EE 595 – MACHINE INTELLIGENCE AND SMART SYSTEMS

MATARAARACHCHI M.A.R.P.

E/15/224

SEMESTER 08

07.05.2021

Pre-Calculation for Face Number

Since I completed the assignment before the final exams, I have constructed the algorithm to recognize all the 15 faces. I have informed the course coordinator regarding this.

Identifying the problem and dataset

The problem is to develop a face recognition algorithm using the provided data set.

Specifications of the data set.

- Name: YALE_32x32
- Total examples: 165
- Number of different faces: 15
- Number of images per person: 11

Each image is a 32x32pixel gray scale image. Therefore, altogether there are 1024 features per example.

The problem is carried out in 7 steps.

- Step 01: Visualize the data set
- Step 02: Reserve some images to use later.
- Step 03: Notice that there are more features than examples.
- Step 04: Principal Component Analysis (To reduce features)
- Step 05: Artificial Data Synthesis (To increase examples)
- Step 06: Neural Network Architecture
- Step 07: Training and Results

Other topics

- Conclusion
- Future Works
- MATLAB Code
- References

Step 01: Visualize the data set

After inspecting the 11 pictures of each person it was found that the 11 pictures followed a certain pattern. For example, the confused face appeared at the 10th place, the winking face appeared at the 11th place, the face with the glasses appeared at the 2nd place, etc. Therefore, to avoid any bias in the dataset (which will result in the Neural Net getting over-fitted) pictures of each person were shuffled in a random order. After shuffling, the 11 pictures of the 1st person looked like the following.

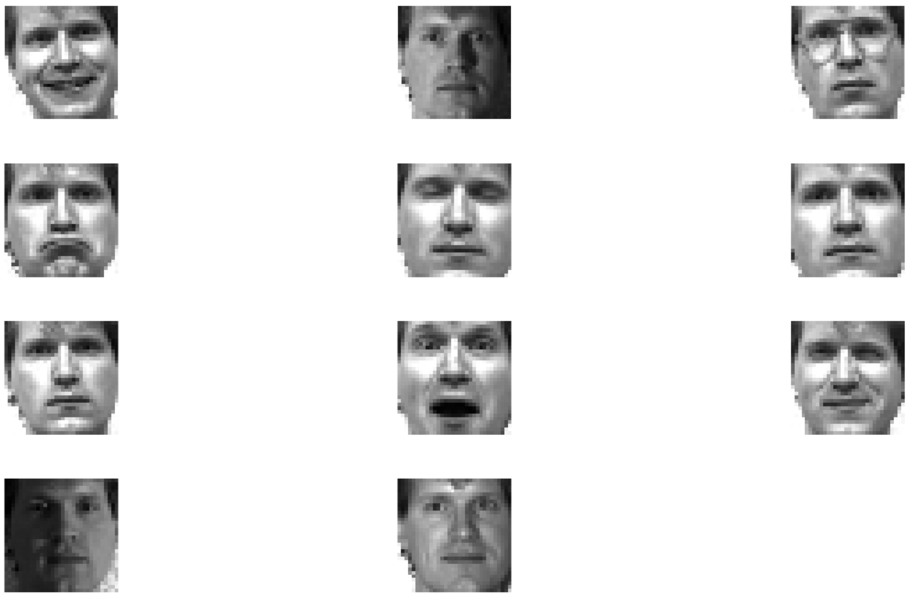


Figure 1: The 11 faces of the first person after shuffling

And the 11 pictures of the 2nd person looked like the following.

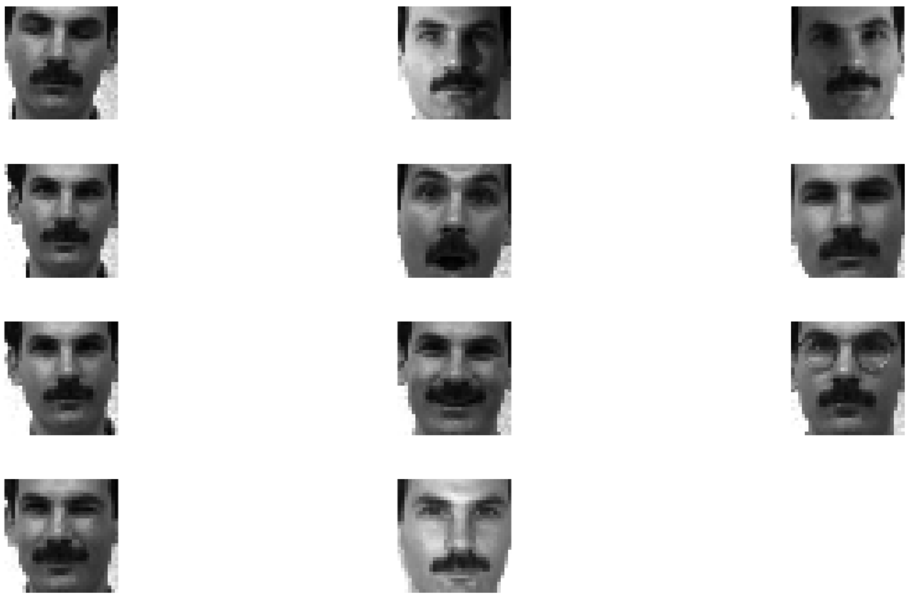


Figure 2: The 11 faces of the second person after shuffling

As seen from the above pictures, the order of the facial expressions is not similar in person 1 and person 2.

Step 02: Reserve some images to use later.

One picture from each person is reserved for later use. These 15 images are kept as a Testing data set which will not be fed into the Training algorithm at any point. This picture set is not even used as a test data set in the learning algorithm.

The pictures reserved for later use are as follows.

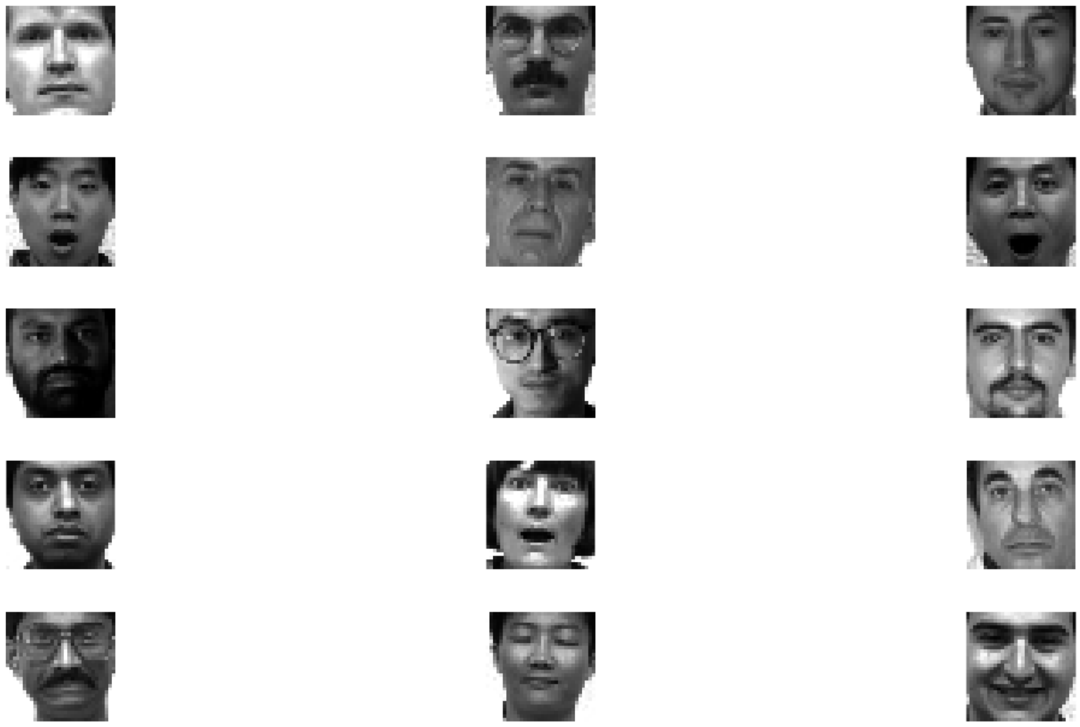


Figure 3: Images reserved for later testing

Step 03: Notice that there are more features than examples.

