

Machine Learning Course Project Report

Indian Institute of Space Science and Technology



Parthasarathi Samanta SC17B106
Kiran, L. SC17B150

June 1, 2020

Abstract

Human face is one of the easiest ways to distinguish and identify individuals. Face recognition is an identification system that identifies a person with his/her physical traits. Human face recognition procedure consists of two steps - face detection and recognition. Face detection and recognition is a much researched topic among community dealing with the branch of Pattern Recognition and Machine Learning. Auto-tagging feature of Facebook is a popular example of application of Machine learning algorithms for facial identification. There are two kinds of methods that are currently popular in developed face recognition namely, Eigenface method and Fisherface method. Facial image recognition through Eigenface method is based on the reduction of face-space dimension using Principal Component Analysis (PCA) for facial features. The main purpose of the use of PCA on face recognition using Eigen faces was formed (face space) by finding the eigenvectors corresponding to the largest eigenvalue of the face image. The software requirements for this project is Jupyter Notebook.

Keywords: Face detection, Face recognition, Eigenface, Image Processing, Principal Component Analysis, Python, Open-CV, PyTorch, kNN, SVM, Neural Network, Ensemble Classifier, Biometric Attendance System.

Extension: There are vast number of applications from this face detection project. This project can be extended to build a system to replace finger print based biometric systems, to improve the search methods in matching a persons's frontal face photo with existing database of criminals, ATM user security, and many more. Further, the process of feature extraction can be included in the neural network by building deep convolutional neural networks to achieve the same.

Acknowledgements

We take this opportunity to express our profound gratitude and deep regards to our guide Dr. Deepak Mishra, Associate Professor & Head, Department of Electronics and Communications (Avionics), for his exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by him, time to time shall carry us a long way in the journey of life on which we are about to embark.

We thank our beloved parents, siblings and dear friends for their continued support and encouragement for the successful completion of our project.

Contents

1	Introduction	5
1.1	Principal Component Analysis	5
1.1.1	Singular Value Decomposition	7
1.2	Problem Statement	8
1.3	Objective	8
2	Literature Survey	9
2.1	Survey on Real Life Requirements of Face recognition systems	9
2.2	Literature Review	10
3	Requirement Specification	12
4	Design	13
4.1	Classifiers	13
4.1.1	k Nearest Neighbours	13
4.1.2	Support Vector Machine	15
4.1.3	Feed Forward Neural Network	19
4.2	Data	22
4.3	Training Pipeline	23
4.4	Face Detection and Recognition Pipeline	26
4.5	Evaluation Metrics	27
4.6	Block Diagram	30
5	Implementation	31
5.1	Ensemble Classifier	38
5.2	Face Alignment	39
5.3	Saving and Loading Models	41
5.4	Washington Dataset	41

5.5	Comparison of Classifiers Performance	45
5.6	Performance of Generalized Neural Network	46
5.7	LFW Dataset (Subset)	46
5.8	Comparison of Classifiers Performance	49
5.9	Fetch Olivetti Faces Dataset	51
5.10	Comparison of Classifiers Performance	55
6	Attendance System	56
6.1	System Description	56
6.1.1	Problem of False Positives	60
7	Conclusion	65
8	Future Enhancement	66
A	Reconstructed Face images of Washington Dataset	67
A	Reconstructed Face images of Fetch Olivetti Faces Dataset	69
A	Reconstructed Face images of LFW Data-Sub-Set	71
A	Pseudo Code	72

Chapter 1

Introduction

1.1 Principal Component Analysis

Basic Idea

PCA[1] is about projecting (Dot Product) data points onto a line, i.e. data points are projected into a lower dimension. PCA is a projection such that a variance in that projection is as high as possible. PCA is just finding the projection such that most of the variance of the data is accounted for.

The first thing in a PCA is a shift of the data onto a new coordinate system by calculating the mean for every dimension in the data, then subtracting every observation in that dimension by the mean.

\tilde{X} is the new X, while μ (mu) is the mean,

$$\tilde{X} = X - \mu.$$

The origin of our coordinate system would now have been shifted to the center of our data, by subtracting the mean: $\sum = \frac{1}{n-1} \sum_{i=1}^n x_i$

Finding Principal Components

The objective would be to find the direction in which we get the most variance (imagine a line through origin and the data points). To find the most variance we want the directions where there are most data points covered (e.g. by drawing a line in that direction).

What we really want is the line, after having projected all data points, where we have maximized the sum of the distance from all points to origin. That is what most variance is. So for each data point, we would have a

distance d and we would have n data points, so we would sum all of the squared distances and divide them by $n - 1$:

$$\frac{d_1^2 + d_2^2 + \dots + d_n^2}{n-1} = \text{variance}$$

Finding the Eigenvectors and Eigenvalues

How to get the line that accounts for most variance for our data. This is what we call an eigenvalue problem and here is the formula: $A\vec{v} = \lambda\vec{v}$
 A is a transformation, a squared matrix, \vec{v} is the eigenvector, and λ is the eigenvalue. Most of the time, the literature will represent or derive matrix A as $\tilde{x}^T\tilde{x}$.

The procedure for finding the line is

- Finding the eigenvalues.
- Finding the eigenvectors from the eigenvalues.

For first part, to find the eigenvalues, we have $\det(A - \lambda I) = 0$

For the second part - to find the eigenvectors, we would want to change up the formula. So we move everything from the right side of the formula to the left side and set it equal to 0: $A\vec{v} = \lambda\vec{v} \iff (A - \lambda I)\vec{v} = 0$

we would find a resulting eigenvector for each eigenvalue. To solve this, we can use a Gaussian process (e.g. Gaussian Elimination).

The eigenvector corresponding to the highest eigenvalue is the principal component. We can have more than one principal component if we use eigenvalues which are significant as compared to the highest eigenvalue.[2]

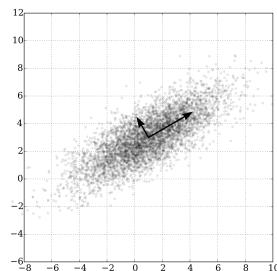


Figure 1.1: Principal Components

1.1.1 Singular Value Decomposition

In linear algebra, the singular value decomposition (SVD) is a factorization of a real or complex matrix that generalizes the eigendecomposition of a square normal matrix to any $m \times n$ matrix via an extension of the polar decomposition.

The singular value decomposition (SVD) provides another way to factorize a matrix, into singular vectors and singular values. The SVD allows us to discover some of the same kind of information as the eigendecomposition. However, the SVD is more generally applicable.

Specifically, the singular value decomposition of an $m \times n$ real or complex matrix \mathbf{M} is a factorization of the form $\mathbf{U}\Sigma\mathbf{V}^*$, where \mathbf{U} is an $m \times m$ real or complex unitary matrix, Σ is an $m \times n$ rectangular diagonal matrix with non-negative real numbers on the diagonal, and \mathbf{V} is an $n \times n$ real or complex unitary matrix. If \mathbf{M} is real, \mathbf{U} and $\mathbf{V}^T = \mathbf{V}^*$ are real orthonormal matrices.

The diagonal entries $\sigma_i = \Sigma_{ii}$ of Σ are known as the singular values of \mathbf{M} . The number of non-zero singular values is equal to the rank of \mathbf{M} . The columns of \mathbf{U} and the columns of \mathbf{V} are called the left-singular vectors and right-singular vectors of \mathbf{M} , respectively.

The singular value decomposition is very general in the sense that it can be applied to any $m \times n$ matrix, whereas eigenvalue decomposition can only be applied to diagonalizable matrices.

$$\begin{pmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & \\ \vdots & \vdots & \ddots & \\ x_{m1} & & & x_{mn} \end{pmatrix}_{m \times n} \approx \begin{pmatrix} u_{11} & \dots & u_{1r} \\ \vdots & \ddots & \\ u_{m1} & & u_{mr} \end{pmatrix}_{m \times r} \begin{pmatrix} s_{11} & 0 & \dots \\ 0 & \ddots & \\ \vdots & & s_{rr} \end{pmatrix}_{r \times r} \begin{pmatrix} v_{11} & \dots & v_{1n} \\ \vdots & \ddots & \\ v_{r1} & & v_{rn} \end{pmatrix}_{r \times n}$$

Figure 1.2: Singular Value Decomposition

1.2 Problem Statement

A face recognition system (software) is to be designed. The program should reduce each face image to a vector, then uses Principal Component Analysis (PCA) to find a linear subspace for faces. This space is spanned by just a few vectors, which means each face image can be defined by a small set of coefficients weighting these vectors, thus reducing and considering only the information from large data for further analysis and processing.

1.3 Objective

The primary objective of this project is to design a system which is able to take in an image, locate faces of people in the image and identify each one of them using Eigenface method. The objective can be broken down into two broad sub-objectives as:

1. Face Detection
2. Face Recognition

Machine Learning algorithms and classifiers are to be used for identifying (classifying) the detected faces in given test image. The input test image may contain faces of more than one person (i.e., group photo).

Chapter 2

Literature Survey

2.1 Survey on Real Life Requirements of Face recognition systems

The literature survey was performed to get an insight to the real life situations where face detection and recognition systems can reduce effort and time.

The most crucial requirement of face recognition system is in **criminology**. The database of photographs of the past accused and convicts is huge. The management of the database itself is a tiresome task. This data contains much of redundant information, for example the pixels corresponding to the background of the person, etc. Often when new cases arise, the photographs of the suspects is matched with the pre-developed database. This task could take long time (increase in time with database size) without use of optimized search algorithms. These two problems of storage size of database and speed of identification can be overcome by building machine learning systems and optimizing over various search optimization algorithms and GPU execution.

Next prominent use of the face recognition system is in social media such as Facebook, Twitter, etc., Facebook, has already implemented and optimized its face recognition system - evident from the accuracy of auto-tagging feature.

Another upcoming technology is the phone face unlock feature which requires use of face recognition systems.

Also, with the increase in the number of CCTV cameras throughout the

globe, the effort and time required for finding missing people (especially, children) can be reduced drastically by incorporating a face recognition system into CCTV cameras in public places with direct data relay to the appropriate police stations. Such applications require real time face identification to identify the people in the video frames.

The face recognition system find its need in ATMs to validate identity of people transacting in the ATMs.

One of the requirements which is much advocated by higher authorities but strongly opposed by students is face recognition based automatic attendance system.

Similarly, many such real life requirements do exist, where face recognition systems can play a crucial role.

2.2 Literature Review

Various students, research scholars and industrialists working in **Pattern Recognition and Machine Learning** have developed face recognition systems for various applications. The methods used by each vary and so does the accuracy and other performance metrics.

A team of 4 students from Jawaharlal Nehru Technological University Kakinada, designed a “Face Recognition System With Face Detection” under the guidance of Ms. SK. AYESHA, M.Tech, Assistant Professor of E.C.E dept during the academic year 2013-17. The implementation was done in MATLAB software. The face detection was achieved through face localization (using features such as skin color detection), followed by mouth, nose and eyes and verifying eyes-mouth triangle. Fully automated face detection of frontal view faces was implemented using a deformable template algorithm relying on image invariants of human faces. Face recognition was based on Principal Component Analysis. 30 test subjects were used for the analysis.

A Face recognition system was built by Yang Li and Sangwhan Cha, PhD. Assistant Professor of Computer Science,[3] in which ORL dataset of 400 frontal face images (of 40 subjects, with 10 faces per person) was used. A Siamese network was designed for measuring similarity between two subject's faces. The model training and complete execution was done with torch tensors on Nvidia GPU. The system so made produces appreciable results only when number of subjects is less and number of faces per person

is large. No block was built for face detection.

A famous project on face detection and recognition is : **Open Face**[4]. Implemented in Python and Torch, the workflow of the system includes following steps: Detect faces with a pre-trained models from dlib or OpenCV, transform the face for the neural network using OpenCV's affine transformation, crop out the face from the image, use a deep neural network for training and use the trained model classifier for clustering, similarity detection and classification.

The above list of projects is non-exhaustive, as many research papers of new architectures for face recognition systems with deep learning, convolutional neural networks, etc., are coming up every now and then with a large community or group of teams putting in effort to build real time applicable face recognition systems.

Chapter 3

Requirement Specification

The following problem scope for this project was arrived at after reviewing the literature on face detection and face recognition, and determining possible real-world situations where such systems would be of use. The following system(s) requirements were identified:

1. A system to detect frontal view faces in static images
2. All implemented systems must display a high degree of lighting invariance.
3. All systems must posses near real-time performance.
4. A fully automated face detection must be supported
5. Frontal view face recognition realised using only a set of known frontal view face images
6. The face recognition sub-system must display a slight degree of invariance to scaling and rotation errors in the segmented image extracted by the face detection sub-system.

Unfortunately although we may specify constricting conditions to our problem domain, it may not be possible to strictly adhere to these conditions when implementing a system in the real-world.

Chapter 4

Design

The present chapter is dedicated to explain the conceptual overview of the face detection and recognition system developed. The theory and algorithms are described with aid of diagrams and flow charts.

4.1 Classifiers

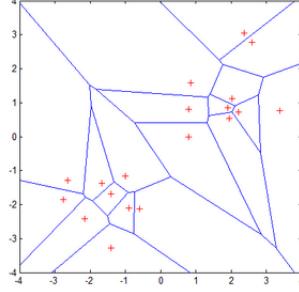
4.1.1 k Nearest Neighbours

The Basic Idea of KNN and Voronoi set

The Basic idea of KNN[5] is based on minimum euclidean distance between a test sample and specified training sample. Let \mathbf{x}_i be input with p features $(x_{i1}, x_{i2}, \dots, x_{ip})$, n be the number of samples ($i = 1, 2, \dots, n$), and p be the total number of features($j=1,2,\dots,p$) . The euclidean distance between \mathbf{x}_i and \mathbf{x}_l ($l = 1, 2, \dots, n$) is given by

$$d(\mathbf{x}_i, \mathbf{x}_l) = \sqrt{(x_{i1} - x_{l1})^2 + (x_{i2} - x_{l2})^2 + \dots + (x_{ip} - x_{lp})^2} .$$

The concept of nearest neighbor is explained using a diagram here. The minimum distance points corresponding to one specific training samples are defined by the boundaries of the Voronoi cell corresponding to that specific training samples.



A Voronoi cell corresponding to one training point is defined by-
 $R_i = \{\mathbf{x} \in \mathbb{R}^p : d(\mathbf{x}, \mathbf{x}_i) \leq d(\mathbf{x}, \mathbf{x}_m), \quad \forall i \neq m\}$, the fundamental properties reflected from these sets are that-

- all possible points within a sample's Voronoi cell are the nearest neighboring points for that sample.
- for any sample, the nearest sample is determined by the closest Voronoi cell edge.

Using the 2nd property, the k-nearest-neighbor classification rule is to assign to a test sample the majority category label of its k nearest training samples. In practice, k is usually chosen to be odd, so as to avoid ties.

