PROJECT NO. 2

Development and Testing of a Face Recognition System

A Project Report

Submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

in

ELECTRICAL ENGINEERING

Submitted By:

Animesh Anant Sharma **Ankita Puwar** **Avishek De**

Guided By:

Dr. Manoj Tripathy

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY ROORKEE

ROORKEE – 247667 (INDIA)

APRIL 2017

DECLARATION

We hereby declare that the work which is presented in this project, entitled **Development and Testing of a Face Recognition System**, towards the fulfilment of requirements for the award of the degree of **Bachelor of Technology** in **Electrical Engineering,** submitted to the Department of Electrical Engineering Indian Institute of Technology Roorkee, is an authentic record of our own work carried out during the period of August 2016 to April 2017 under the guidance of **Dr. Manoj Tripathy**, Department of Electrical Engineering, IIT Roorkee.

The matter embodied in this project report, to the best of our knowledge, has not been submitted for the award of any other degree elsewhere.

Date: April 12, 2017

Animesh Anant Sharma	**Ankita Puwar**	**Avishek De**
13115021	13112011	13113030

This is to certify that the above statement made by candidates is correct to the best of my knowledge.

DR. MANOJ TRIPATHY
Associate Professor
Department of Electrical Engineering
IIT Roorkee

ACKNOWLEDGEMENT

We would like to thank our supervisor Dr. Manoj Tripathy for providing us an opportunity to work on this challenging project. His support and continued motivation helped us to gain insight into various aspects of research and development.

We would also like to extend our thanks to global open source communities for their help in providing the implementations for various modules of our project. Our project depended on several open source libraries and publically available datasets for efficient and effective implementations.

We are very much thankful to Dr. S. P. Srivastava, Head of Department (Electrical Engineering) and the laboratory assistant of the Signals and Systems laboratory for providing us with the necessary infrastructure and equipments required to complete this project.

Last but not the least, we would also like to thank our friends who have provided us with constant support and motivation. We also sincerely acknowledge everyone who has directly or indirectly helped us in our project including our friends, whose images we have taken for implementing several algorithms used throughout the project. This project could not have been completed without them.

ABSTRACT

A real time face recognition system is developed by using a Convolutional Neural Network based face recognition algorithm with a Support Vector Machines based classifier. The project comprises of four major components: an imaging system which generates an image of the class of students; a selector module for selecting a portion of the generated image; a detection algorithm to detect faces in the selected image portion; and a recognition module responsive to the detection algorithm for determining whether a detected image of a person resembles one of a reference set of images of individuals. Eigenfaces, Fisherfaces and Local Binary Pattern Histogram (LBPH) were used as benchmark algorithms to compare the proposed work. The method of Convolutional Neural Networks (CNN) and Support Vector Machines (SVM) based image classifier consistently outperformed all the benchmark algorithms with Radial Basis Function (RBF) kernel based SVM providing the highest accuracy of around 92%. The dataset consisted of images of around 33 students of IIT Roorkee and 67 subjects from the Labelled Faces in the Wild (LFW) dataset. Standard LFW experiment was performed on the combined dataset and the training time for different algorithms were also compared with a Graphics Processing Unit (GPU) based CNN giving the best possible performance. Later multiple faces in a single image were recognized with good confidence measure. Then a web application was developed so as to get real time implementation of the face recognition project which can be readily employed in classrooms to get attendance marking. This application can be trained using images of the video feed and then can be employed to recognize faces at later stage. The face detection was done using dlib library written in C++, preprocessing was done using OpenCV, feature extraction was done using Lua and finally face recognition was performed with the help of script written in python.

CONTENTS

LIST OF FIGURES

ABBREVIATIONS

CNN	Convolutional Neural Network
SVM	Support Vector Machine
LBPH	Local Binary Pattern Histogram
LFW	Labelled Faces in the Wild
RBF	Radial Basis Function
GPU	Graphics Processing Unit
HOG	Histogram of Oriented Graphs
PCA	Principal Component Analysis
FRUE	Face Recognition in Unconstrained Environment
FLD	Fisher's Linear Discriminant
ReLU	Rectified Linear Unit
KKT	Karush Kuhn Tucker
CASIA	Chinese Academy of Sciences Institute of Automation
LDA	Linear Discriminant Analysis

Chapter 1

Introduction and Literature Survey

1.1 Introduction

A facial recognition system is an application capable of identifying or recognising a person from a digital image or a video frame from a video source, which works by comparing the selected facial features from the image to the face database. There are two main types of face recognition systems, geometric and photometric. The geometry based algorithm looks at distinguishing local facial features like the distances between different facial points whereas the photometric method is a template based method. It converts an image into values and compares these values with templates, which are developed by statistical tools like Support Vector Machine (SVM), Principal Component Analysis (PCA) etc. Over the past few years, face recognition has become a popular area of research in computer vision and pattern recognition where promising results are produced by deep learning, in particular Convolutional Neural Network (CNN).