By examining the data set, we see that there are more features than examples.

- 1024 Features
- 165 Examples

This results in a problem commonly known as 'the curse of dimensionality'. When there are more features than examples the machine learning model is destined to over-fit. This happens because the lower number of examples does not occupy the whole feature space. i.e., Feature space is sparse. To overcome this problem we can use 2 methods.

1. Reduce the number of features
2. Increase the number of training examples

In the following steps both of these methods are used to train the model.

To reduce the number of features, Principal Component Analysis (PCA) is used.

To increase the number of training examples, artificial data synthesis is used.

Step 04: Principal Component Analysis (To reduce features)

Principle component analysis (PCA) is used to reduce the number of features in the dataset. By reducing the features we reduce the number of dimensions in the feature space. Thereby, the examples (samples) can occupy a considerable amount of the feature space.

PCA essentially means finding a new basis for the dataset. The vector space spanned by this basis is commonly called as facespace in the facial recognition field.

To perform PCA, first we should subtract the mean face of the dataset from all the faces. Then, take the singular value decomposition (SVD) of the matrix sigma, where sigma is the covariance matrix of training data. Note that we create the basis vectors by using only the training data. Performing SVD is like taking the Fourier series expansion of a signal. Just like we can look at the most dominant frequencies in a signal by Fourier expansion we can look at the most dominant basis vectors by performing SVD. After performing SVD, we choose only the dominant basis vectors and neglect the less significant ones. This is like passing the images through a low pass filter. After finding the new basis we reduce the dimension of all the data including the test data and validation data.

However we should do this while preserving the details in the image. If an image is represented with a very low number of dimensions it may not look similar to the original image. To preserve the quality of the image, we reduce to a dimension where at least 90% of the variance is preserved. This is done to preserve the information content of the image. In matlab singular value decomposition can be done using the *svd()* function.

In this project we explore several dimensions from 64 upwards. That means, images previously represented by 1024 data points can now be represented by only 64 data points. Roughly 96.5% variance is preserved when reduced to 64 dimensions.

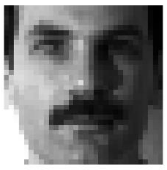


Figure 4.1: Original face with 1024 dimensions

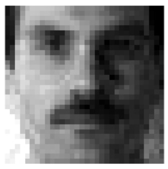


Figure 4.2: Reconstructed face after PCA

As seen from the two images most of the facial features are preserved after going through PCA.

The mean face of the data set can also be visualized.

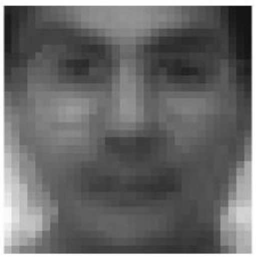


Figure 5: Mean (Average) face of all the faces

Additionally, we can even visualize the basis vectors used to represent these faces. This is also a good advantage of using PCA. Following are the 15 most dominant basis vectors in the facespace out of the 64 basis vectors.

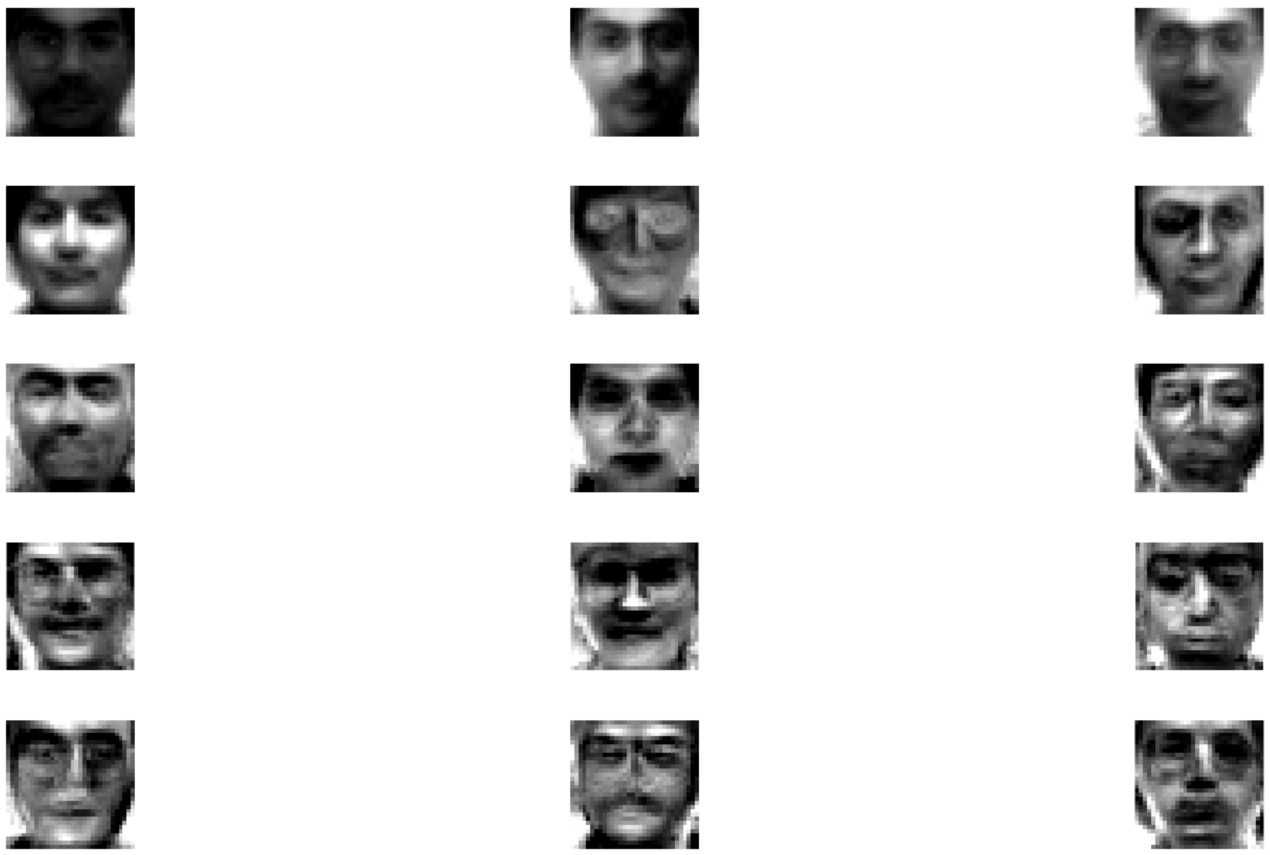


Figure 6: First 15 basis vectors of the facespace (Principal Components)

Step 05: Artificial Data Synthesis (To increase examples)

In order to increase the number of training examples, we create new data from our data set. In the beginning we only have 150 examples (we reserved 15 for testing). We create data by two methods,

1. Changing the brightness of the images (bright and dark images).
2. Changing the brightness of the image by scaling (bright parts get brighter, dark parts get darker).

By doing this, we create,

- 4 sets of bright images
- 4 sets of dark images
- 4 sets of scaled images

With the original data set we now have 13 data sets. Now, the total examples have now increased to 1950 examples (13x150).