Classification decision rule

Classification typically involves partitioning samples into training and testing categories. Let \mathbf{x}_i be a training sample and \mathbf{x} be a test sample, and let ω be the true class of a training sample and $\hat{\omega}$ be the predicted class for a test sample ($\omega, \hat{\omega} = 1, 2, \dots, \Omega$). Here, Ω is the total number of classes. During the training process, we use only the true class ω of each training sample to train the classifier, while during testing we predict the class $\hat{\omega}$ of each test sample. It warrants noting that kNN is a "supervised" classification method in that it uses the class labels of the training data. With 1-nearest neighbor rule, the predicted class of test sample \mathbf{x} is set equal to the true class ω of its nearest neighbor, where \mathbf{m}_i is a nearest neighbor to \mathbf{x} if the distance-

$$d(\mathbf{m}_i, \mathbf{x}) = \min_j \{d(\mathbf{m}_j, \mathbf{x})\}.$$

For k-nearest neighbors, the predicted class of test sample \mathbf{x} is set equal to the most frequent true class among k nearest training samples. This forms the decision rule $D : \mathbf{x} \rightarrow \hat{\omega}$.

Confusion Matrix

The confusion matrix used for tabulating test sample class predictions during testing is denoted as \mathbf{C} and has dimensions $\Omega \times \Omega$.

- During testing, if the predicted class of test sample \mathbf{x} is correct (i.e., $\hat{\omega} = \omega$), then the diagonal element $c_{\omega\omega}$ of the confusion matrix is incremented by 1.
- However, if the predicted class is incorrect (i.e., $\hat{\omega} \neq \omega$), then the off-diagonal element $c_{\omega\hat{\omega}}$ is incremented by 1.

Once all the test samples have been classified, the classification accuracy is based on the ratio of the number of correctly classified samples to the total number of samples classified, given in the form-

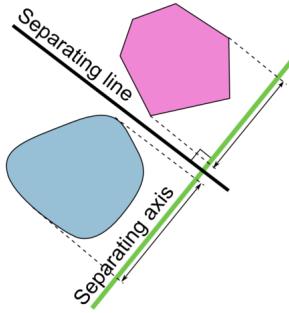
$$Acc = \frac{\sum_{\omega} c_{\omega\omega}}{n_{total}}$$

where $c_{\omega\omega}$ is a diagonal element of \mathbf{C} and n_{total} is the total number of samples classified.

4.1.2 Support Vector Machine

Objective

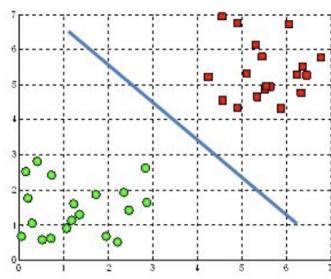
The objective of SVM[6] is to find a hyperplane in an N-dimensional space (N-Number of features) that distinctly classifies the data points. In the context of support-vector machines, the optimally separating hyperplane or maximum-margin hyperplane is a hyperplane which separates two convex hulls of points and is equidistant from the two.



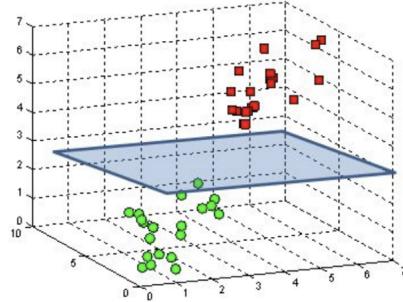
Hyperplane

In an N-dimensional space, a hyperplane is a flat affine subspace of dimension N-1. Visually, in a 2D space, a hyperplane will be a line and in 3D space, it will be a flat plane. In simple terms, hyperplane is a decision boundary that helps classifying data points.

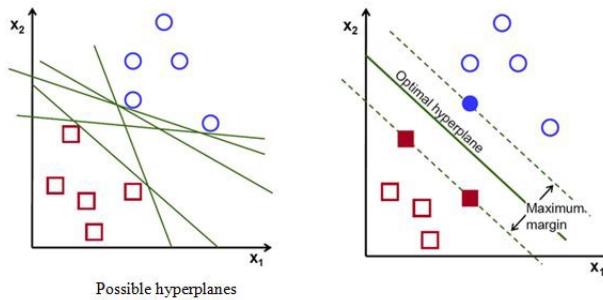
A hyperplane in \mathbb{R}^2 is a line



A hyperplane in \mathbb{R}^3 is a plane

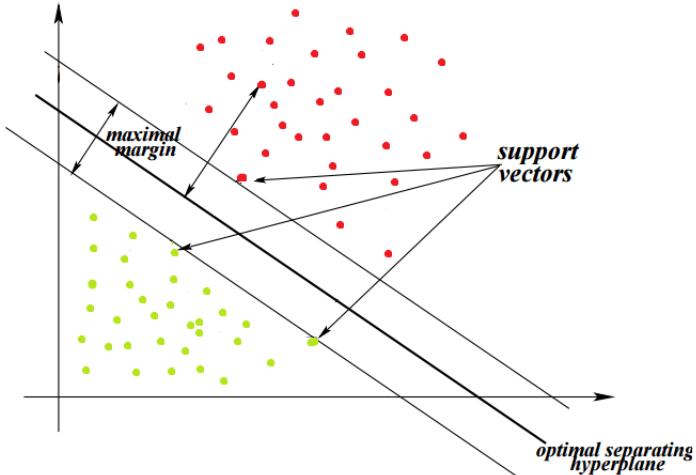


Now, to separate two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin i.e. the maximum distance between data points of both classes and below figure clearly explains this fact.



Support Vectors

Support Vectors are the data points that are on or closest to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors we maximize the margin of the classifier and deleting these support vectors will change the position of the hyperplane. These are actually the points that help us build SVM.



Support Vectors are equidistant from the hyperplane. They are called support vectors because if their position shifts, the hyperplane shifts as well. This means that the hyperplane depends only on the support vectors and not on any other observations.

Non-linear Classification

Concept of Kernel in SVM to classify non-linearly separated data. A kernel is a function which maps a lower-dimensional data into higher dimensional data. There are two ways by which kernel SVM will classify non-linear data.

- Soft Margin
- Kernel Tricks

Soft Margin

In this method, SVM tolerates a few dots to get misclassified and tries to balance the tradeoff between finding the line that maximizes the margin and minimizes misclassification.

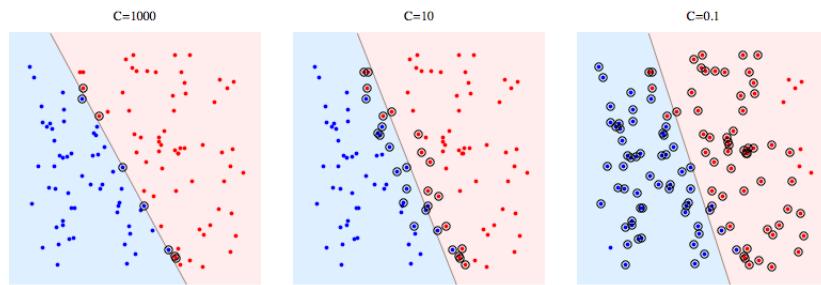
There are two types of misclassifications that can happen:

- The data point is on the wrong side of the decision boundary but on the correct side.
- The data point is on the wrong side of the decision boundary and on the wrong side of the margin.

Degree of Tolerance

How much tolerance we want to set when finding the decision boundary is an important hyper-parameter for the SVM (both linear and nonlinear solutions). In Sklearn, it is represented as the penalty term — ‘C’.

The bigger the C, the more penalty SVM gets when it makes misclassification. Therefore, the narrower the margin is and fewer support vectors the decision boundary will depend on.

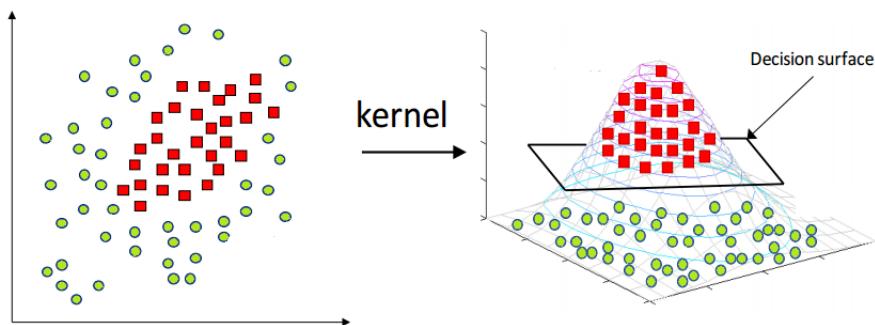


Kernel Trick

The idea is mapping the non-linear separable data from a lower dimension into a higher dimensional space where we can find a hyperplane that can separate the data points.

Kernel Functions are generalized functions that take 2 vectors(of any dimension) as input and output a score(dot product) that denotes how similar the input vectors are. If the dot product is small, vectors are different and if the dot product is large, vectors are more similar.

Pictorial representation of the Kernel Trick-



Different types of kernel Functions are:

- Linear
- Polynomial
- Radial Basis Function
- Sigmoid

The most popular rbf kernel is Gaussian Radial Basis function.
Mathematically:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2)$$

where γ controls the influence of new features on the decision boundary. Higher the value of γ more influence of features on the decision boundary.

Similar to Regularization parameter/penalty term(C) in the soft margin, γ is a hyperparameter that can be tuned when we use kernel trick.

4.1.3 Feed Forward Neural Network

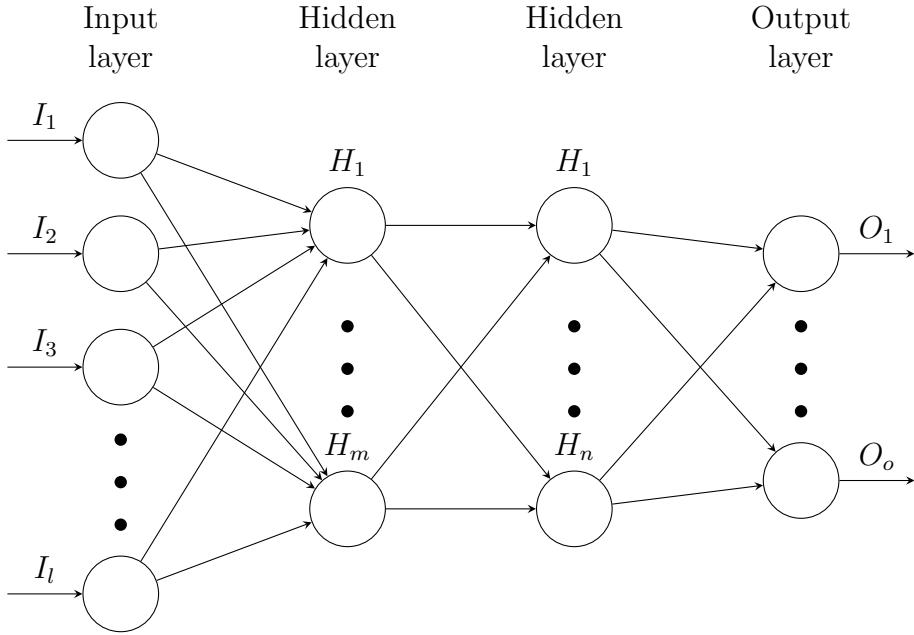
Basic Idea

A neural network[7] simply consists of neurons (also called nodes). These nodes are connected in some way. Then each neuron holds a number, and each connection holds a weight.

These neurons are split between the input, hidden and output layer. In practice, there are many layers and there are no general best number of layers.

Input Layer

The input data is just the dataset, where each observation is run through sequentially from $x = 1, \dots, x = i$. Each neuron has some activation — a value between 0 and 1, where 1 is the maximum activation and 0 is the minimum activation a neuron can have



Input Layer to Hidden Layer

each neuron has an activation a and each neuron that is connected to a new neuron has a weight w . Activations are typically a number within the range of 0 to 1, and the weight is a double, one could multiply activations by weights and get a single neuron in the next layer, from the first weights and activations w_1a_1 all the way to w_na_n :

$$w_1a_1 + w_2a_2 + \dots + w_na_n = \text{new neuron}.$$

The procedure is the same moving forward in the network of neurons, hence the name feed forward neural network, and finally we reach the output layer.

Hidden Layer to Output Layer

The neuron outputs from the final hidden layer is given to the Output layer, which contains neurons among which one activation will be higher than all others corresponding to a particular class, sometimes it can be related to a probability, where each neuron in the output layer contains some activation value(some small positive value for all neurons except the one corresponding to the right class which will have activation number close to 1.)

Activation Function

An activation function, most commonly a sigmoid function, just scales the output to be between 0 and 1 again — so it is a logistic function.

$$\text{sigmoid} = \sigma = \frac{1}{1+e^{-x}} = \text{number between 0 and 1.}$$

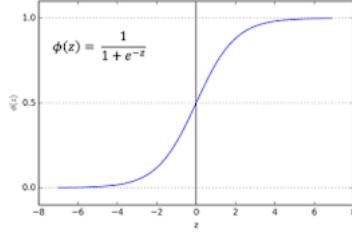


Figure 4.1: Sigmoid Function

We wrap the equation for new neurons with the activation, i.e. multiply summarization of the result of multiplying the weights and activations $\sigma(w_1a_1 + w_2a_2 + \dots + w_na_n) = \text{new neuron.}$

Other commonly used activation functions include: tanh, ReLU, leaky ReLU, etc., ReLU and leaky ReLU were introduced to overcome the issue of vanishing gradients when the activation function saturates, causing no effective learning.

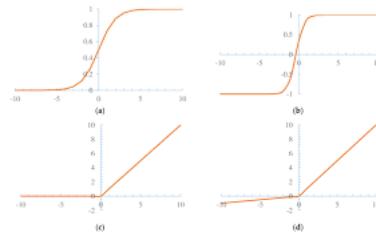


Figure 4.2: Different Activation Functions

Bias

Bias is trying to approximate where the value of the new neuron starts to be meaningful. We just need to add or subtract bias from the multiplication of weights and activations. $\sigma(w_1a_1 + w_2a_2 + \dots + w_na_n + b) = \text{new neuron.}$

Cross Entropy Loss

Of the various loss function used for training a neural network such as MSE, 1-0 loss function, etc., cross entropy loss is most commonly used in practice, along with L2 regularization. Below is the description of the cross entropy loss function:

Cross-entropy loss is often simply referred to as “cross-entropy,” “logarithmic loss,” “logistic loss,” or “log loss” for short.

Each predicted probability is compared to the actual class output value (0 or 1) and a score is calculated that penalizes the probability based on the distance from the expected value. The penalty is logarithmic, offering a small score for small differences (0.1 or 0.2) and enormous score for a large difference (0.9 or 1.0).

Cross-entropy loss is minimized, where smaller values represent a better model than larger values. A model that predicts perfect probabilities has a cross entropy or log loss of 0.0.

Cross-entropy for a binary or two class prediction problem is actually calculated as the average cross entropy across all examples.

Cross-entropy can be calculated for multiple-class classification. The classes have been one hot encoded, meaning that there is a binary feature for each class value and the predictions must have predicted probabilities for each of the classes. The cross-entropy is then summed across each binary feature and averaged across all examples in the dataset.