1.2 Motivation

The face recognition is a solved problem with numerous real world applications already in place. It is a non-invasive, biometric technology finding applications in fields from medicine to law enforcement to psychology, and thus it has attracted attention of researchers and neuroscientists. This simple process of face recognition, which comes naturally and effortlessly to human beings, is however a challenge to implement due to the different conditions in which the human face can be found. It is believed that the progress in research in fields of computer vision and pattern recognition will provide useful insights to psychologists into how human brain memorises face. But face recognition softwares with an impressive accuracy and adequate performance have been property of multinational corporations like Facebook and Google. This has changed with the development of open source projects like OpenFace which are built on the similar lines of GoogleNet, a face recognition application based on deep learning architecture. The proposed method in this report is based on OpenFace since the attendance recording can be easily automated using face recognition techniques.

The problem of finding an ideal facial feature which is robust for Face Recognition in Unconstrained Environments (FRUE) still remains. However in the past few years,

convolutional neural networks (CNN) have achieved very impressive results on FRUE. The proposed application belongs to the same class of problem. With the development of CNN and large publicly available datasets, the application has been developed with adequate accuracy and functionality.

Present methods include the manual way or employing biometric techniques for attendance verification in academic institutes, industries or offices. The manual way is time consuming and leads to proxy attendances which are difficult to eliminate. The biometric attendance marking was one possible solution to the problem but the machine is troublesome to handle since the lecture interrupts multiple times in case of malfunctioning of machine. It has been observed that in a large class of 100 students, this becomes a very time consuming process. The face recognition can be a good option since adequate number of group photos can be taken in order to compile the attendance and the training dataset can be regularly updated in order to account for the changes in appearance.

1.3 Problem Statement

The problem statement is to develop a face recognition system that can be used to detect faces in a class so that attendance marking system can be automated. The system must be reliable with adequate accuracy and should be efficient enough so that a real time implementation can be obtained. The system should be able to recognize faces in a crowd of 20 people so that attendance can be taken in batches of 20 people at a time and then the angle of the camera can be changed to obtain the image of the next batch of 20 people and so on.

1.4 State of the Art Methods

Face recognition has been active research topic since the 1970's [1] and much advancements have been done since. Jafri and Arabnia provide comprehensive survey of face recognition techniques upto 2009. These include statistical methods such as Principal Component Analysis (PCA), which represent faces as combination of eigenvectors [2]. Eigenfaces and fisherfaces are benchmark methods in PCA-based face recognition. Fisherfaces improves PCA method by using Linear Discriminant Analysis (LDA) method better suited for classification. [2-3]

Today's best performing face recognition techniques are based on convolutional neural networks. Facebook's DeepFace and Google's FaceNet systems yield the highest accuracy [4-5]. However, these deep neural network-based techniques are trained with private datasets

which contain millions of images taken from social media and other sources that are orders of magnitude higher than what is available for research.

The first approach that was explored was that of PCA. The face recognition is associated with a high dimensional problem if all the pixel values of image are considered. It is very helpful if all the relevant features are somehow extracted from the image and then classification algorithms are applied on these reduced set of features. First and foremost relevant work in the application of PCA to face recognition problem was done in [2]. The main demerit of this application is that the accuracy associated with it is not satisfactory. Also deep learning based networks are outperforming this approach and PCA was implemented for the purpose of learning.

The second approach is based on Fisherfaces. Fisher's Linear Discriminant (FLD) is an example of a class specific method, in the sense that it tries to "shape" the scatter in order to make it more reliable for classification [3]. Both PCA and FLD have been used to project the points from 2D down to 1D. PCA actually smears the classes together so that they are no longer linearly separable in the projected space. Although PCA achieves larger total scatter, FLD achieves greater between-class scatter, and, consequently, classification is simplified.

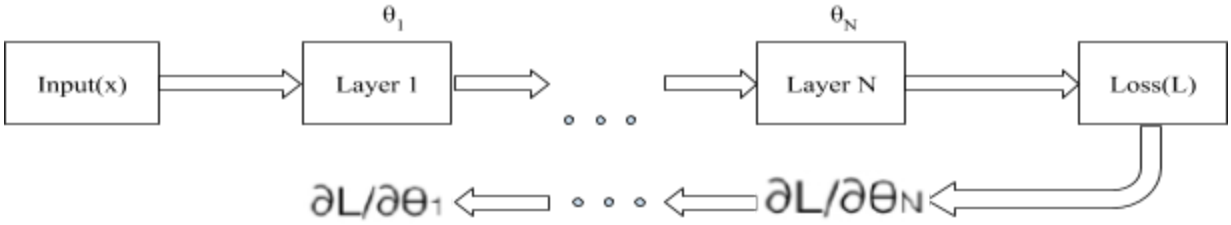


Fig 1. Training method for feed-forward neural network

The third approach is based on deep learning. The deep neural networks are based on neural networks, as shown in Fig. 1, with the number of layers increased. More relevant information regarding the same can be found at [6]. The proposed approach was implemented on the same lines as that of the OpenFace project developed in [7]. Also was tailored for the desired application. The results were compared with the results obtained from the application of Fisherfaces, Eigenfaces and local binary pattern histograms [8].