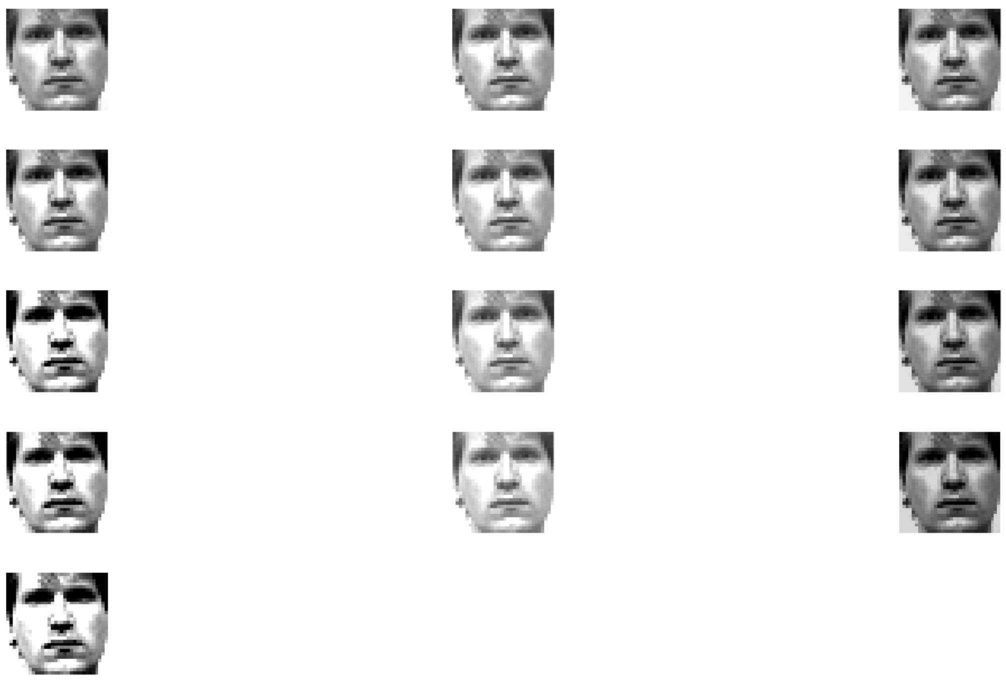


Figure 7: Different examples generated by the same face (First image is the original)

Step 06: Neural Network Architecture

Type of Network

Instead of the *feedforwardnet()* type network a *patternnet()* was used to build the model. This is because our problem is a classification problem. A pattern recognition network performs well in this situation. However when using *patternnet(),* the ground truth must be given as a Boolean vector.

Hidden layers and neurons

After several trial and error sessions it was found that adding more than 1 hidden layer does not improve the performance of the model. By adding more than 1 layer the performance of the model tend to decrease. Therefore only one hidden layer was added. Several models are run by varying the number of neurons in that hidden layer.

Training function

After reading through the matlab documentation scaled conjugate gradient back-propagation training function was used to train the model. In the documentation it was stated that it performed well for classification problems. And the computational time with that function is very low compared to other functions.

Scaled conjugate gradient back-propagation function in matlab can be set by,

```
net.trainFcn = 'trainscg';
```

Train, Test and Validation data

After creating artificial data, the data set is divided into Training, testing and validating data sets. Testing and validating data are chosen such that they cover all 15 different faces. Inserting the data sets into the model can be done by setting,

```
net.divideFcn = 'divideind';
```

Now, we have to give the indices of datasets as,

```
net.divideParam.trainInd = trn; %Training data
net.divideParam.testInd = ts;   %Testing data
net.divideParam.valInd = cv;    %Validation data
```

Where *trn, ts, cv* are the indices of training, testing, validating data sets respectively.

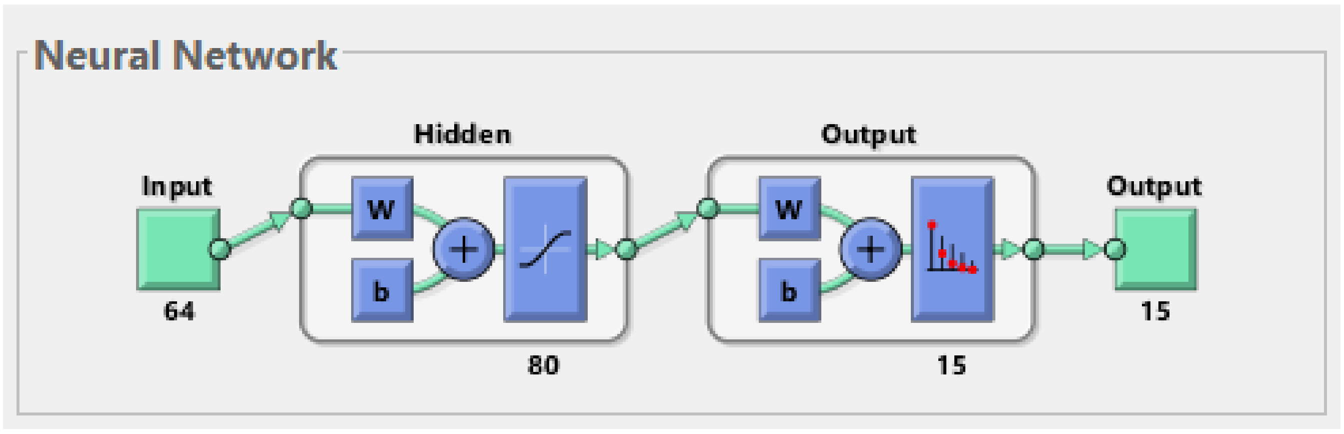


Figure 8: Network Architecture

Step 07: Training and Results

Initially, a model of 64 Dimensions and 80 neurons (in the hidden layer) was run. After training the model we see the performance of our neural network as follows.