4.2 Data

In this project, 3 different datasets were used, as briefed below:

1. **Washington Dataset[8]:** The data comprises of totally 24 different people with two images per person - one smiling and one non-smiling. This cropped (but, not aligned) images form the training samples. The test images are the group photos, with three people in a group. All the images were colored images (having R,G,B components).
2. **Fetch Olivetti Faces Dataset:** This data comprised of frontal faces of 40 different people with 10 faces per person. No testing samples are available. Hence, the classifier designed needs to be tested using validation set obtained by splitting the data samples into training an

validation sets. All face images are cropped images, hence, there is no necessity to detect the faces.

3. **Labeled Faces in the Wild Home (LFW) Dataset:** This dataset comprised of 13233 images of 5749 people with two or more images of 1680 people. The complete dataset was not used, due to various constraints. Instead, a sub-set of the dataset, with images of 93 people was used. The number of faces per person was limited to 3 or 4 excluding those with only 1 or 2.

4.3 Training Pipeline

The data was appended (data augmentation) to the training samples by adding Gaussian (white) noise to each image and rotating it by random angle within a finite interval comprising of rotation in both sense (clockwise and anti-clockwise) and generating new images, so as to increase number of samples per person. The training samples are first pre-processed using OpenCV's image processing tool-box. The image processing steps include:

1. Resizing: All the training images were resized to an image of common size. Each training images made available are not of identical dimension.
2. Histogram Equalization: It is a technique to improve contrast in images, which is accomplished by effectively spreading out the most frequent intensity values, i.e. stretching out the intensity range of the image. This method usually increases the global contrast of images when its usable data is represented by close contrast values. This allows for areas of lower local contrast to gain a higher contrast.
3. Face Alignment: The face alignment step is used to ensure that all the faces are aligned such that face is not tilted (i.e., is straight).
4. RGB to Gray scale conversion: All the input images were converted to gray scale images as individual components of the images were all identical to each other, expect for the scale of each image pixel range. As no extra advantage over gray scale image is provided by colored images, all the images (both test and train samples) were converted to gray scale images.

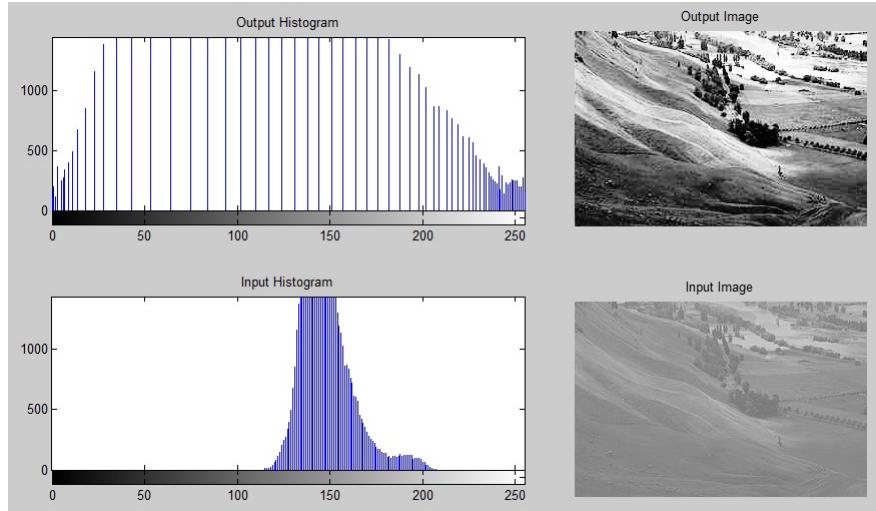


Figure 4.3: Histogram Equalization

Next, is the step relating to Principal Component Analysis (PCA). Before proceeding, all the images were linearized (into linear 1D arrays). Through the method of PCA, the input training images were mapped from a image space of higher dimensions (say, m) to face space of lower dimensions (say, n , (here, $\frac{1}{8}$ of the original number of dimensions)). In the process, the first n eigenfaces were considered for further analysis. Also, a transformation matrix was constructed to map input images from image (pixel) space to face space. Next, for training of the classifier, these face space vector representation of the training images were used.

For classification purpose three different classifiers were tested and finally, an ensemble of classifier was designed for improved performance of the system. The three classifiers used are:

1. k Nearest Neighbour
2. Support Vector Machine
3. (Fully Connected) Neural Network

The theory of each of the above classifiers is described in detail in the section of "Classifiers" of this chapter.

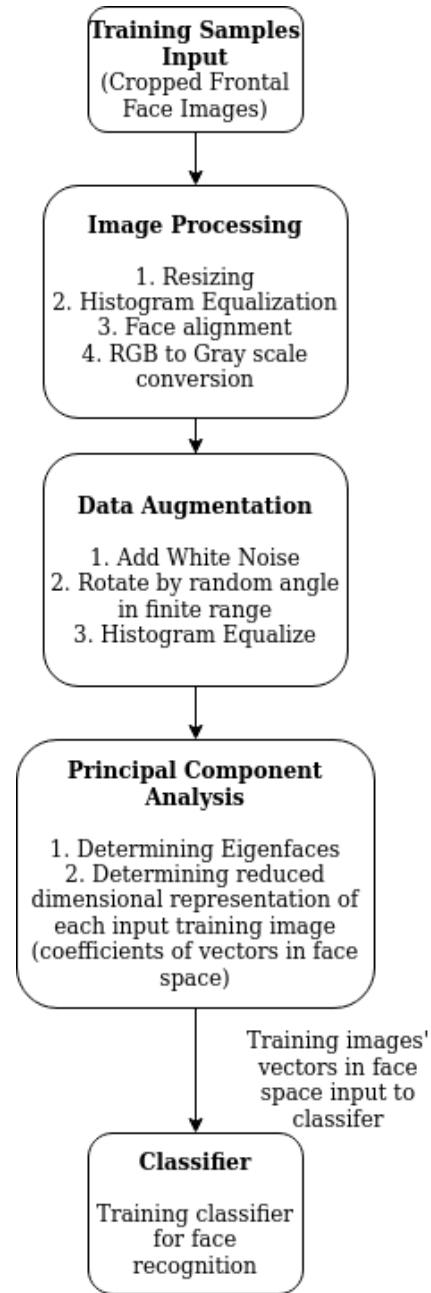


Figure 4.4: Training Pipeline

4.4 Face Detection and Recognition Pipeline

As mentioned previously, the test images, if available were used, else, validation images were obtained from the dataset. The first step in this pipeline is detecting of faces in a group photo. This is achieved using OpenCV's Haar cascade classifiers (based on Viola Jones Algorithm).

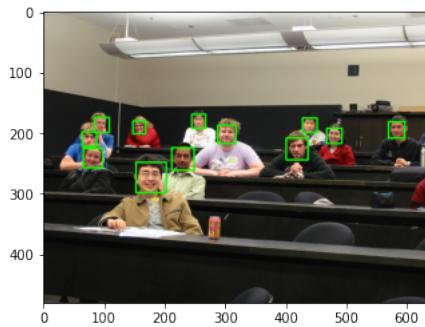


Figure 4.5: Face Detection - on Practice Image

Once the classifier locates the faces of the people in the test image, a rectangle is drawn around the face. Then, the coordinates of these rectangle's corner and width and height of each rectangle are used to crop out the faces from the image.

Next, the names of the person, as identified by the face recognition system, is tagged to the detected faces in the image.

The step after cropping out the faces is basically image processing step. Not all the rectangles drawn by the algorithm are of same dimensions, hence, each is cropped face needs to be resized. The crucial step here is re-cropping the faces cropped out of group photos. The step is crucial because, the accuracy of the classifier critically depends on the degree of resemblance of the cropped test face images with the training set images.

After cropping the images to match the training samples as much as possible, all the images are resized to the same dimensions as pre-processed training samples, followed by histogram equalization and then RGB to Gray scale conversion.

Once, the pre-processing the test images is done, the images are mapped to face space vectors and plugged into trained classifier(s) for face recognition. The predicted outputs are compared with the ground truth and the top-1,5,10 accuracy scores and confusion matrix are determined.

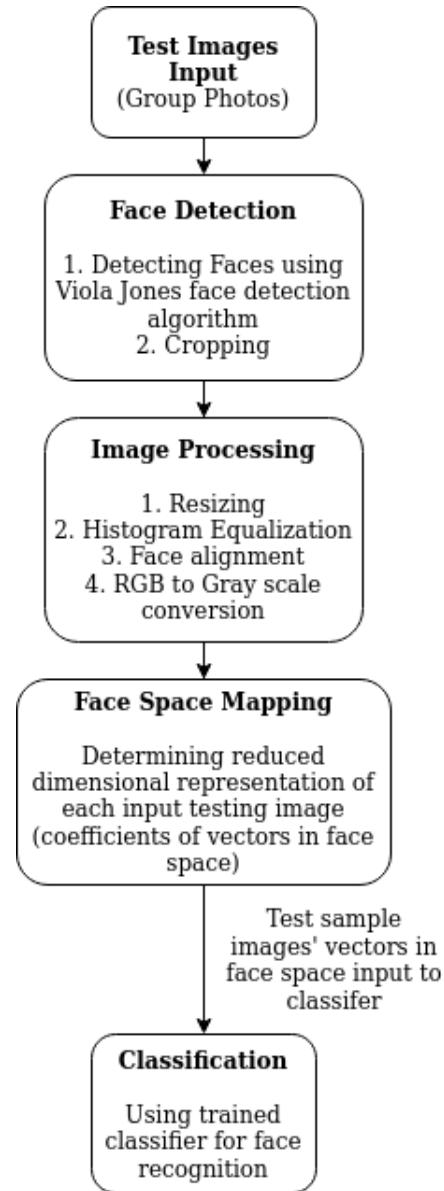


Figure 4.6: Face Detection and Recognition Pipeline

4.5 Evaluation Metrics

Confusion Matrix

A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized

	<i>Class 1 Predicted</i>	<i>Class 2 Predicted</i>
<i>Class 1 Actual</i>	TP	FN
<i>Class 2 Actual</i>	FP	TN

Figure 4.7: Confusion Matrix

with count values and broken down by each class. This is the key to the confusion matrix. The confusion matrix shows the ways in which your classification model is confused when it makes predictions. It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made. Here

- Class1:positive
- Class2:negative

Definition of the Terms:

- Positive (P) : Observation is positive (for example: is an apple).
- Negative (N) : Observation is not positive (for example: is not an apple).
- True Positive (TP) : Observation is positive, and is predicted to be positive.
- False Negative (FN) : Observation is positive, but is predicted negative.
- True Negative (TN) : Observation is negative, and is predicted to be negative.
- False Positive (FP) : Observation is negative, but is predicted positive.

Classification Rate/Accuracy:

Classification Rate or Accuracy is given by the relation:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

However, there are problems with accuracy. It assumes equal costs for both kinds of errors. A 99% accuracy can be excellent, good, mediocre, poor or terrible depending upon the problem.

Recall:

$$Recall = \frac{TP}{TP+FN}$$

Recall can be defined as the ratio of the total number of correctly classified positive examples divide to the total number of positive examples. High Recall indicates the class is correctly recognized (a small number of FN).

Precision:

$$Precision = \frac{TP}{TP+FP}$$

To get the value of precision we divide the total number of correctly classified positive examples by the total number of predicted positive examples. High Precision indicates an example labelled as positive is indeed positive (a small number of FP).

- **High recall, low precision** This means that most of the positive examples are correctly recognized (low FN) but there are a lot of false positives
- **Low recall, high precision:** This shows that we miss a lot of positive examples (high FN) but those we predict as positive are indeed positive (low FP).

F-measure:

Since we have two measures (Precision and Recall) it helps to have a measurement that represents both of them. We calculate an F-measure which uses Harmonic Mean in place of Arithmetic Mean as it punishes the extreme values more. The F-Measure will always be nearer to the smaller value of Precision or Recall. $F - measure = \frac{2 \times Recall \times Precision}{Recall + Precision}$

4.6 Block Diagram

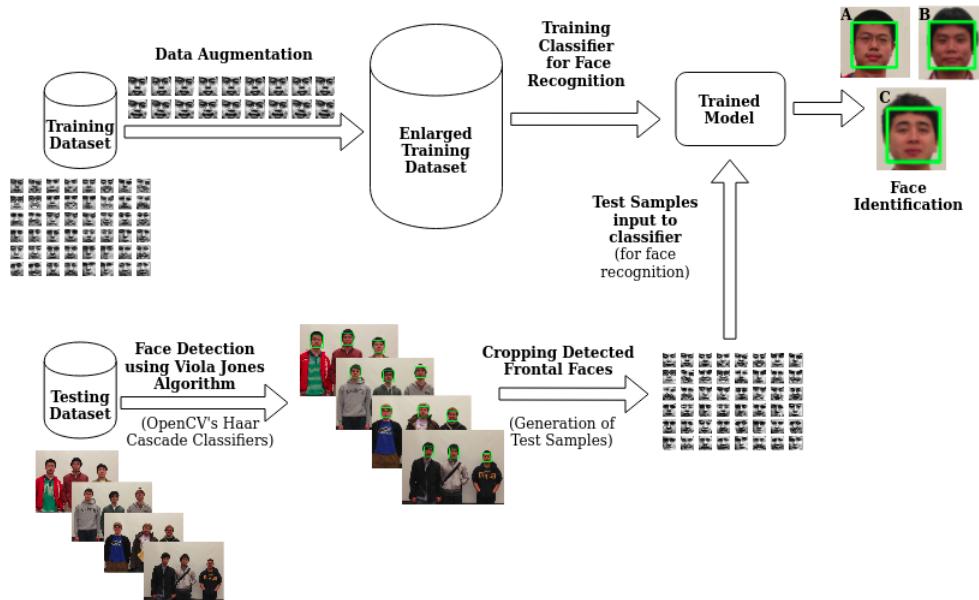


Figure 4.8: Block Diagram of Face Recognition System Designed

Chapter 5

Implementation

The system design described in above chapter was implemented in Python programming language (in Jupyter notebook on Google Colab platform - with GPU hardware accelerator). Effort was made to hand-code most of the important functions in the pipeline and minimize the usage of in-build functions for the same. Next, PCA was performed using user defined functions as shown in the flow chart below.

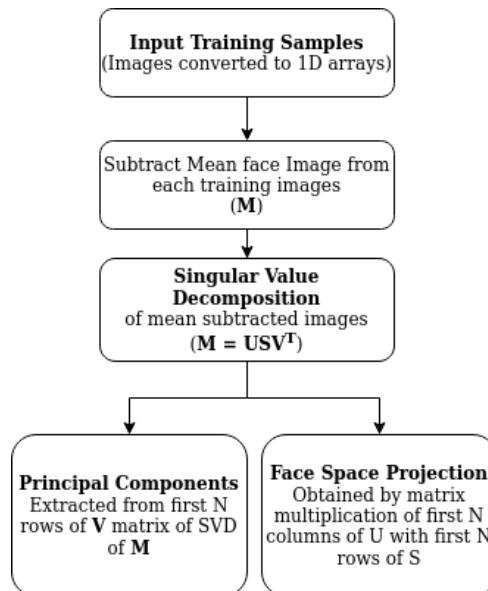


Figure 5.1: PCA Pipeline

The face detection was achieved using Haar cascade classifier for face detection - *haarcascade_frontalface_alt.xml* - with the parameters of scaleFactor and minNeighbors variable with the dataset used.