1.5 Layout of Report

The preprocessing step is explained in chapter 2, feature extraction using CNN is explained in chapter 3 and finally the classification techniques are introduced in chapter 4. After this, the development of web application and dataset selection is given in chapter 5 and then methodology and results are explained in chapter 6 and 7 respectively.

Chapter 2

Face Detection and Face Alignment

2.1 Introduction

The input to the feature extractor cannot be raw image since deep neural network will not be able to perform efficiently. As a result the data is preprocessed using ***dlib*** and affine transformation. This is an important step in the pipeline since it improves the overall efficiency of the project. The feature extractor can then work on the preprocessed input to provide adequate representation of the face which can be used to classify the subject image.

2.2 Face Detection Using Dlib [7]

Most modern cameras have the feature of face detection. This ensures that all the available faces in the photo are focussed on before the shot is made. But, this can also be used to identify the areas of the image containing a face and passing it onto the next stage of the pipeline. One of the examples for face detection is shown in Fig. 2.

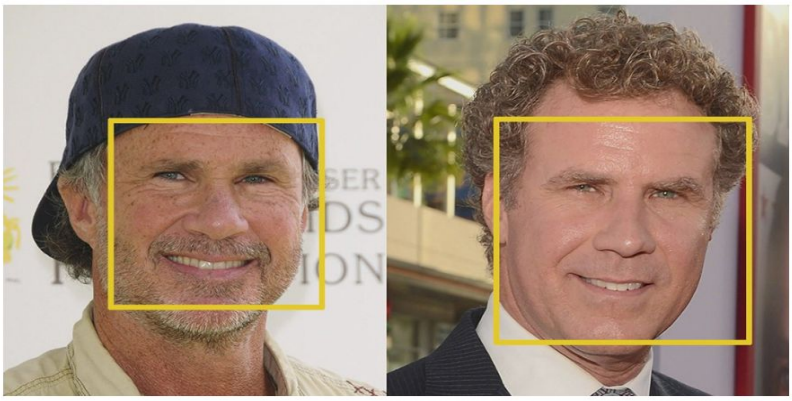


Fig. 2. Faces detected in images [9]

Detection of faces is done using the Histogram of Oriented Gradients (HOG) method in ***python*** and ***dlib***. The problem in analyzing pixels without preprocessing is that dark and light images of the same person will have different pixel values. But in HOG, each pixel is replaced by an arrow which shows the gradient of light from darkness to light. Thus both dark images and bright images of the same person will still be represented equally according to this method. Rather than repeating the process for each individual pixel the image is divided into small

squares of 16x16 pixels each. In each square, as shown in Fig. 3, the number of gradient arrows in each direction is counted and the square is finally replaced with the direction that was present maximum number of times. Eventually, the part of the image that looks similar to available HOG of face samples is extracted and detected as a face. The end result is we convert the original image into a very simple representation of a face which is easy to process and store for the next steps in the pipeline.