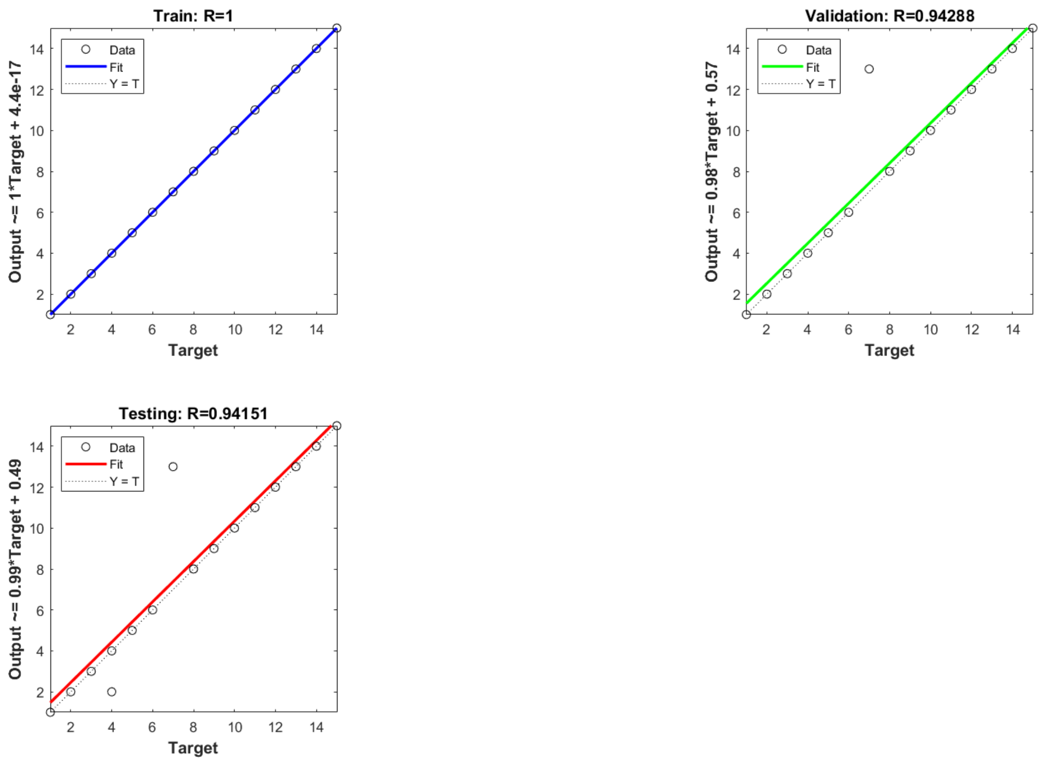


Figure 9: Regression Plot

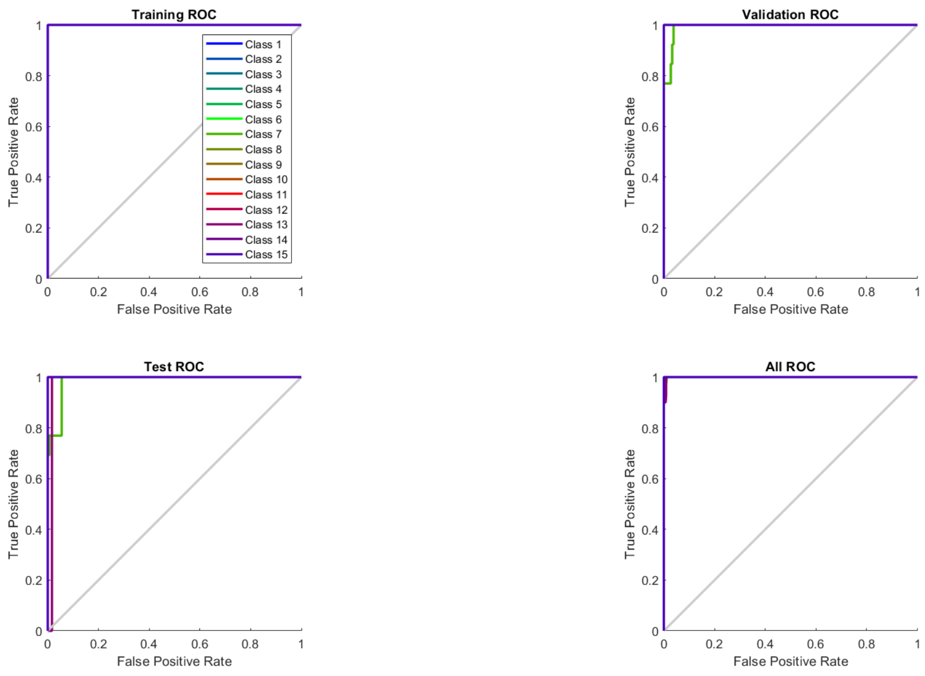


Figure 10: ROC Plot

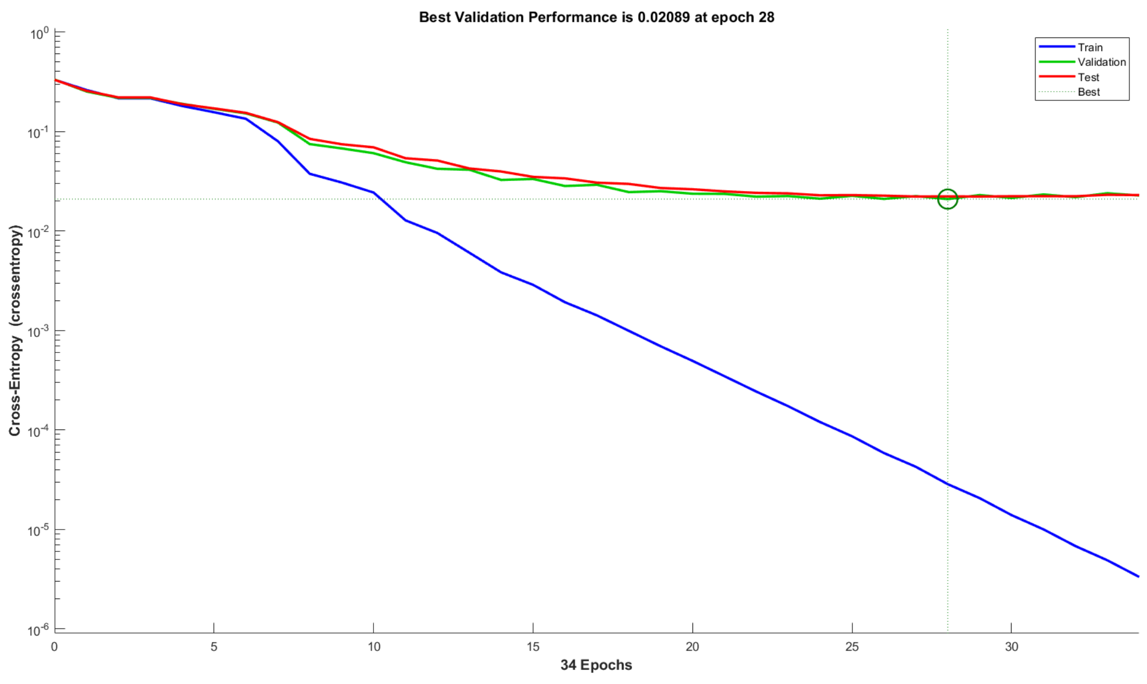


Figure 11: Error Convergence Plot

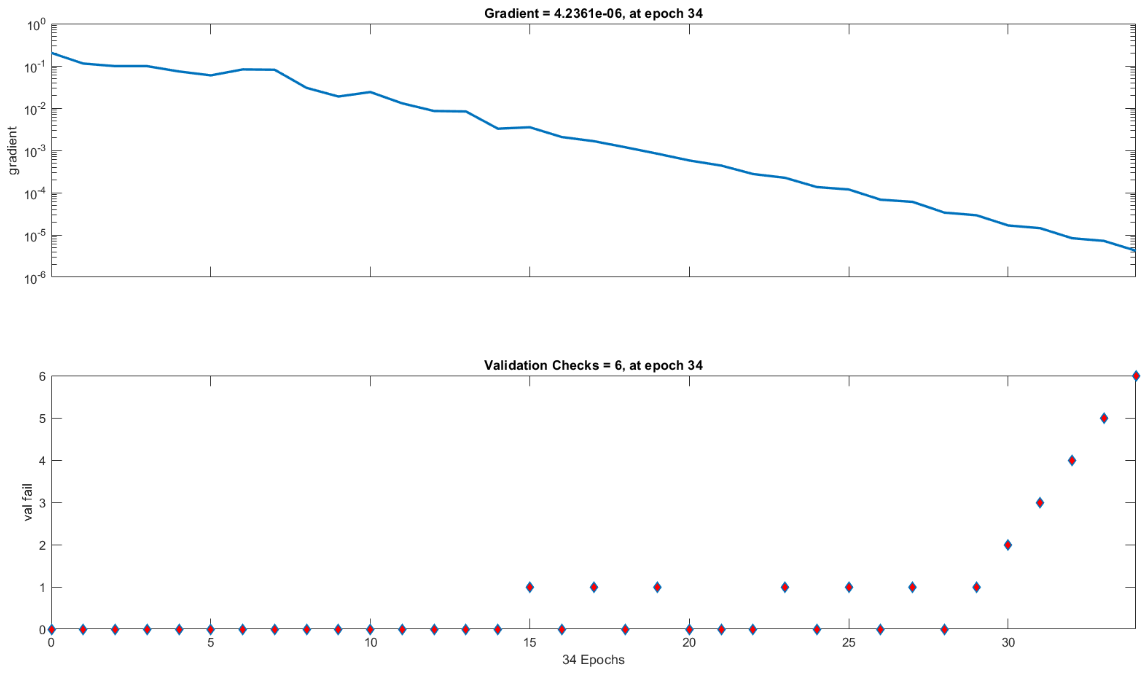


Figure 12: Training state plots

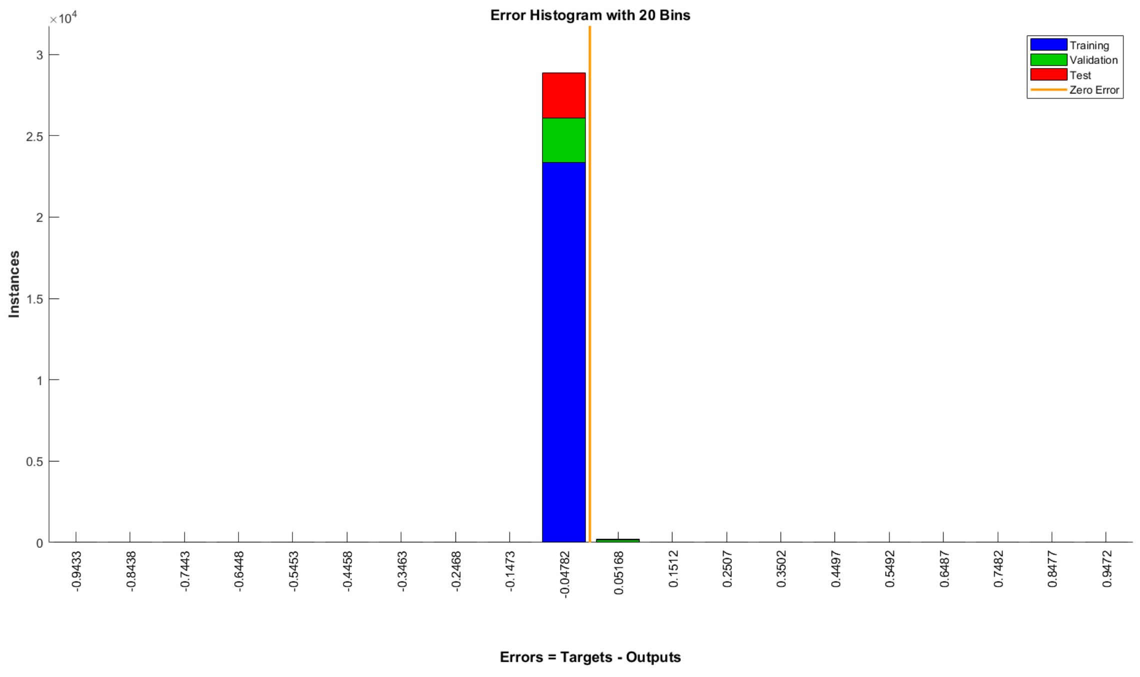


Figure 13: Error histogram

From the above plots we see that the model gives good results. The ROC and Regression plots suggest that the training, test and validation data have less error which is confirmed by the error convergence plot. Also, from the error histogram we see that those errors have a small variance.

But this is only after training the model once. When in reality, the performance of a model actually depends on the initial weights and biases defined on the model randomly. Therefore, the accuracy is subjected to change every time when the model is trained. To avoid this, we train the same model for 100 times and get the average accuracy. This figure is more reliable than what we get from running the model only once.