The *rectangle* function is used for visualizing face detection process. The cropping of test face images to best match the training images of frontal face, a common crop dimensions and crop location was used (as mentioned above). This is a trade off between degree of resemblance and manual effort. It is possible to crop out each image manually to best resemble the training samples, yet not a good practice in general as real life situations contain images of numerous (1000s of) people. No metric was used to test the best crop dimensions, instead was completed by simple visual aid.

Post face detection, the test images' pre-processing was done using following OpenCV's module functions:

Function	Usage description
<i>resize</i>	Used to resize all the images (both train and cropped test images) to a common higher dimensions along with interpolation
<i>hist_equalizer</i>	Histogram Equalization
<i>crop</i>	Used to crop detected face images to match training samples
<i>rectangle</i>	Used to draw a rectangle around the detected faces in the group photo
<i>putText</i>	Used to tag the face images with the names of the person as identified by the system.

Next is the step of training the classifier(s). The first classification algorithm used is k Nearest Neighbour (kNN). A significant decline in the accuracy was obtained with decrease in the value of k.

Next classifier used was Support Vector Machine (SVM). For this classifier, python libraries and inbuilt functions were used. Attempt was made to hand-code the function for SVM. Due to lack of knowledge in optimization algorithms and their implementation, inbuilt functions and kernels were used.

Next classifier used is Fully Connected Neural Network. A shallow neural network was implemented. Variety of neural network configurations were tried and top-1,5,10 accuracy values were used to choose among the models tried. Further, hyper-parameters of the neural network, which include:

```

class FFNetwork(FFNetwork):
    def __init__(self, num_hidden=250, init_method='xavier', activation_function='sigmoid', leaky_slope=0.1)
    def forward_activation(self, X)
    def grad_activation(self, X)
    def get_accuracy(self)
    def softmax(self, X)
    def forward_pass(self, X, params=None)
    def grad(self, X, Y, params=None)
    def fit(self, X, Y, epochs=1, algo="GD", l2_norm=False, lambda_val=0.8, display_loss=False, eta=1, save_fig=False, save_dir='Output-images/', save_name='FFNN_Regularization')
    def accuracy(self, y_hat, y)
    def cross_entropy(self, label, pred)
    def log_loss(self, yt, yp)
    def predict(self, X)
    def accuracy_n(self, y_hat, y, topk=(1,))

    A1
    A2
    layer_sizes
    leaky_slope
    gradients
    H1
    H2
    num_layers
    activation_function
    update_params
    params
    prev_update_params

```

Figure 5.2: Class FFNetwork

1. activation function (sigmoid, tanh, ReLU, Leaky ReLU, ..)
2. learning rate (0.01,0.001,..)
3. number of epochs (100s,1000s,..)
4. regularization method (L1, L2, ...)
5. type of initialization method (random, Xavier, He, ...)
6. loss function (cross entropy loss, MSE, ..)

were varied and the optimal configuration was determined.

Below is the description of the experiments conducted to check over fitting of the model and also visualize the efficiency of regularization. All the results here are obtained through use of Washington dataset.

For experimentation purpose and partially as a learning exercise, the complete neural network (not the optimal or final configuration presented) class was hand coded, including the back propagation. For this, no optimization algorithm was implemented and simple (Vanilla) Gradient Descent method was used. The loss function used was cross entropy loss.

First, the simple model was trained without regularization. To prevent over-fitting of the model to the training data, the input images, during data

augmentation was divided into training and validation sets. The validation set accuracy of the model was monitored to prevent over fitting. It was observed that the accuracy of model on validation set was correlated (and in par) with training set, with both reaching 100% accuracy within 100 epochs.

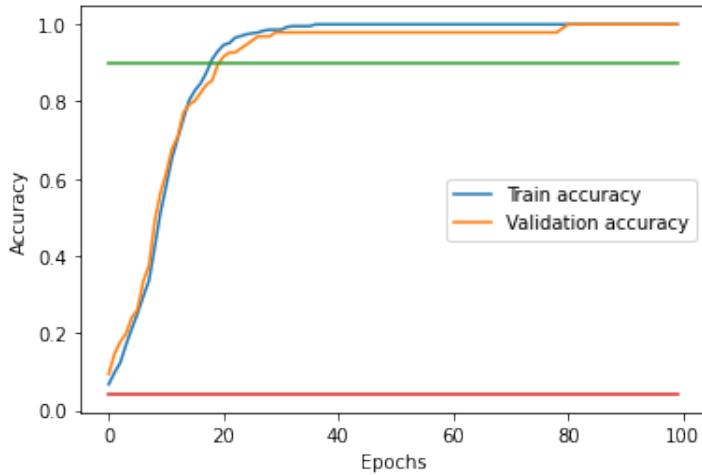
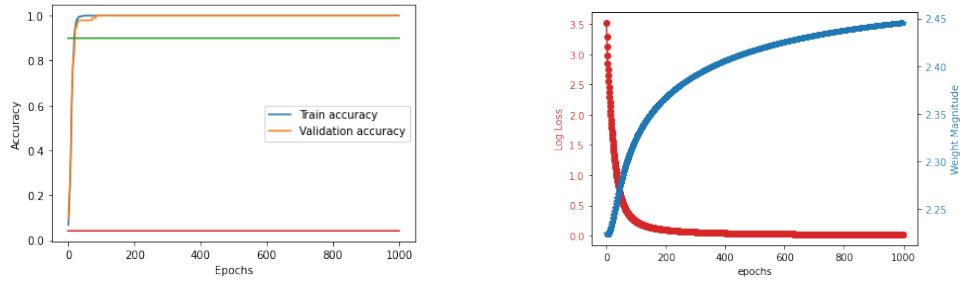


Figure 5.3: Without Regularization

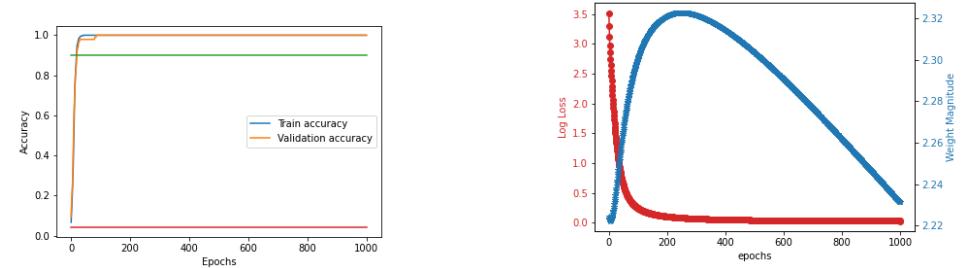
The green line represents the expected accuracy value on validation set. Further, as not much variability was present in the training set, with only small amplitude (0.1) noise added and two different expressions, we expect the validation set accuracy to not truly reflect lack of over-fitting. Thus, **L2 regularization** was included in the model. The weightage of regularization term was varied and loss plots were observed.

Once, the code for neural network was optimized in terms of hyper-parameters, focus was then shifted towards satisfying the requirement of real time performance by the system.

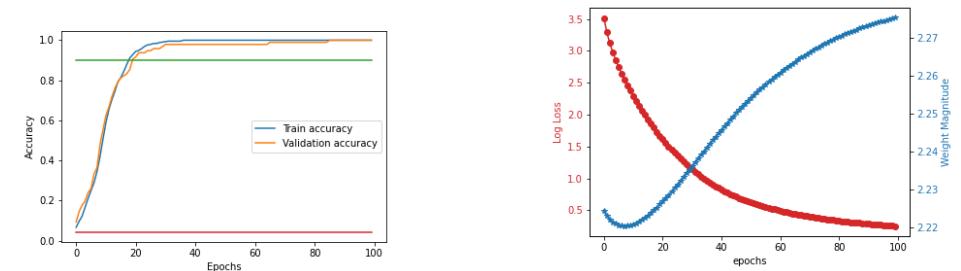
The class was re-defined using PyTorch's **nn.Sequential** with regularization included. The set of hyper-parameters along with their values chosen are displayed in the following table.



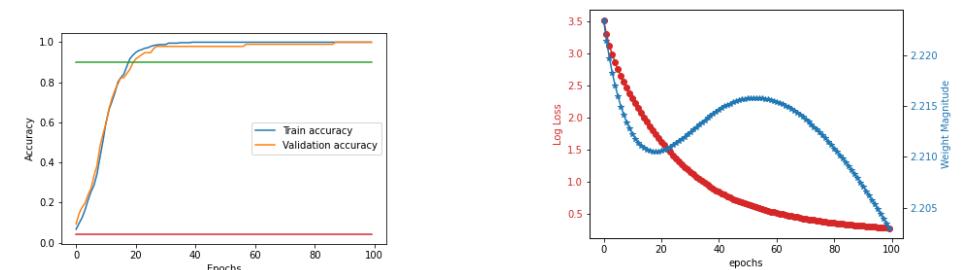
(a) $\lambda = 0.1$



(b) $\lambda = 1$



(c) $\lambda = 2$



(d) $\lambda = 5$

(e) Model Performance vs Regularization fraction

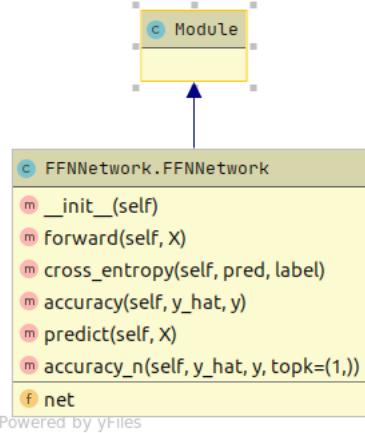


Figure 5.5: Class FFNNetowrk (without regularization)

Parameter	Value
Activation Function	Tanh (input scaled and shifted)
Regularization Method	L2 Regularization
Type of Initialization	Random
Optimization Algorithm	Stochastic Gradient Descent

The above are the parameters of the final neural network used for classification, which was trained using PyTorch on GPU.

As mentioned in the requirements sections, one of the requirements is real time execution (fast execution). Thus, the neural network was trained on GPU using PyTorch, making use of the massive parallelization offered by Nvidia CUDA. As a good practice during training, the training samples were shuffled using shuffle functionality imported from **sklearn.utils**. The testing accuracy did drop when training along with shuffling, but not by a very significant amount.

The back propagation step, now was performed using PyTorch’s Automatic Differentiation package. **`torch.autograd`** provides classes and functions implementing automatic differentiation of arbitrary scalar valued functions. It requires minimal changes to the existing code - one only need to declare torch Tensors for which gradients should be computed with the **`requires_grad=True`** keyword.[9]

Further, unlike in the previous execution, where simple Vanilla Gradient Descent was used for minimizing loss, here, Stochastic Gradient Descent

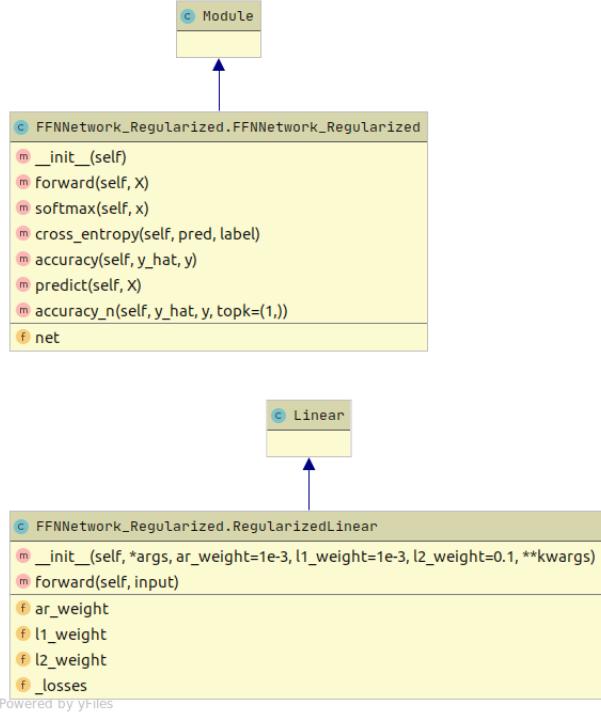


Figure 5.6: Class FFNNNetwork_Regularized

was used. Adam optimizer did perform well at initial epochs, but decreased with increase in number of epochs. Also, the updation of the weights and bias values is completed with just two command: `opt.step()` followed by `opt.zero_grad()`, where “`opt`” refers to the optimizer used, here, **optim.SGD** (`torch.optim` is a package implementing various optimization algorithm).

The activation functions tried are: sigmoid, tanh, ReLU and leaky ReLU. The performance of the FFNN with sigmoid and tanh were appreciable, with that with tanh better than with sigmoid. The use of ReLU (with Xavier initialization) or leaky ReLU (with He initialization) for the same purpose produces very bad accuracy values, and also some times ”nan” value for loss (since, the inputs to the activation function had values exceeding unity). Thus, the two activation functions were not considered for further search in hyper parameter space.

Also, an attempt was made to define a custom activation function. Input

scaled and shifted version of tanh function - $f(x) = \tanh(a(x - x_0))$, where, a is the scale parameter, and x_0 is the shift parameter was employed.

In terms of performance, the accuracy values are same as that obtained previously. Amazingly, the time for fitting the model was drastically reduces from of the order of a few minutes to a few seconds.

Also, the hidden layers were batch normalized with drop-out, which upgraded the performance by 1 – 3%. In general, even with increase in the number of epochs, the performance of the FFNN with dropout was observed to be lower than those without dropout. Yet, drop-out was included in one of the hidden layers as it is always a good practice to do so, while building generalized neural networks.

5.1 Ensemble Classifier

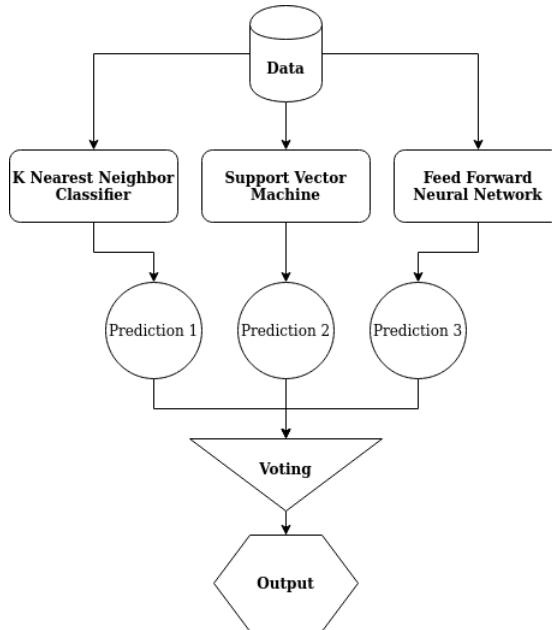


Figure 5.7: Ensemble Classifier

The above three classifier considered were further used to create an ensemble of classifier. The weightage for each classifier's prediction proportional to [0.3,0.3,0.4].

