear all;
load('faceConvnet.mat');
faceDataLabels = categories(faceData.Labels);
row = size(faceDataLabels,1);
testAccuracy = zeros(row,1);

for i=1:row
    tempDataPath = strcat(faceDatasetPath,'/',faceDataLabels{i});
    tempDataStore = imageDatastore(tempDataPath);
    tempRow = size(tempDataStore.Files,1);
    testResult = classify(faceConvnet,tempDataStore);
    tempAccuracy = sum(testResult == faceDataLabels{i})/tempRow;
    testAccuracy(i) = tempAccuracy;
end
%create label set
labelSet = cell(row,1);
for i=1:row
    labelSet{i} = strcat('S',string(i));
end

figure;
barh((1:row),testAccuracy); ylim([0 40]);
grid on;
xticks([0 0.5 1]);
xticklabels({'0%','50%','100%'});
yticks([1:1:40]);
yticklabels(labelSet);
ylabel('Face Label');xlabel('Classification Accuracy %');
title('Percentage Classification Accuracy');
```

(6)(a) plotImageResults.m

This function is used by the script presentTestResults.m
It is responsible for plotting the images corresponding the data labels.

```
function plotImageResults(tempDataStore,testResults,actualLabel)
    offset = 0;
    count = 1;
    flag = 0;
    while(count<=10)
        subplot(5,4,count+offset);
        tempImg = imread(tempDataStore.Files{count});
        if(testResults(count) == actualLabel)
            borderColor = 'g';
        else
            borderColor = 'r';
        end
        tempImgWithShape = insertShape(tempImg,'Rectangle',[2 2 90
110],'Color',borderColor,'Linewidth',3);
        imshow(tempImgWithShape);
        title(string(testResults(count)));
        count = count + 1;

        flag = xor(flag,1);

        if(flag == 0)
            offset = offset + 2;
        end
    end
end
```

(7) presentTestResults.m

This script is responsible for obtaining the classification accuracy of each data class. It also plots the accuracy corresponding to classification of each data class.

```
%%
%source of training faces:
http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html
%face recognition using convolutional neural network
clear all;close all;clc;

faceDatasetPath = 'C:\Work\02_Study\17_RobotVision_ECE588\FinalProject\att_faces';
faceData =
imageDatastore(faceDatasetPath,'IncludeSubfolders',true,'LabelSource','foldernames');
pause_flag = 0;

numLabels = size(categories(faceData.Labels),1);

%create label set
labelSet = cell(numLabels,1);
for i=1:numLabels
    labelSet{i} = strcat('S',string(i));
end

load('faceConvnet.mat');
figure;
for i=1:numLabels
    tempDataPath = strcat(faceDatasetPath,'\','s',string(i));
    tempDataStore = imageDatastore(char(tempDataPath));
    testResult = classify(faceConvnet,tempDataStore);
    numResults = size(testResult,1);
    labelCount = zeros(numLabels,1);
    for j=1:numResults
        for k=1:numLabels
            labelCount(k) = labelCount(k) + (testResult(j) == strcat('s',string(k)));
        end
    end
    actualLabel = strcat('s',string(i));
    subplot(1,2,1);
    plotImageResults(tempDataStore,testResult,actualLabel);
    subplot(1,2,2);
    bar((1:1:40),labelCount*100/sum(labelCount),'r');hold on;
    xlim([0 42]);ylim([0 105]);
    bar(i,labelCount(i)*100/sum(labelCount),'g');hold off;
    xticks([1:1:40]);
    xtickangle(90);
    xticklabels(labelSet);
    ylabel('Face Label');xlabel('Classification Accuracy %');
    title(strcat({'Actual Label: '},{string(actualLabel)}));
    xlabel('Label Number');ylabel('Percentage Accuracy');
    if(pause_flag == 0)
        pause;
        pause_flag = 1;
    end
    pause(0.1);

end
```