Also, there are two factors we can change in every model,

1. The number of dimensions to which the data set is reduced.
2. The number of neurons in the hidden layer.

We check the average accuracy of the 15 images reserved in step 02. Note that we didn't feed these 15 images into the neural network in any way (not even as test data for training).

In all the cases the model is run for 100 times and the average accuracy is plotted. We change the dimensions and number of neurons in the following manner and check for average accuracy.

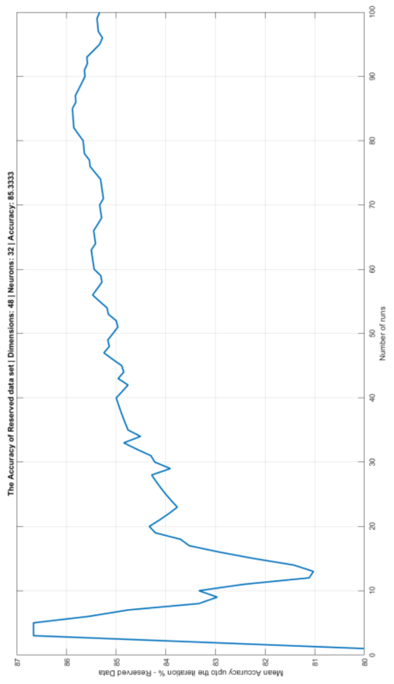
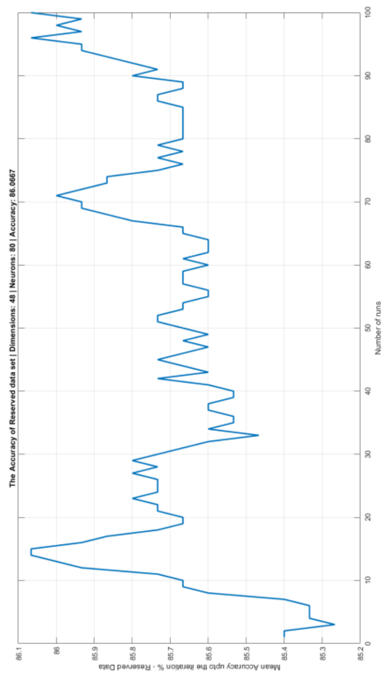
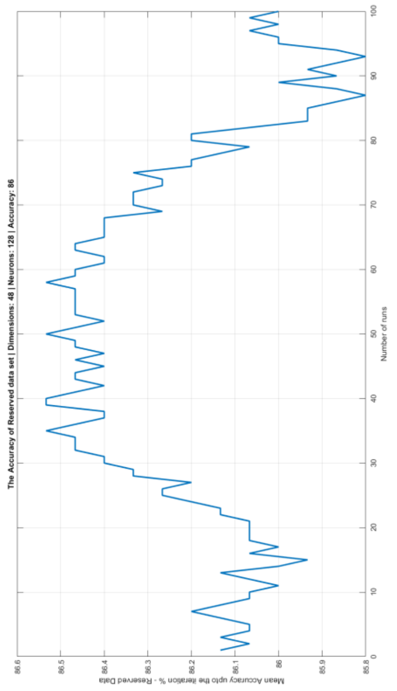
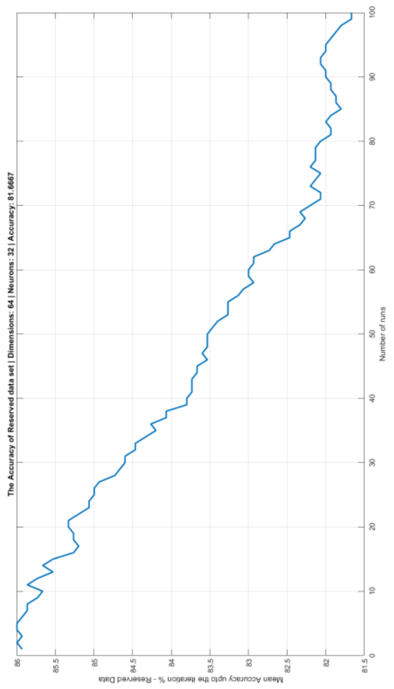
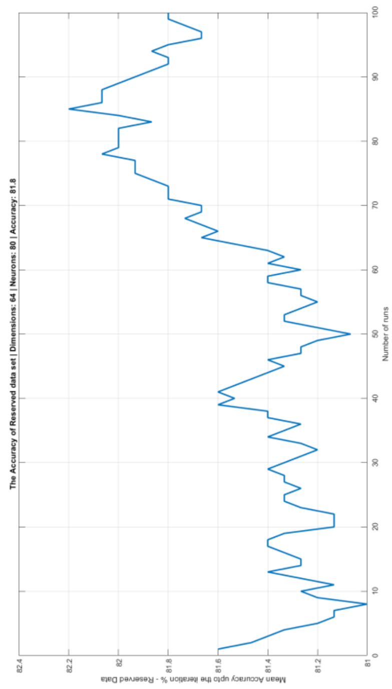
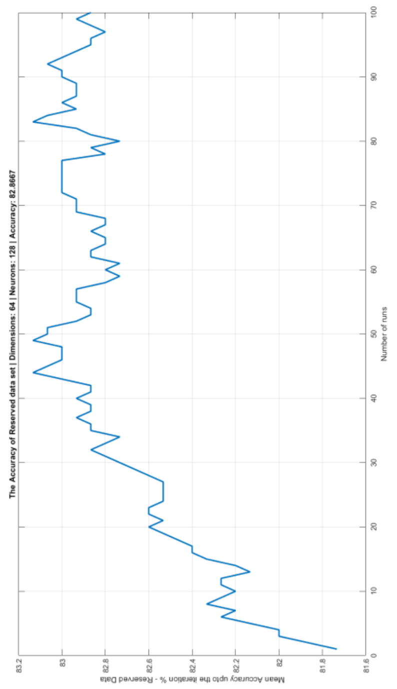
- Dimensions (D): 48, 64, 128, 256
- Neurons (N): 32, 80, 128