5.2 Face Alignment

As a general procedure in any face recognition project, effort was made for face alignment[10] using the algorithm described in the flowchart below.[11]

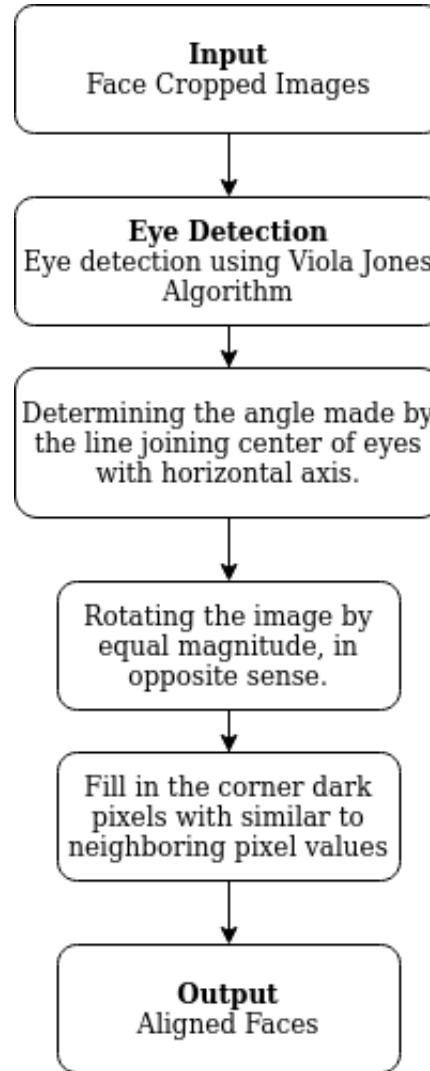


Figure 5.8: Face Alignment Flow chart

Eye detection using - *haarcascade_eye.xml* - on face cropped images, resulted in detection of mouth (teeth), dark hair patch (at corner of face cropped image), nose, and other irrelevant background portions as eyes and

in some cases, did not detect any thing. In very few images did the face alignment algorithm function efficiently.

To over come these issues and achieve face alignment, an algorithm was designed to detect eyes in a fool proof manner. The algorithm is briefed below:

1. Detect eyes using Viola Jones Algorithm (i.e., using OpenCV's Eye Cascade Classifier) with fine values of scaleFactor and minNeighbors parameters. It is fine even if the algorithm detects other facial regions as eyes.
2. Next, divide the frontal face image into 4 quadrants with origin at the center.
3. Now, consider only those regions, detected by OpenCV's eye cascade classifier, which are in either first or second quadrant. (using Cartesian coordinates convention)
4. Next, out of those filtered out regions, consider those which are closer towards the left/right borders in quadrant II/I.
5. Use the above considered regions as detected eye locations and use them for face alignment by finding the tilt angle of the line joining the centers of the two regions and rotating the image in opposite sense, thus, achieving face alignment.
6. In case only one of the eyes is detected (which is observed to happen in very rare cases), ignore it and consider the images for further processing, whether aligned or not.



Figure 5.9:
Unaligned

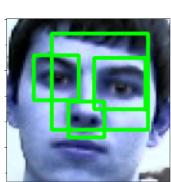


Figure 5.10:
Eye Detection

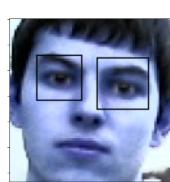


Figure 5.11:
Best eyes



Figure 5.12:
Rotation
Angle

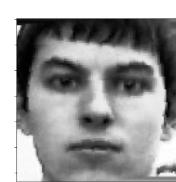


Figure 5.13:
Aligned Face

5.3 Saving and Loading Models

The model so trained during the making of this project has been saved using PyTorch’s save function as **state_dict** (using `torch.save(model.state_dict(), PATH)`). Two models have been trained and saved - one trained by looping orderly over the training data and other by looping over the training data randomly. These saved models can be loaded and either be used directly for classification purpose (on testing samples) or used for further training to improve the model further. This was achieved using PyTorch’s **load_state_dict** function (i.e., using `model.load_state_dict(torch.load(PATH))`) [12]. For SVM, the values corresponding to optimum classifier has been listed in the sections above. These values can be plugged into the classifier directly and used for classification purpose. Below is the comparison of all the classifiers considered above (including the ensemble classifier).

Also, the matrices required for conversion of face-space images (vectors) to lower-dimension space vectors are saved using numpy’s **savetxt** function. Similarly, even the training data and labels can be stored using same numpy function.

These saved numpy arrays can be re-loaded using numpy’s **loadtxt** function.

5.4 Washington Dataset

The input images were all “.tga” images. Hence **PIL** module’s **Image** function was used to read the images. Further, image processing was done using **OpenCV**’s module functions. As the number of images per person for training was very low (2 per person), data augmentation was done by appending noisy images to the data. Random Gaussian (white) noise was added to each original training samples and the number of images per person was extended to 22 (11 per facial expression per person). In the process, the training images were even rotated by an angle and the range ($-5^\circ, 5^\circ$) so as to train the model to perform well even on data in which the input images are not aligned straight. Further, the above set of images were also border cropped by random pixels and resized, to increase variability of the input samples. At the end, each new augmented image was histogram equalized. Histogram equalization of augmented data is necessary, since all

the test images are also histogram equalized. It was, in fact observed that the performance of each classifier improved significantly (increase in test accuracy by $\approx 7\%$ for kNN, $\approx 2\%$ for SVM and $\approx 4\%$ for FFNN) after histogram equalizing the augmented noisy data.

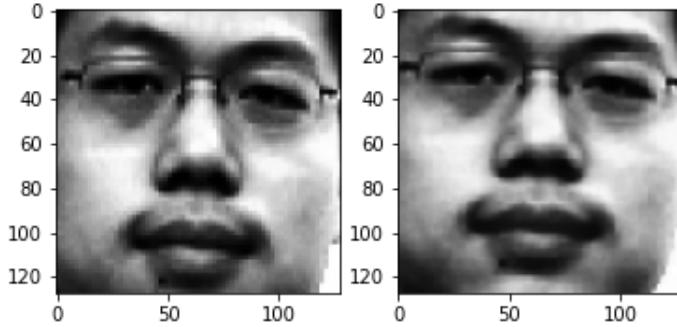


Figure 5.14: Training image vs Cropped and resized test image (an example)

By trial and error, it was found that cropping the section [28:116,20:108] of each image, produced an accuracy of the order of 70% in contrast to an accuracy of the order of 25% when the Viola Jones algorithm based cropped images were straight forwardly used for face recognition. Thus, resemblance of test images with training samples is pivotal in performance of any trained classifier in face recognition. 126 principal components were considered for classification.

The testing samples, i.e., group photos were passed through the face detection pipeline to detect faces. Further, these detected faces were aligned using the above described face alignment pipeline. The scaleFactor and minNeighbours used for face detection haar cascade classifier are 1.2 and 5, while those for eye detection are 1.07 and 1.

Next step is the classification step. First, simplest algorithm, i.e., kNN algorithm was applied and an accuracy of 72.916% was obtained using this algorithm with $k = 1$. High value of accuracy for $k = 1$ signifies that the training and testing samples have high similarity and also low variability of

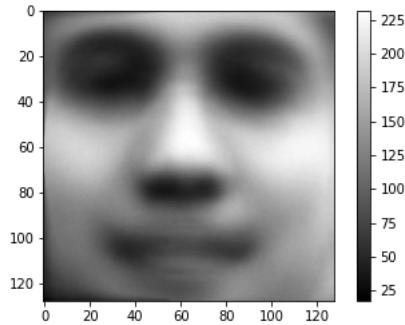


Figure 5.15: Mean Face (computed from training set)



Figure 5.16: Eigen Faces used for Face Recognition (First 54 Principal Components)

in the training samples. Thus, we observe that our effort of inducing variability by augmenting noisy data to the training samples was not very successful. This also, shows the need for truly different frontal face images of people with as high a number of faces per person as possible.

Next classifier used is SVM. This classifier produced an accuracy of



Figure 5.17: Testing Samples: Cropped Faces from the group photos

85.41%. The specifications of the SVM classifier used are as follows:

Parameter	Value
C	10
Kernel	rbf
γ	0.001

Next classifier used is FFNN. The neural network configuration is as shown below. This neural network configuration produced an accuracy comparable to that of SVM. Further, the hidden layers were batch normalized and neurons in the middle most layer were dropped-out with a probability of 0.2, which increased the test accuracy of the classifier from 79.166% to 91.666%!

Finally, all the predicted outputs of each class were weighted and used in the ensemble classifier to get the final prediction. The weightage used is : [0.3,0.3,0.4] for [kNN,SVM,FFNN].

```
FFNNetwork-Regularized(
    (net): Sequential(
        (0): RegularizedLinear(in_features=126, out_features=250, bias=True)
        (1): BatchNorm1d(250, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): Hakuna_Matata()
        (3): RegularizedLinear(in_features=250, out_features=150, bias=True)
        (4): BatchNorm1d(150, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (5): Dropout(p=0.2, inplace=False)
        (6): Hakuna_Matata()
        (7): RegularizedLinear(in_features=150, out_features=48, bias=True)
        (8): BatchNorm1d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (9): Hakuna_Matata()
        (10): RegularizedLinear(in_features=48, out_features=24, bias=True)
        (11): BatchNorm1d(24, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (12): Hakuna_Matata()
        (13): Softmax(dim=None)
    )
)
```

Figure 5.18: Neural Network Configuration

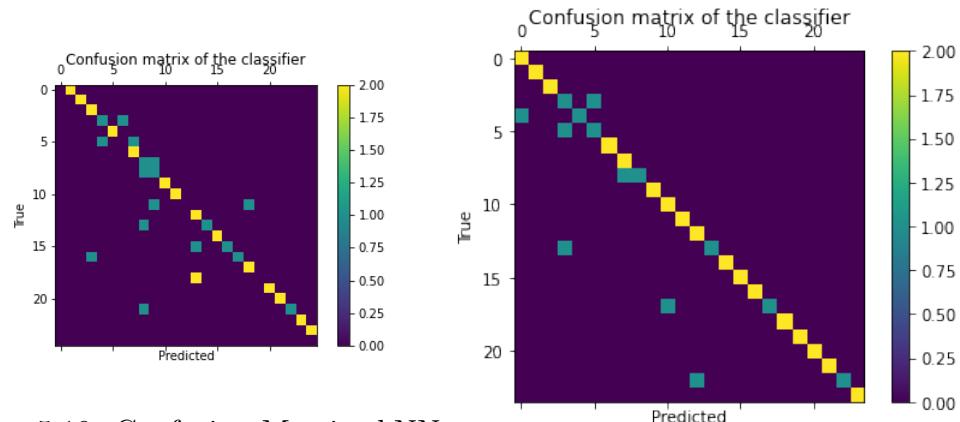


Figure 5.19: Confusion Matrix: kNN

Figure 5.20: Confusion Matrix: SVM

5.5 Comparison of Classifiers Performance

Along with the accuracy of the test samples based on prediction of each classifier the precision, recall and F-scores were used to compare the classifier performance. The numerical values are observed to depend on the initial random seeding of the augmented data. The best obtained model parameters of SVM and FFNN have been displayed in above relevant sections. The model file corresponding to best accuracy scores obtained using FFNN has been saved, which can be loaded and used.

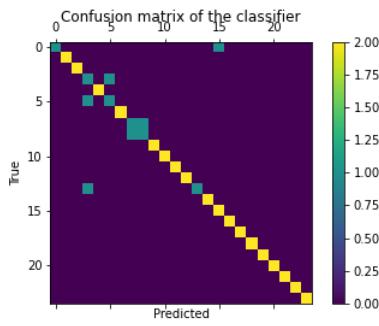


Figure 5.21: Confusion Matrix: FFNN

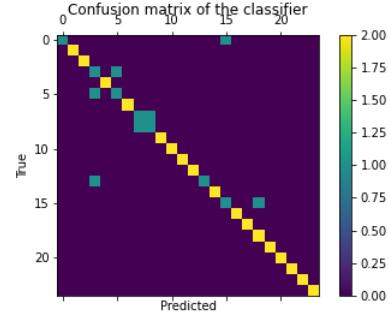


Figure 5.22: Confusion Matrix: Ensemble Classifier

Classifiers	KNN	SVM	Neural Network	Ensemble Classifier
Accuracy	i. For k=1:72.91666 ii. For k=2:50.0 iii. For k=3:50 iv. For k=4:35.41666 v. For k=5:29.1666	Training Set Score:100% Test Set Score:85.4167%	<ul style="list-style-type: none"> Top-1 Training Accuracy- 100% Test Accuracy-91.666% Top-5 Training Accuracy-100% Test Accuracy-93.75% Top-10 Training Accuracy-100% Test Accuracy-95.83% 	<ul style="list-style-type: none"> Top-1 Testing Accuracy-87.5% Top-5 Testing Accuracy-93.75% Top-10 Testing Accuracy-93.75%
Comments	Maximum Accuracy was obtained for K=1, with accuracy-72.9166	Best Performance Parameters: <ul style="list-style-type: none"> Best C (Degree of Tolerance)—10 Best Kernel—RBF(Radial Basis Function) Best Gamma (parameter of the Gaussian RBF)—0.001 	<ul style="list-style-type: none"> Loss Before Training- 3.218 Loss After Training- 1.4169 Time for Training Model—44.004s 	Accuracy for Top-5 and Top-10 Classifier is the same.

Figure 5.23: Comparison of Classifiers for Dataset Specific Neural Network

5.6 Performance of Generalized Neural Network

5.7 LFW Dataset (Subset)

The input images were all “.jpg” images. Hence OpenCV’s function was used to read the images. Further, image processing was done using **OpenCV**’s module functions. As the dataset was large, and partially due to other constraints such as time, computation power and skill, only a small subset of the complete dataset was used. Here, the data was selected by

```

FFNNNetwork_Regularized(
    (net): Sequential(
        (0): RegularizedLinear(in_features=262, out_features=2048, bias=True)
        (1): BatchNorm1d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): Dropout(p=0.1, inplace=False)
        (3): Hakuna_Matata()
        (4): RegularizedLinear(in_features=2048, out_features=1024, bias=True)
        (5): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (6): Hakuna_Matata()
        (7): RegularizedLinear(in_features=1024, out_features=512, bias=True)
        (8): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (9): Hakuna_Matata()
        (10): RegularizedLinear(in_features=512, out_features=250, bias=True)
        (11): BatchNorm1d(250, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (12): Hakuna_Matata()
        (13): RegularizedLinear(in_features=250, out_features=93, bias=True)
        (14): BatchNorm1d(93, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (15): Hakuna_Matata()
        (16): Softmax(dim=None)
    )
)

```

Figure 5.24: Generalized Neural Network

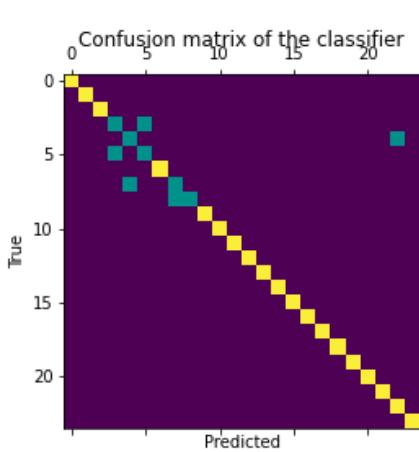


Figure 5.25: Confusion Matrix: Generalized FFNN

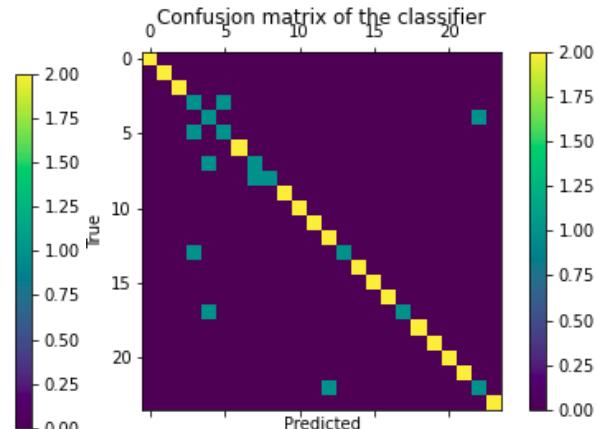


Figure 5.26: Confusion Matrix: Ensemble Classifier (with generalized FFNN)

considering at-most 4 people with same alphabet in their first name. Those with low number of faces per person, such as those with only 1 or 2 were rejected and those with 3 or higher were considered, with upper limit on the number of faces per person limited to 4. Data augmentation was done by appending noisy images to the data, similar to that in Washington Dataset. The number of images per person was extended to 20-21. In the

Classifiers	KNN	SVM	Neural Network	Ensemble Classifier
Accuracy	i. For k=1:70.83 ii. For k=2:52.0 iii. For k=3:52.08 iv. For k=4:45.83 v. For k=5:35.416	Training Set Score:100% Test Set Score:85.4167%	<ul style="list-style-type: none"> • Top-1 Training Accuracy- 100% Test Accuracy-87.5% • Top-5 Training Accuracy-100% Test Accuracy-93.75% • Top-10 Training Accuracy-100% Test Accuracy-95.83% 	<ul style="list-style-type: none"> • Top-1 Testing Accuracy-83.3% • Top-5 Testing Accuracy-91.66% • Top-10 Testing Accuracy-93.75%
Comments	Maximum Accuracy was obtained for K=1, with accuracy-70.83	Best Performance Parameters: <ul style="list-style-type: none"> • Best C (Degree of Tolerance)—10 • Best Kernel—RBF(Radial Basis Function) • Best Gamma (parameter of the Gaussian RBF)—0.001 	<ul style="list-style-type: none"> • Loss Before Training-3.197 • Loss After Training-1.418 • Time for Training Model—95.819s 	Accuracy for Top-5 and Top-10 Classifier is the same.

Figure 5.27: Comparison of Classifiers for Generalized Neural Network

process, the training images were even rotated by an angle and the range ($-5^\circ, 5^\circ$) so as to train the model to perform well even on data in which the input images are not aligned straight. At the end, each new augmented image was histogram equalized. Histogram equalization of augmented data is necessary, since all the test images are also histogram equalized.

By trial and error, the best crop section after face detection was found to be [14:62,8:54] of each image with image size of 64x64. 262 principal components were considered for classification.

As no pre-defined test samples were available, the validation samples were obtained from the augmented dataset. Also, one of the original images of each person was used to testing the classifier performance. Further, these detected faces were aligned using the above described face alignment pipeline. The scaleFactor and minNeighbours used for face detection haar cascade classifier are 1.01 and 1, while those for eye detection are 1.05 and 1.

Next step is the classification step. First, simplest algorithm, i.e., kNN algorithm was applied and an accuracy of 98.347% was obtained using this algorithm with $k = 1$. Thus, once again, we observe that our effort of inducing variability by augmenting noisy data to the training samples was not very successful.

Next classifier used is SVM. This classifier produced an accuracy of 95.86%. The specifications of the SVM classifier used are as follows (same as those used in Washington Dataset):



Figure 5.28: First 25 Eigen Faces used for Face Recognition (total 262 Principal Components)

Parameter	Value
C	10
Kernel	rbf
γ	0.001

Next classifier used is FFNN. The neural network configuration is as shown below. This neural network configuration produced an accuracy comparable to that of SVM. Further, the hidden layers were batch normalized and neurons in the middle most layer were dropped-out with a probability of 0.1, as a result, overall validation accuracy of the classifier was obtained to be 92.24%.

Finally, all the predicted outputs of each class were weighted and used in the ensemble classifier to get the final prediction. The weightage used is : [0.3,0.3,0.4] for [kNN,SVM,FFNN].

We observe from the loss plot that though the accuracy values increase the relative decrease in loss is very low.

5.8 Comparison of Classifiers Performance

Below is the chart comparing the performance of each classifier:

```

FFNNNetwork_Regularized(
    (net): Sequential(
        (0): RegularizedLinear(in_features=262, out_features=2048, bias=True)
        (1): BatchNorm1d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): Dropout(p=0.1, inplace=False)
        (3): Hakuna_Matata()
        (4): RegularizedLinear(in_features=2048, out_features=1024, bias=True)
        (5): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (6): Hakuna_Matata()
        (7): RegularizedLinear(in_features=1024, out_features=512, bias=True)
        (8): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (9): Hakuna_Matata()
        (10): RegularizedLinear(in_features=512, out_features=250, bias=True)
        (11): BatchNorm1d(250, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (12): Hakuna_Matata()
        (13): RegularizedLinear(in_features=250, out_features=93, bias=True)
        (14): BatchNorm1d(93, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (15): Hakuna_Matata()
        (16): Softmax(dim=None)
    )
)

```

Figure 5.29: Neural Network Configuration

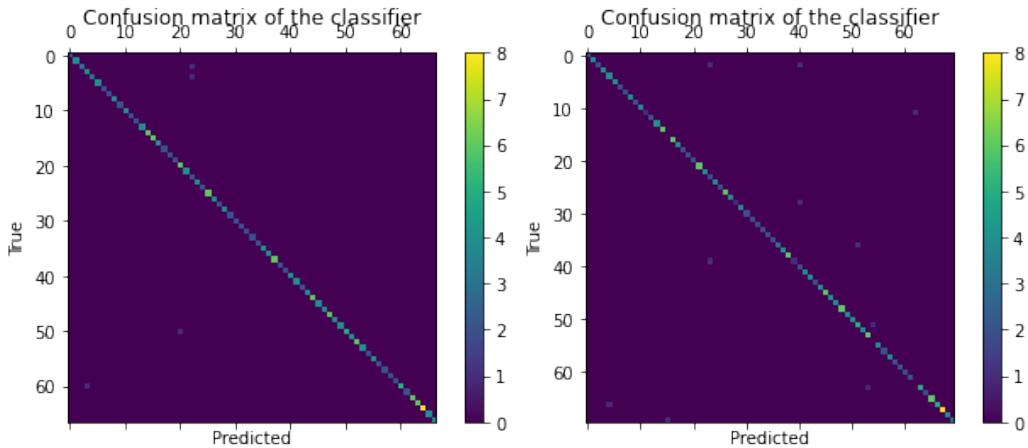


Figure 5.30: Confusion Matrix: kNN Figure 5.31: Confusion Matrix: SVM

Along with the accuracy of the test samples based on prediction of each classifier the precision, recall and F-scores were used to compare the classifier performance. The numerical values are observed to depend on the initial random seeding of the augmented data. The best obtained model parameters of SVM and FFNN have been displayed in above relevant sections. The model file corresponding to best accuracy scores obtained using FFNN has been saved, which can be loaded and used.

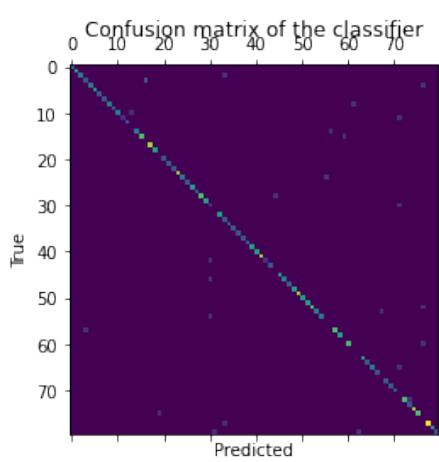


Figure 5.32: Confusion Matrix: FFNN

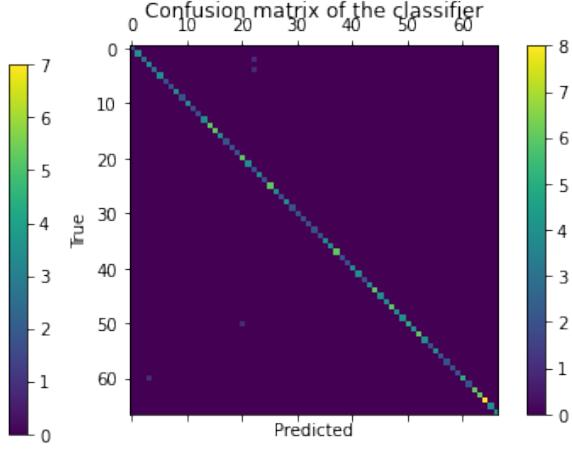


Figure 5.33: Confusion Matrix: Ensemble Classifier

5.9 Fetch Olivetti Faces Dataset

The input images are available online, hence, can be loaded `sklearn.datasets`. There are 10 faces per person and 40 different persons, which sums up to 400 images in the complete dataset. As the number of faces per person is quite high, no data was augmented. 50 principal components were considered for classification.

As no pre-defined test samples were available, the validation samples were obtained from the given dataset. Further, there was no need to align faces

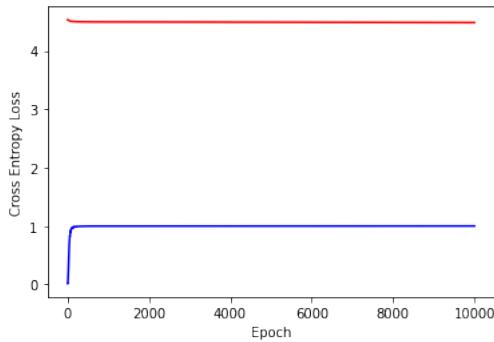


Figure 5.34: Loss Plot (Red) and Accuracy Plot (Blue)

Classifiers	KNN	SVM	Neural Network	Ensemble Classifier
Accuracy	i. For k=1:99.137% ii. For k=2:78.017% iii. For k=3:78.017% iv. For k=4:63.362% v. For k=5:54.741%	Training Set Accuracy:100% Test Set Score: 94.39%	<ul style="list-style-type: none"> • Top-1 Training Accuracy- 100% Test Accuracy-92.241% • Top-5 Training Accuracy-100% Test Accuracy-95.089% • Top-10 Training Accuracy-100% Test Accuracy-96.552% 	<ul style="list-style-type: none"> • Top-1 Testing Accuracy-94.827% • Top-5 Testing Accuracy-99.138% • Top-10 Testing Accuracy-89.5833
Comments	Maximum Accuracy was obtained for K=1, with accuracy-99.137%	Best Performance Parameters: <ul style="list-style-type: none"> • Best C (Degree of Tolerance)—10 • Best Kernel—RBF(Radial Basis Function) • Best Gamma (parameter of the Gaussian RBF)—0.001 	<ul style="list-style-type: none"> • Loss Before Training-4.5325 • Loss After Training-4.4833 • Time for Training Model —136.911s • No. of Epochs = 10000 	Accuracy for Top-5 and Top-10 Classifier is the same.

Figure 5.35: Comparison of Classifiers



Figure 5.36: Eigen Faces used for Face Recognition (50 Principal Components)

since all the data in this dataset were pre-processed. Also, there was no need for face detection for the same reason.

Next step is the classification step. First, simplest algorithm, i.e., kNN algorithm was applied and an accuracy of 100% was obtained using this algorithm with $k = 1$. The decrease in the accuracy with increase in k value is found to slow (relative to that in other two datasets). Thus, once again,

we observe that our effort of inducing variability by augmenting noisy data to the training samples was not very successful.

Next classifier used is SVM. This classifier produced an accuracy of 100%. The specifications of the SVM classifier used are as follows (same as those used in Washington Dataset):

Parameter	Value
C	10
Kernel	rbf
γ	0.001

Next classifier used is FFNN. The neural network configuration is as shown below. It is the same as that used on Washington dataset. This neural network configuration produced an accuracy comparable to that of SVM. Further, the hidden layers were batch normalized and neurons in the

```
FFNNNetwork-Regularized(
    net): Sequential(
        (0): RegularizedLinear(in_features=126, out_features=250, bias=True)
        (1): BatchNorm1d(250, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): Hakuna_Matata()
        (3): RegularizedLinear(in_features=250, out_features=150, bias=True)
        (4): BatchNorm1d(150, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (5): Dropout(p=0.2, inplace=False)
        (6): Hakuna_Matata()
        (7): RegularizedLinear(in_features=150, out_features=48, bias=True)
        (8): BatchNorm1d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (9): Hakuna_Matata()
        (10): RegularizedLinear(in_features=48, out_features=24, bias=True)
        (11): BatchNorm1d(24, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (12): Hakuna_Matata()
        (13): Softmax(dim=None)
    )
)
```

Figure 5.37: Neural Network Configuration

middle most layer were dropped-out with a probability of 0.1, as a result, overall validation accuracy of the classifier was obtained to be 100%.

Finally, all the predicted outputs of each class were weighted and used in the ensemble classifier to get the final prediction. The weightage used is : [0.3,0.3,0.4] for [kNN,SVM,FFNN].

We observe from the loss plot that loss decreases rapidly even after the accuracy values saturates.