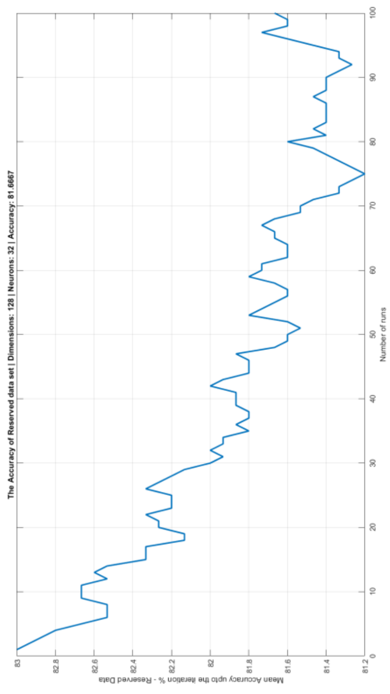
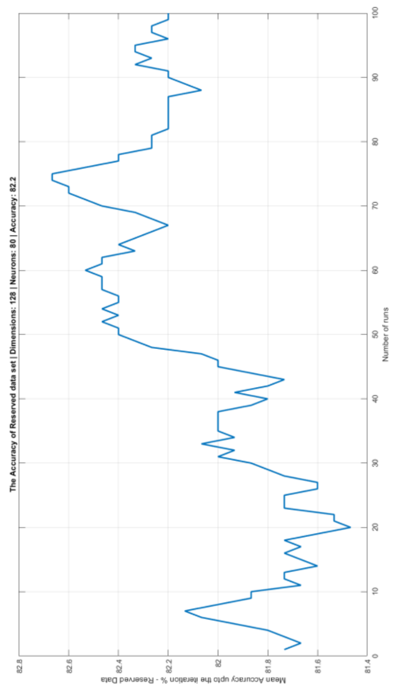
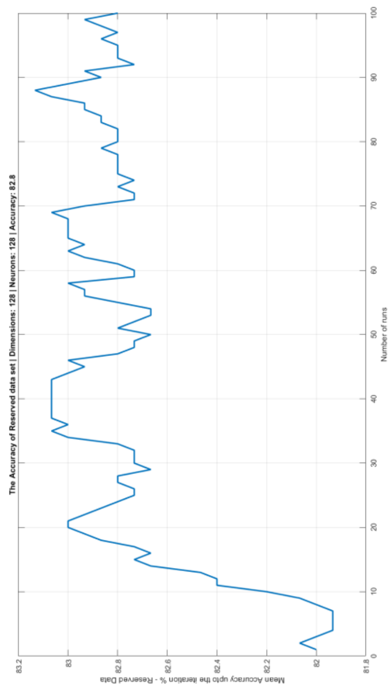
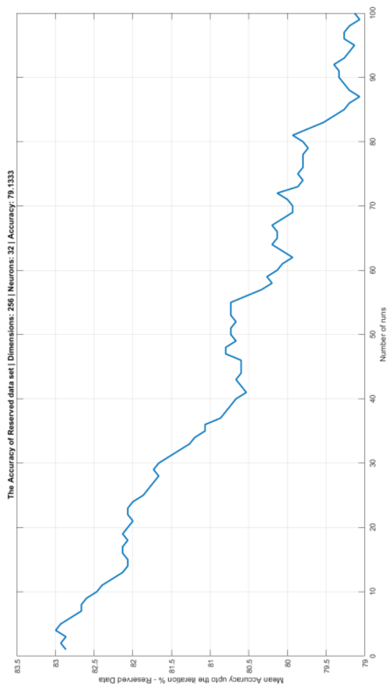
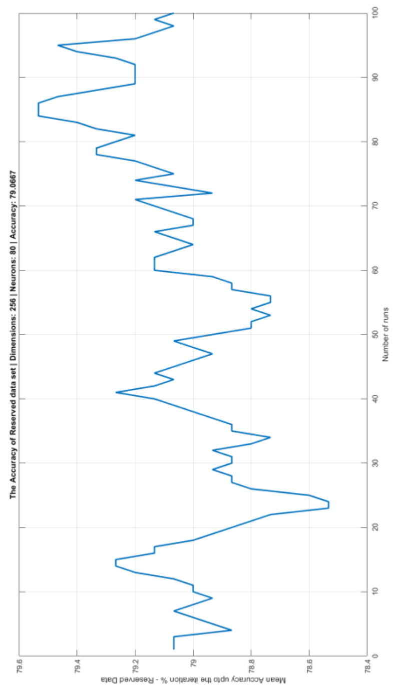
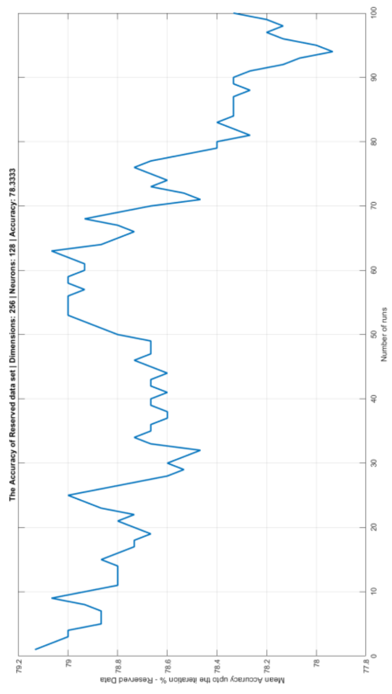
The variance retained in each dimension is as follows,

Table 1: The variance retained when the dimensionality is reduced to 48,64,128 and 256

Dimensions reduced to	48	64	128	256
Variance Retained	94.12%	96.48%	100%	100%

Table 2: The average accuracy vs. No of model runs for different dimensions and neurons

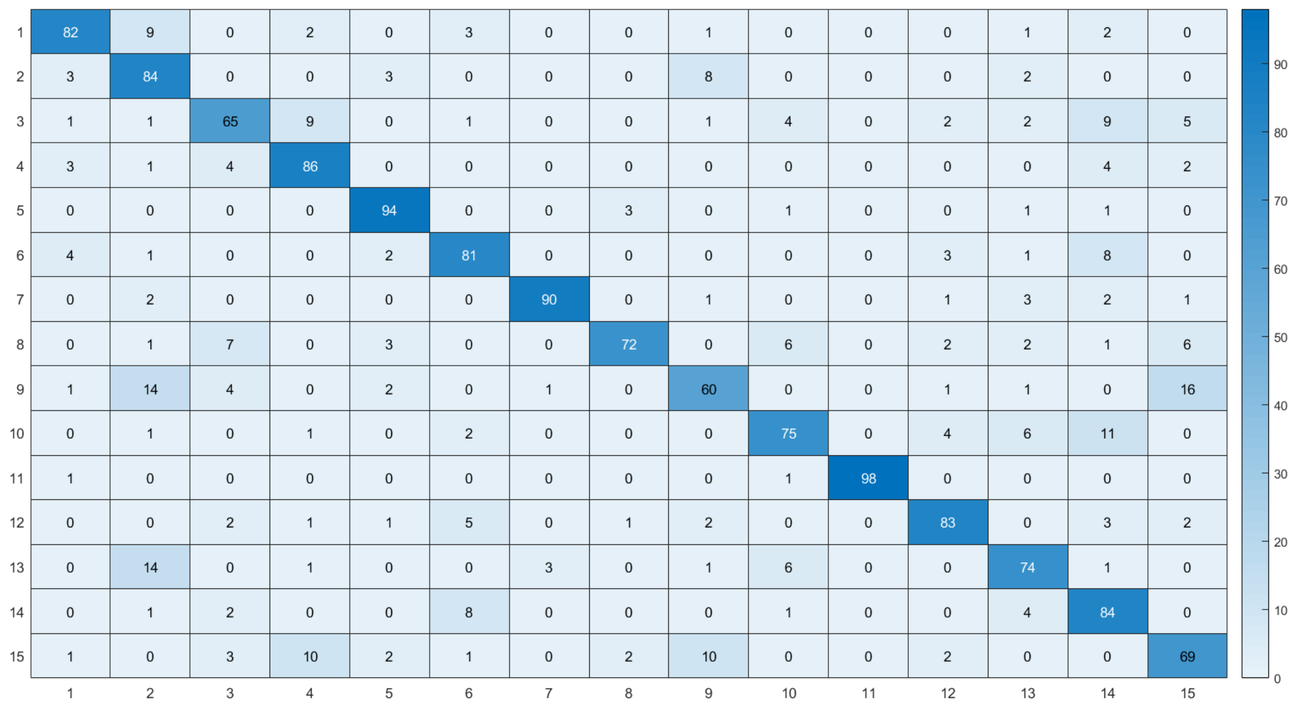
D/N	32	80	128
48			
64			

128			
256			

As seen from the above plots, the test data accuracy increases as the number of dimensions reduces. This is because the model gets over-fit as the number of dimensions increases (as described in step 3). But we cannot reduce the number of dimensions arbitrarily because when reducing the dimensions variance retained gets dropped. We need variance retained to be more than 90% (as described in step 4).

The number of neurons in the hidden layer does not have a significant impact on the test data accuracy.

Now let’s look at how the reserved face set in step 2 performed under the best model out of the previous models. The best model out of the previously analyzed models is the model with 48 dimensions and 80 neurons in the hidden layer. The confusion matrix for the 15 images reserved for later is as follows. The results are from 100 model runs.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	82	9	0	2	0	3	0	0	1	0	0	0	1	2	0
2	3	84	0	0	3	0	0	0	8	0	0	0	2	0	0
3	1	1	65	9	0	1	0	0	1	4	0	2	2	9	5
4	3	1	4	86	0	0	0	0	0	0	0	0	0	4	2
5	0	0	0	0	94	0	0	3	0	1	0	0	1	1	0
6	4	1	0	0	2	81	0	0	0	0	0	3	1	8	0
7	0	2	0	0	0	0	90	0	1	0	0	1	3	2	1
8	0	1	7	0	3	0	0	72	0	6	0	2	2	1	6
9	1	14	4	0	2	0	1	0	60	0	0	1	1	0	16
10	0	1	0	1	0	2	0	0	0	75	0	4	6	11	0
11	1	0	0	0	0	0	0	0	0	1	98	0	0	0	0
12	0	0	2	1	1	5	0	1	2	0	0	83	0	3	2
13	0	14	0	1	0	0	3	0	1	6	0	0	74	1	0
14	0	1	2	0	0	8	0	0	0	1	0	0	4	84	0
15	1	0	3	10	2	1	0	2	10	0	0	2	0	0	69

0
10
20
30
40
50
60
70
80
90

Figure 14: Confusion matrix for the 15 faces

Conclusions

- Principal Component Analysis (PCA) can be used as a successful method to train face recognition algorithms.
- Adding artificial data when the number of training examples are less, proved to increase the accuracy and give better results.
- The accuracy increased with fewer numbers of dimensions. But the number of dimensions can’t be decreased arbitrary because otherwise the information present in the image may vanish.
- When the number of dimensions is 48 and the number of neurons is 80, the algorithm gave an average accuracy of 86.06% for 100 runs.
- Increasing the number of hidden layers reduced the accuracy.

Future Works

- Explore better ways to add artificial data.
- Run the algorithm for other data sets.

MATLAB Code

1. main.m

```
%% Initialize
clc; close all; clear all;

%Load data file
load Yale_32x32.mat;
Dim = [48];      %Number of Dimensions after dimensionality reduction [Can be input as
an array of values]
Neurons = [80];       %Number of neurons in the hidden layer [Can be input as an array
of values]
bgt = 12;            %Brightness - For generating artificial data
sf = 1.2;            %Scale factor - For generating artificial data
Numitr = 100;       %Number of times to run the NN - Set to 1 to run only once
ltrmap = zeros(15,15);    %Initialize confusion matrix

 for ind1=1:length(Dim)
    D = Dim(ind1);
    for ind2=1:length(Neurons)
        N = Neurons(ind2);

if Numitr ~=1        %Generate the accuracy vs Numitr plot
    figure
end
meanaccts=0;         %For plotting
meanaccrs=0;         %For plotting
for ind=1:Numitr             %For determining the accuracy of NN
%% Setup NN and run it

% Shuffle faces of each person
for shuf=1:11:length(gnd)
    temp1 = fea(shuf:shuf+10,:);    temp2 = gnd(shuf:shuf+10);
    idx = randperm(11);
    temp1 = temp1(idx,:);           temp2 = temp2(idx);
    fea(shuf:shuf+10,:) = temp1;    gnd(shuf:shuf+10) = temp2;
end

%Reserve images for later use [Note that this data isn't fed to the neural network at
all]
ltr = 15;                                        %One face of each 15 different faces
is reserved
ltrX = fea(1,:);    ltrY = gnd(1);
fea1 = fea(2:11,:);    gnd1 = gnd(2:11);
for later=12:11:165
ltrX = [ltrX; fea(later,:)]; ltrY = [ltrY; gnd(later)];
fea1 = [fea1; fea(later+1:later+10,:)]; gnd1 = [gnd1; gnd(later+1:later+10)];
end
ltrX = ltrX';

fea = fea1; gnd = gnd1;             %Update fea and gnd

%Generate Artificial Data from the set of 165 images
bright = fea+bgt;                   %Generate data by increasing brightness
dark = fea-bgt;                     %Generate data by reducing brightness
scaled = ((fea-128).*sf)+128;       %Generate data by scaling brightness to both sides
bright2 = fea+20;                   %Generate data by increasing brightness
dark2 = fea-20;                     %Generate data by reducing brightness
scaled2 = ((fea-128).*2)+128;       %Generate data by scaling brightness to both sides
bright3 = fea+30;                   %Generate data by increasing brightness
dark3 = fea-30;                     %Generate data by reducing brightness
scaled3 = ((fea-128).*1.8)+128;     %Generate data by scaling brightness to both sides
bright4 = fea+40;                   %Generate data by increasing brightness
dark4 = fea-40;                     %Generate data by reducing brightness
scaled4 = ((fea-128).*2.2)+128;     %Generate data by scaling brightness to both sides
```

```
features = [fea; bright; dark; scaled; bright2; dark2; scaled2; bright3; dark3;
scaled3; bright4; dark4; scaled4];     %Concatenate all the data
ground_truth = [gnd; gnd; gnd; gnd; gnd; gnd; gnd; gnd; gnd; gnd; gnd; gnd; gnd];
%Corresponding ground truth values


%Train the neural network
    %Outputs - Retained variance, heat map of reduced dimension vs number of
    %neurons, trained neural network, Eigen vectors upto the reduced
    %dimensions, heatmap for test data

    %Inputs - Features, Ground truth, Dimensions to be reduced to, Number
    %of neurons in the hidden layer

[Var,DimvN,net,U_red,testmap,Z,~] = trainNN(features,ground_truth,D,N);
fprintf('\nVariance Retained: %f%% \n',Var*100);
fprintf('Test Data Accuracy: %f%% \n\n',DimvN*100);

variance(ind1,ind2) = Var; %Variance Matrix
%% Results

%Feed the original 165 images back to the NN and check the accuracy
load Yale_32x32.mat;    %Load fresh data again

%Preprocess data to feed into the NN
I165 = fea';
mu = mean(I165');
I165 = I165-mu';        %Mean Normalization
I165 = U_red'*I165;     %Dimensionality reduction

y=net(I165);            %Get predicted output
map1 = zeros(15,15);    %Initialize a heatmap
c=0;

Y = zeros(15,length(gnd)); %Make gnd into a boolean vector
for k=1:length(gnd)
    Y(gnd(k),k) = 1;
end

for j=1:length(gnd)
    [d,c] = max(Y(:,j));        %Actual face
    [p,q] = max(y(:,j));        %Predicted face
    map1(c,q)=map1(c,q)+1;      %Save to heatmap

end

tp=sum(diag(map1));                     %Total True positives
accuracy = (tp/length(gnd))*100;        %Accuracy of 165 Image data set
fprintf('Predicted images out of 165 images: %d | Accuracy: %f%% \n',tp,accuracy);

%Feed the reserved data into the NN and check the accuracy
mu = mean(ltrX');
ltrXmu = ltrX-mu';        %Mean normalization
ltrXred = U_red'*ltrXmu;     %Dimensionality reduction

y=net(ltrXred);            %Get predicted output
map2 = zeros(15,15);     %Initialize heatmap

c=0;
Y = zeros(15,ltr); %Make gnd into a boolean vector
for k=1:ltr
    Y(ltrY(k),k) = 1;
end

for j=1:length(ltrY)
```

```
    [d,c] = max(Y(:,j));     %Actual face
    [p,q] = max(y(:,j));     %Predicted face
    map2(c,q)=map2(c,q)+1;     %Save to heatmap

end
ltrmap = ltrmap+map2;          %Update heatmap
tp=sum(diag(map2));                    %Total True positives
accuracy = (tp/length(ltrY))*100;    %Accuracy of reserved data set
fprintf('Predicted images out of %d images: %d | Accuracy: %f%% \n',ltr,tp,accuracy);

% figure;
% heatmap(testmap);             %Generate heatmap for test data

%% For Determining the accuracy of the NN

if Numitr~=1
% accts(ind) = DimvN*100;           %Accuracy array - Test data
% meanaccts(ind) = mean(accts);     %Mean accuracy upto now - Test data
% subplot(2,1,1)
% plot(1:ind,meanaccts,'LineWidth',2)
% xlabel('Number of Iterations'); ylabel('Mean Accuracy upto the iteration % - Test
Data');
% title("The Accuracy of Test data set | Dimensions: "+D+" | Neurons: "+N+" |
Accuracy: "+meanaccts(ind));
% grid on;
accrs(ind) = accuracy;            %Accuracy array - Test data
meanaccrs(ind) = mean(accrs);     %Mean accuracy upto now - Test data
% subplot(2,1,2)
plot(1:ind,meanaccrs,'LineWidth',2)
xlabel('Number of runs'); ylabel('Mean Accuracy upto the iteration % - Reserved
Data');
title("The Accuracy of Reserved data set | Dimensions: "+D+" | Neurons: "+N+" |
Accuracy: "+meanaccrs(ind));
grid on;
end
end
end
 end

 %Plot confusion matrix
 figure; heatmap(ltrmap);
```