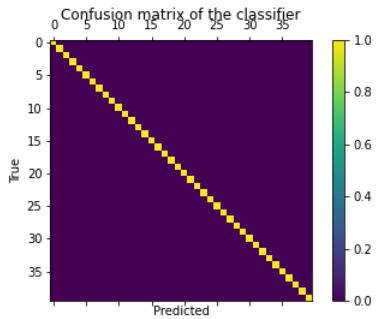


Figure 5.38: Confusion Matrix: kNN

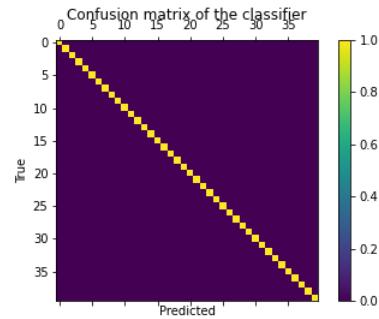


Figure 5.39: Confusion Matrix: SVM

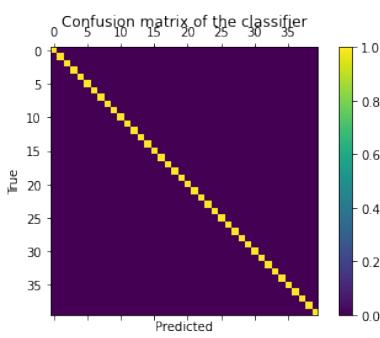


Figure 5.40: Confusion Matrix: FFNN

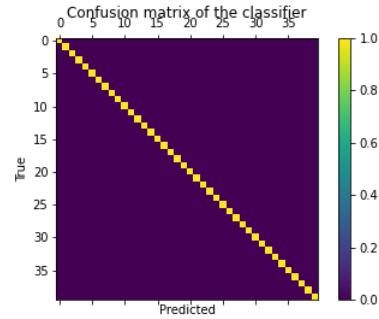


Figure 5.41: Confusion Matrix: Ensemble Classifier

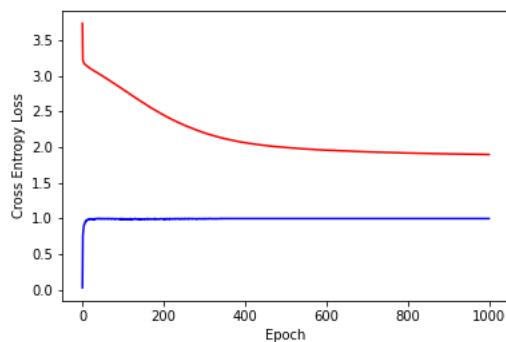


Figure 5.42: FFNN Loss Curve

5.10 Comparison of Classifiers Performance

Along with the accuracy of the test samples based on prediction of each classifier the precision, recall and F-scores were used to compare the classifier performance. The numerical values are observed to depend on the initial random seeding of the augmented data. The best obtained model parameters of SVM and FFNN have been displayed in above relevant sections. The model file corresponding to best accuracy scores obtained using FFNN has been saved, which can be loaded and used. Due to high quality of the dataset, (i.e., good pre-processed data) we have obtained 100% accuracy on validation set. Yet, nothing can be assured of the performance of the classifier on the testing samples (if exists), as the model accuracy depends crucially, as shown in the above two datasets, on the degree of similarity of the test samples with the training samples, which calls for significant effort on pre-processing the testing samples to obtain accurate predictions.

Chapter 6

Attendance System

One of the application of the designed system is **Face Recognition based Biometric Attendance System**. Here, we present our work related to the face recognition based bio-metric attendance system.

6.1 System Description

The trained model of SVM and FFNN are saved in a file. These models are then loaded and used for classification of test samples, which here can be group photo of people attending any given session, conference, meeting or like. This system is also advantageous as the test images serve as a proof for a person's attendance.

The system so designed takes the test images in a given directory, detects faces in each image and using the pre-trained models, identifies the person in the image and marks that person as present by marking “1” for that person in the excel sheet (attendance register) in the column corresponding to the date specified. In case no attendance register or excel sheet is found, it creates a new one automatically. Also, in case no date is specified, it marks the attendance on the date corresponding to execution day of the file. For kNN based classification, the saved version of the training data (in reduced dimension space) and corresponding labels are used.

Further, the detected faces are framed using OpenCV’s haar cascade classifier for frontal face detection and the name of the person as identified by the system is tagged (printed) to the frame. This is to aid in detecting false-positives.

Below are the set instructions to use the attendance system.

1. Save the test files in a separate folder (here, **Test_images**) with no other files in the same folder.
2. Create another folder to save the name tagged images (here, **Labelled_images**).
3. Open the code for attendance system and change the values of the variables - *path, folder, data_folder, save_folder* to appropriate directory paths. (*data_folder* is folder containing test samples and *save_folder* is that in which the name tagged images needs to be saved)
4. Open terminal in the same directory where the code for attendance system is present.
5. Type the following command:
\$python3 <attendance_system>.py <date><key>.
Here, date is in the format dd.mm.yyyy. The delimiter used for a given attendance file must be same and must not be “/” (since it will be considered as directory structure). Note that the order of specifying the arguments does not matter.
6. The key value needs to be set to “1” if one wants to save the name tagged images in a separate folder as a record. Else, it will be stored in the temporary folder, where it will be erased each time the code is run.
7. In case the date of updation of attendance is the same of the date of execution of the above command, then the date in the above command need not be specified.

Here, we present two types of attendance updating codes -

1. One which considers the test files to be group photos and that each identified person in each of the test image is in the attendance list
2. Other which considers the test files to be individual photos of each person, which might contain other people in the background, yet having major area covered by the person to be identified.

In case the test data is in the former form, then the file “update_attendance_Washington.py” or “update_attendance_Washington_FFNN.py” need to be used and in case of test data being in latter form “update_attendance_LFW.py” or “update_attendance_LFW_FFNN.py” needs to be used.

Note here that the above two files require the following data/arrays to function properly:

- (a) Linearized Mean face array (here, “M.txt”)
- (b) Face space to Eigen space vector Transformation matrix (here, “C.txt”)
- (c) Training data (here, “X_train.txt”)
- (d) Training data class labels (here, “Y_train.txt”)
- (e) FFNN Model file (here, “CLASS_FFNNNetwork.txt”)
- (f) SVM kernel parameters (i.e., kernel, C and gamma)

Also, the file names mentioned above are only the first part of the file names, with appropriate code the suffix “_LFW” or “_Washington” needs to be used.

Here, as mentioned above, there are two files for each form of test images. One, with no suffix, which considers all three classifiers and identifies the person based on the ensemble classifier output. The code with suffix “_FFNN” uses only the FFNN for identification and is faster than the former one. The reason for the fastness of the code which considers FFNN only is that kNN based classification is very time consuming, along with the file read-write processes.

As shown in the below snap-shots, the date may or may not be specified in the terminal. In case the attendance updation date is not the same as the date of execution of the file, only then is it necessary to mention the appropriate date.

Below are the snap-shot of the terminal outputs and the results obtained.

```

kiran@kiran-HP-Pavilion-dv6-Notebook-PC:/media/kiran/New Volume/SEM_6/Machine Learning for Signal Processing/Course Project/FINAL/SC17B106_SC17B150/Attendance_System$ python3 update_attendance_LFW.py
/home/kiran/.local/lib/python3.6/site-packages/torch/nn/modules/container.py:100: UserWarning: Implicit dimension choice for softmax has been deprecated. Change the call to include dim=X as an argument.
    input = module(input)
Total Time for Attendance Updation : 104.05464720726013s

```

(a) update_attendance_LFW.py

```

kiran@kiran-HP-Pavilion-dv6-Notebook-PC:/media/kiran/New Volume/SEM_6/Machine Learning for Signal Processing/Course Project/FINAL/SC17B106_SC17B150/Attendance_System$ python3 update_attendance_LFW_FFNN.py 27/05/2020
/home/kiran/.local/lib/python3.6/site-packages/torch/nn/modules/container.py:100: UserWarning: Implicit dimension choice for softmax has been deprecated. Change the call to include dim=X as an argument.
    input = module(input)
Total Time for Attendance Updation : 25.90116262435913s

```

(b) update_attendance_LFW_FFNN.py

```

kiran@kiran-HP-Pavilion-dv6-Notebook-PC:/media/kiran/New Volume/SEM_6/Machine Learning for Signal Processing/Course Project/FINAL/SC17B106_SC17B150/Attendance_System$ python3 update_attendance_Washington.py 27/05/2020
/home/kiran/.local/lib/python3.6/site-packages/torch/nn/modules/container.py:100: UserWarning: Implicit dimension choice for softmax has been deprecated. Change the call to include dim=X as an argument.
    input = module(input)
Total Time for Attendance Updation : 27.53991961479187s

```

(c) update_attendance_Washington.py

```

kiran@kiran-HP-Pavilion-dv6-Notebook-PC:/media/kiran/New Volume/SEM_6/Machine Learning for Signal Processing/Course Project/FINAL/SC17B106_SC17B150/Attendance_System$ python3 update_attendance_Washington_FFNN.py
/home/kiran/.local/lib/python3.6/site-packages/torch/nn/modules/container.py:100: UserWarning: Implicit dimension choice for softmax has been deprecated. Change the call to include dim=X as an argument.
    input = module(input)
Total Time for Attendance Updation : 16.964905500411987s

```

(d) update_attendance_Washington_FFNN.py

```

kiran@kiran-HP-Pavilion-dv6-Notebook-PC:/media/kiran/New Volume/SEM_6/Machine Learning for Signal Processing/Course Project/FINAL/SC17B106_SC17B150/Attendance_System$ python3 update_attendance_Washington_FFNN.py 28.05.2020 1
/home/kiran/.local/lib/python3.6/site-packages/torch/nn/modules/container.py:100: UserWarning: Implicit dimension choice for softmax has been deprecated. Change the call to include dim=X as an argument.
    input = module(input)
Total Time for Attendance Updation : 19.809237241744995s

```

(e) Specifying the key value to be “1” to save the name tagged images in a separate folder

Figure 6.1: Usage of Attendance system files

6.1.1 Problem of False Positives

One of the major limitations of the system designed is the problem of false positives. In other words, marking of person not present as present. To overcome this issue as well as the issue of true negatives, we save name tagged files in a separate folder for verification. Thus, by physical verification, this problem can be overcome.

Here, this calls for the requirement of the person verifying to be acquainted with the names and faces of each person, which is not always possible. We intend to overcome in the near future to be able to build a fool proof bio-metric frontal face recognition based identification system.

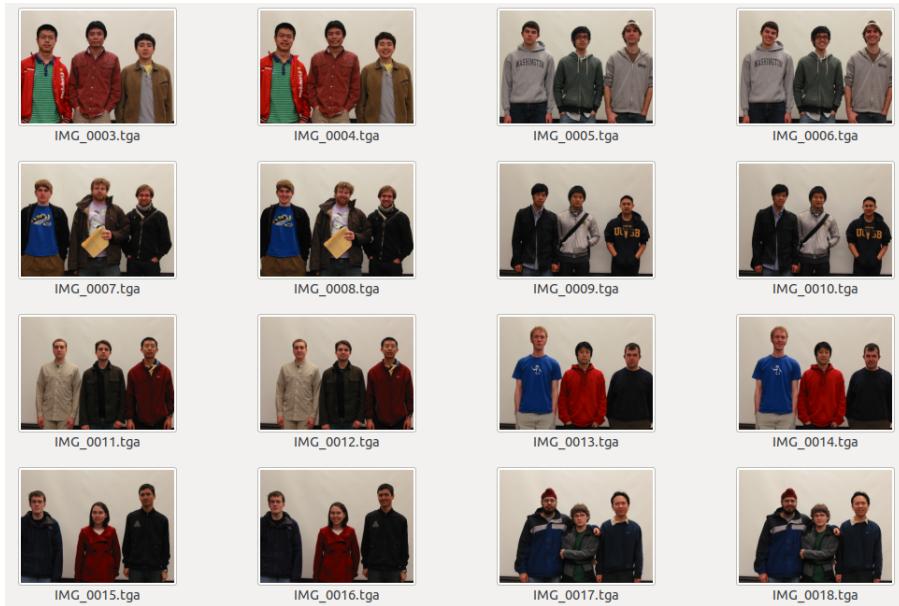


Figure 6.2: Input to file update_attendance_Washington.py



Figure 6.3: Output of file update_attendance_Washington.py

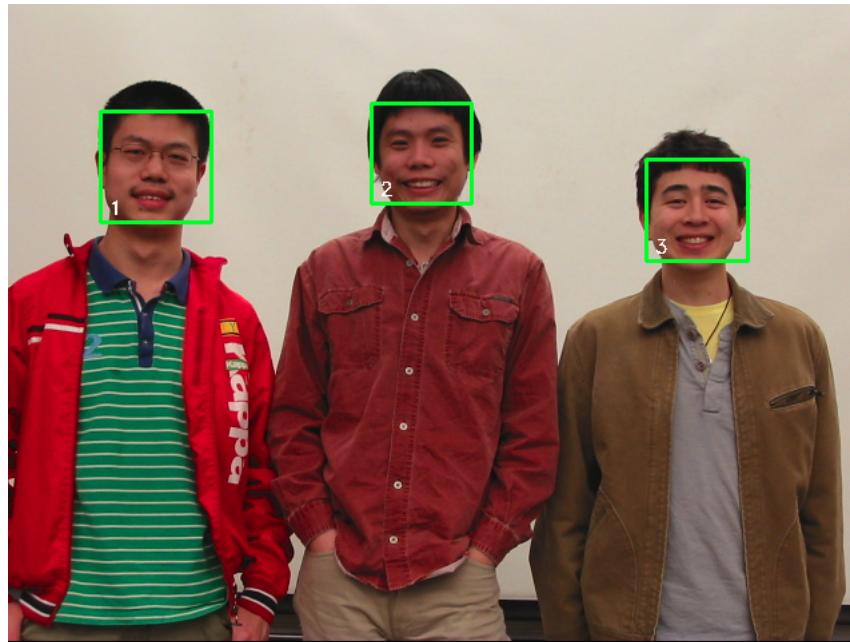


Figure 6.4: Correct Identification of all people in the test image

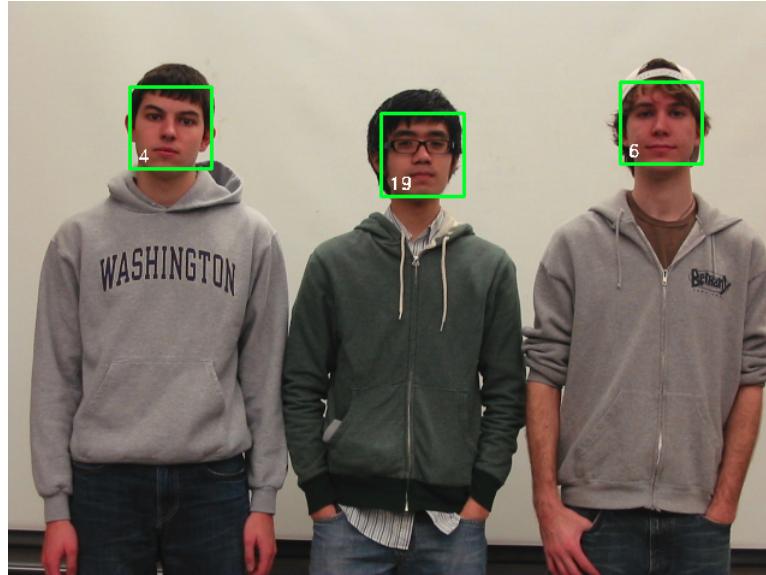


Figure 6.5: Incorrect Identification of Person with ID 5

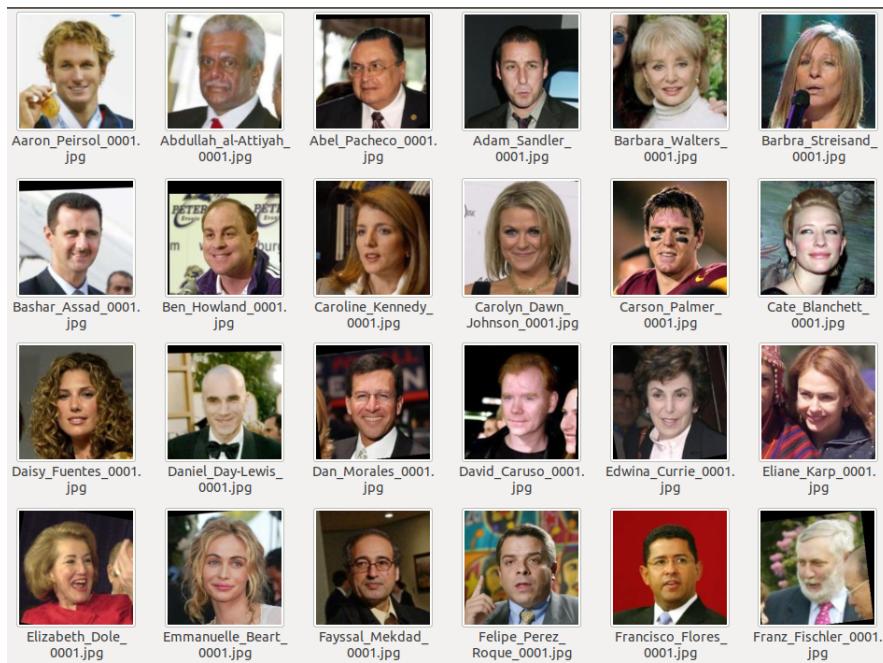


Figure 6.6: Input to file update_attendance_LFW.py

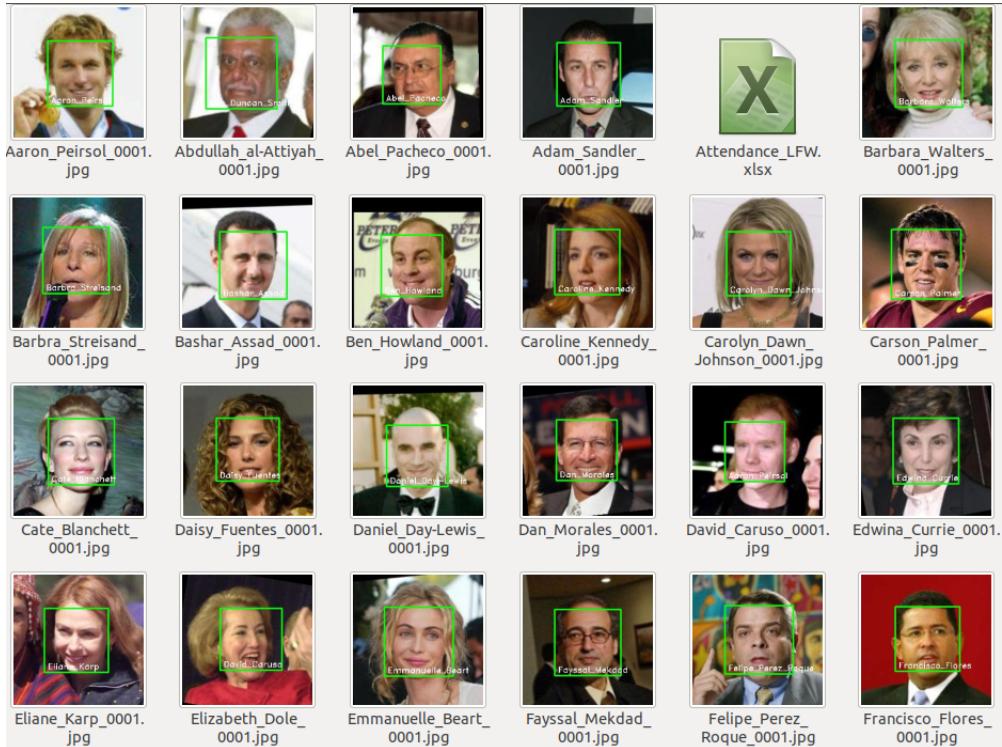


Figure 6.7: Output of file update_attendance_LFW.py



Figure 6.8: Sourav Ganguly (Correct) Figure 6.9: J K Rowling (Incorrect Identification)

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Name	25/05/2020	26/05/2020	27/05/2020									
2	Aaron_Peiris	1	1	1									
3	Abdullah_al-Attiyah	0	0	0									
4	Abel_Pacheco	1	1	1									
5	Adam_Sandler	1	1	1									
6	Barbara_Walters	1	1	1									
7	Barbra_Streisand	1	1	1									
8	Bashar_Assad	1	1	1									
9	Ben_Howland	1	1	1									
10	Cate_Blanchett	1	1	1									
11	Carson_Palmer	1	1	1									
12	Caroline_Kennedy	1	1	1									
13	Carolyn_Dawn_Johnson	1	1	1									
14	Daisy_Fuentes	1	1	1									
15	Dan_Morales	1	1	1									
16	Daniel_Day-Lewis	1	1	1									
17	David_Caruso	1	1	1									
18	Edwina_Currie	1	1	1									
19	Eliane_Karp	1	1	1									
20	Elizabeth_Dole	0	0	0									
21	Emmanuelle_Beart	1	1	1									
22	Faysal_Mekdad	1	1	1									
23	Felipe_Perez_Roque	1	1	1									
24	Francisco_Flores	1	1	1									
25	Franz_Fischler	1	1	1									
26	GL_Peiris	1	1	1									
27	Gary_Carter	1	1	1									
28	George_Galloway	1	1	1									

(a) Attendance Sheet Updation of LFW Data Sub-set

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Name	25/05/2020	26/05/2020	27/05/2020											
2	1	1	1	1											
3	2	1	1	1											
4	3	1	1	1											
5	4	1	1	1											
6	5	1	1	1											
7	6	1	1	1											
8	7	1	1	1											
9	8	1	1	1											
10	9	1	1	1											
11	10	1	1	1											
12	11	1	1	1											
13	12	1	1	1											
14	13	1	1	1											
15	14	1	1	1											
16	15	1	1	1											
17	16	1	1	1											
18	17	1	1	1											
19	18	1	1	1											
20	19	1	1	1											
21	20	1	1	1											
22	21	1	1	1											
23	22	1	1	1											
24	23	1	1	1											
25	24	1	1	1											
26															
27															
28															

(b) Attendance Sheet Updation of Washington Dataset

Figure 6.10: Attendance Updation in Excel File

Chapter 7

Conclusion

The presented face detection and recognition system successfully detects faces from input group photos and identifies (classifies) the people with appropriate training set used. Higher the variability of the face images used for training and also higher the number of faces per person, better is the performance of the classifiers designed. Provision has been made for data augmentation in case of less number of training samples. The performance of the classifiers SVM and FFNN is appreciable. The performance of the FFNN may vary with the dataset. Thus, to account for this, a set of models with different neural network configuration, activation functions (some models having different activation functions at different layers), etc., which can be used in case the best model suggested does not provide satisfactory results. Necessary implementation of face alignment have been done, which can be tried on dataset under consideration, as the used algorithm based on haar cascade classifier for eye detection could not successfully detect eyes in all the training images in the dataset. The overall performance of the system crucially depends on the dataset used, in case the dataset contains images which can be aligned using the haar cascade eye-detection based algorithm, better performance can be expected. The attendance system presented here is ready to be used in small scale applications, but in case of there being large number of subjects the problem of false positives (and equivalently, true negatives) becomes a serious limitation.

Chapter 8

Future Enhancement

The work so reported can be extended by enhancing the robustness of the classifier by using datasets with more variability per face image. The existing work can be extended to be used in biometric systems such as the automatic attendance system. Further, the other methods for face detection can be used. One such example is the use of Convolutional Neural Networks, where the task of feature extraction (here, features extracted are face space vectors) can be done by the set of convolution layers and max pool or average pool layers. This can be followed by a set of fully connected layer for classification task. Some of the CNN architectures which are currently used for similar applications are: LeNet, AlexNet, GoogLe Net, Resnet, VGG-16, VGG-19, etc., by altering the output layer to output number of outputs as the number of people in the dataset. Further, with batch normalization and dropout, better architectures can be constructed for the same purpose.

Also, as discussed in the chapter of Attendance systems, better algorithms and methods needs to be designed to overcome the issue of false positives and make the system more fool proof for use in commercial applications.

Appendix A

Reconstructed Face images of Washington Dataset



Figure A.1: Reconstructed Training samples



Figure A.2: Reconstructed Testing samples

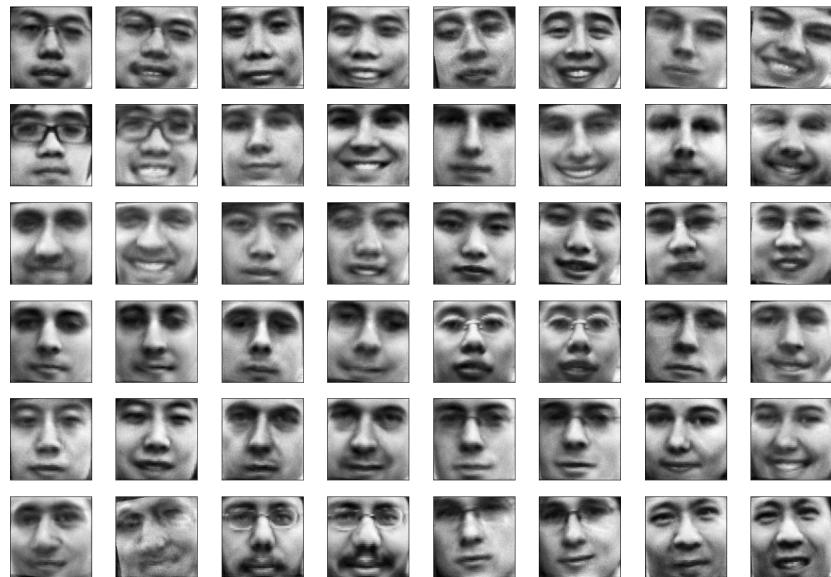


Figure A.3: Reconstructed Validation samples

Appendix A

Reconstructed Face images of Fetch Olivetti Faces Dataset



Figure A.1: Example Faces



Figure A.2: Reconstructed Training samples

Appendix A

Reconstructed Face images of LFW Data-Sub-Set



Figure A.1: Reconstructed Training Samples

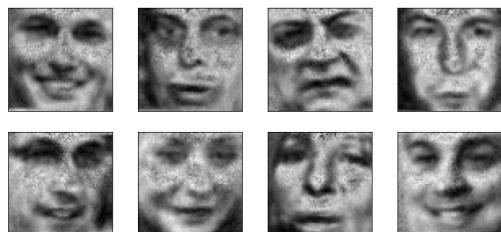


Figure A.2: Reconstructed Validation samples

Appendix A

Pseudo Code

Algorithm 1 Data Augmentation

```
1: for  $iteration = 1, 2, \dots, N$  do
2:   Read training images
3:   Resize and histogram equalize the image
4:   Linearize the image into 1-D array form and save to training set
5:   Add white noise to and rotate and fill the copied image ( $-5^\circ, 5^\circ$ )
6:   Histogram equalize and append image to training set
7: end for
```

Algorithm 2 Principal Component Analysis

```
1: Determine U,S,V matrices by SVD of mean subtracted images matrix
2: The first  $n$  rows of V corresponds to the Principal Components (PCs)
3: Map each image vector to face space vector using PC matrix
```

Algorithm 3 Face Detection

```
1: for  $iteration = 1, 2, \dots, \hat{N}$  do
2:   Read in the testing (group) photo
3:   Detect faces using OpenCV's haar cascade classifiers for face detection
4:   Crop and resize detected frontal faces similar to training images
5:   Histogram equalize and map linearized images to face space.
6: end for
```

Algorithm 4 Face Alignment

```
1: for  $iteration = 1, 2, \dots, \hat{N}$  do
2:   Detect eyes using OpenCV's Eye Cascade Classifier.
3:   Consider only those regions detected by eye cascade classifier which
      are in either first or second quadrant (which must contain the eyes) of
      the face image (following Cartesian coordinates convention with origin
      at the center)
4:   Out of those filtered out regions, consider those which are closer to-
      wards the right/left borders in quadrant I/II, as eyes respectively.
5:   if Two eyes are detected then
6:     Use detected as eye locations to find tilt angle of line joining the
       centers of the two regions and rotating the image in opposite sense.
7:   else
8:     if Only one eye is detected then
9:       Ignore it and consider the images for further processing,
       whether aligned or not.
10:    end if
11:   end if
12: end for
```

Algorithm 5 Classification using kNN

- 1: Use Euclidean distance for kNN algorithm for required k values.
 - 2: Compute accuracy scores and confusion matrix.
-

Algorithm 6 Classification using SVM

- 1: Search through the space of parameters of SVM (kernel, C, γ) for most optimal SVM based classifier and classify test samples.
 - 2: Compute accuracy scores and confusion matrix.
-

Algorithm 7 Classification using FFNN

- 1: Map all data to torch tensor and design a neural network.
 - 2: Port all data and network class to Nvidia GPU (if available)
 - 3: **for** $epoch = 1, 2, \dots, Max_Epochs$ **do**
 - 4: Train FFNN and compute *Cross Entropy Loss* with L2 regularization
 - 5: Using Stochastic GD, back propagate and update weights and biases.
 - 6: **end for**
 - 7: Use trained model for face identification on the test samples.
 - 8: Compute top-1,5,10 accuracy values and confusion matrix
-

Algorithm 8 Classification using Ensemble Classifier

- 1: Port predicted classes from FFNN to *numpy nd-array* from *torch.tensor*
 - 2: One hot encode all the outputs (predictions) from each classifiers
 - 3: Assign weightage to each classifier's prediction proportional to it's individual (top-1) accuracy on the test set and determine voted prediction
 - 4: Determine top-1,5,10 accuracy values and confusion matrix
-

Bibliography

- [1] <https://github.com/parulnith/Face-Detection-in-Python-using-OpenCV/blob/master/Face>
- [2] <https://www.datacamp.com/community/tutorials/face-detection-python-opencv>
- [3] <https://arxiv.org/pdf/1901.02452.pdf>
- [4] <http://cmusatyalab.github.io/openface/>
- [5] http://www.scholarpedia.org/article/K-nearest_neighbor
- [6] <https://towardsdatascience.com/a-friendly-introduction-to-support-vector-machines-svm-925b68c5a079>
- [7] <https://mlfromscratch.com/neural-networks-explained/>
- [8]
<https://courses.cs.washington.edu/courses/cse455/09wi/projects/project4/web/project4.html>
- [9] <https://pytorch.org/docs/stable/autograd.html>
- [10] https://github.com/contail/Face-Alignment-with-OpenCV-and-Python/blob/master/align_faces.py
- [11] <https://sefiks.com/2020/02/23/face-alignment-for-face-recognition-in-python-within-opencv/>
- [12] https://pytorch.org/tutorials/beginner/saving_loading_models.html