2. trainNN.m

```
function [Var,DimvN,net,U_red,map,Z,mu] = trainNN3(fea,gnd,Dim,Neurons)

[m,n] = size(fea);  %Number of data and featuress
f = 15;             %Number of distinct faces
Var = zeros(1,length(Dim));                 %Initialize variance percentage array
DimvN = zeros(length(Dim),length(Neurons)); %Initialize True positives matrix
com = 0;                                    %For tracking progress
tot = length(Dim)*length(Neurons);          %For tracking progress

for idx1=1:length(Dim)

k = Dim(idx1);      %Dimension to be reduced to
X = fea';           %Features x Examples matrix
gndY = gnd;         %Grounth truth

%Divide data into Test, Train and Validation sets
tr_data = X(:,3:10);   tr_y = gndY(3:10);
ts_data = X(:,1);      ts_y = gndY(1);
cv_data = X(:,2);      cv_y = gndY(2);
for divide=11:10:m
tr_data = [tr_data X(:,divide+2:divide+9)]; tr_y = [tr_y; gndY(divide+2:divide+9)];
ts_data = [ts_data X(:,divide)]; ts_y = [ts_y; gndY(divide)];
cv_data = [cv_data X(:,divide+1)]; cv_y = [cv_y; gndY(divide+1)];
end

%Randomly shuffle each data set
idx_tr = randperm(size(tr_data,2));
tr_data = tr_data(:,idx_tr); tr_y=tr_y(idx_tr);
idx_ts = randperm(size(ts_data,2));
ts_data = ts_data(:,idx_ts); ts_y=ts_y(idx_ts);
idx_cv = randperm(size(cv_data,2));
cv_data = cv_data(:,idx_cv); cv_y=cv_y(idx_cv);


%Concatenate all data sets together
X = [tr_data ts_data cv_data]; gndY = [tr_y; ts_y; cv_y];

%Make gnd into a boolean vector
Y = zeros(f,m);
for i=1:m
    Y(gndY(i),i) = 1;
end

%Identify the Train, Test, Validation indices
trn=1:length(tr_y);                                                     %Train
data
ts=length(tr_y)+1:length(tr_y)+length(ts_y);                            %Test
data
cv=length(tr_y)+length(ts_y)+1:length(tr_y)+length(ts_y)+length(cv_y);
%Validation data


%% Principle Component Analysis to reduce dimension from 1024 to k

%Mean normalization (Substract the mean face from the set of faces)
mu = mean(X');
X = X-mu';

m1=max(trn);                         %Number of training data (PCA IS DONE ONLY FOR
TRAINING DATA)
sigma = (1/m1)*X(:,trn)*X(:,trn)';    %Covariance matrix of data
[U,S,~] = svd(sigma);                %Single Value Decomposition
U_red = U(:,1:k);                    %Dimensionality reduction
Z = U_red'*X;                        %Dimension reduced data
```

```
%Check the percentage of the variance retained
s_k=0;
for p=1:k
    s_k=s_k+S(p,p);                   %Variance of Dimension reduced data
end
s_n=0;
s_n = sum(diag(S));                   %Variance of data before reducing the dimension

Var(idx1) = s_k/s_n;                  %Percentage of the variance retained

for idx2=1:length(Neurons)

neuron = Neurons(idx2);               %Number of Neurons in the hidden layer

%% Neural Network
net = patternnet([neuron neuron]);      %Initialize the neural net
net.trainFcn = 'trainscg';      %Define training fcn
net.divideFcn = 'divideind';    %Define how the data set is divided
net.divideParam.trainInd = trn; %Training data
net.divideParam.testInd = ts;   %Testing data
net.divideParam.valInd = cv;    %Validation data
[net,tr] = train(net,Z,Y);           %Train nnet
com = com+1;                    %For tracking progress
fprintf('Dimension: %d | Neurons: %d | Training... | %d/%d
Completed!\n',k,neuron,com,tot);


%% Calculate the accuracy from test data

y=net(Z(:,ts));                %Feed only the test data back the trained NNet
map = zeros(f,f);              %Initialize heat map
c=0; Y_ts=Y(:,ts);             %Get test data ground truth
for i=1:length(ts)
    [d,c] = max(Y_ts(:,i));      %Actual face
    [p,q] = max(y(:,i));       %Predicted face
    map(c,q)=map(c,q)+1;       %Save to the heat map

end
tp=sum(diag(map));                     %Number of True positives are in the diagonal of
the heat map
DimvN(idx1,idx2) = tp/length(ts);      %Update True positives matrix
% heatmap(map);

%% Generate regression plot
out = vec2ind(net(Z));
actual = vec2ind(Y);


trOut = out(tr.trainInd);
vOut = out(tr.valInd);
tsOut = out(tr.testInd);
trTarg = actual(tr.trainInd);
vTarg = actual(tr.valInd);
tsTarg = actual(tr.testInd);
%plotregression(trTarg,trOut,'Train',vTarg,vOut,'Validation',tsTarg,tsOut,'Testing')


end
end
end
```

References

- Curse of Dimensionality - https://towardsdatascience.com/the-curse-of-dimensionality-50dc6e49aa1e?gi=e318d5ee72dc