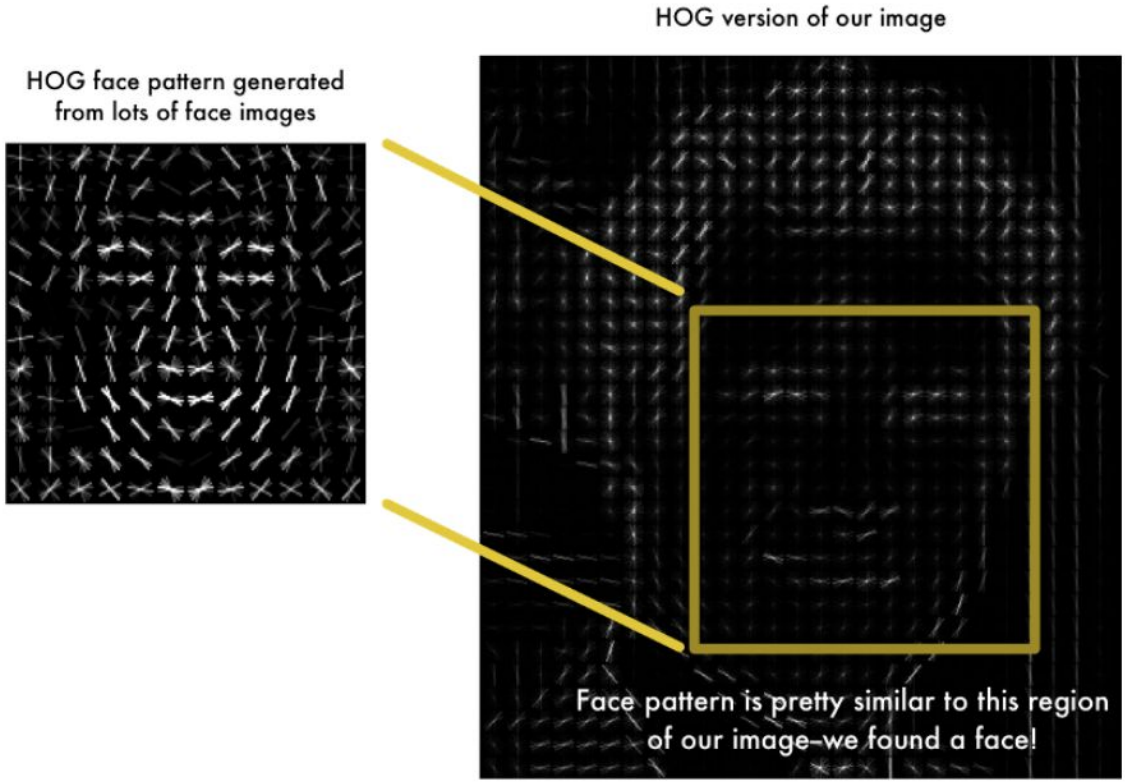


Fig. 3. Face Detection using HOG Technique [9]

2.3 Face Alignment through Affine Transformation

Different orientations of the face look completely different to the computer and thus it adds to the difficulty in the process of face recognition. Thus after detection of the face affine transformation is done so that position of eyes, nose and lips is same for all images i.e. the image becomes upright. Face landmark estimation algorithm [7] determines the 68 landmark features for each input image. The algorithm determines features like the inner edge of both eyes, top of chin. Rotation and scaling is done to get perfectly centred positions for the eyes and nose and lips. The affine transformation, which outputs a normalised fixed size image is fed to the neural network for feature extraction.

Fig. 4. 68 face landmarks for affine transformation [9]

Fig. 4 gives the 68 points determined on the subject image. These points are then used to make the image upright so that the feature extraction can be performed with more efficiency.

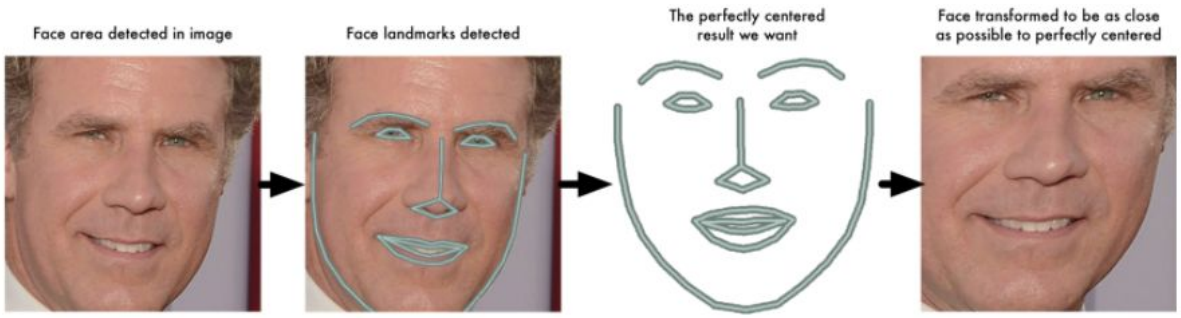


Fig. 5. Depiction of face alignment using affine transformation [9]

Fig. 5 gives one of the examples for face alignment. The detected face is given as input and then the 68 landmark features are detected. After this the image is made upright with the help of the template given in Fig. 4.

2.4 Summary

This chapter introduced the preprocessing steps which consists of face detection and face alignment. Face detection was done with the help of HOG algorithm and the code was written in C++ and the ***dlib*** library was imported. Similarly face alignment was based on affine transformation.

Chapter 3

Convolutional Neural Networks

3.1 Introduction

The preprocessed image is very high dimensional and the classifier will not be able to segregate images efficiently. Therefore, a feature extractor is employed so that important information can be extracted from the image and provided. Classical feature extraction techniques include principal component analysis and histogram methods but these methods use set templates in order to get features. The deep learning framework proposed here provides much better feature extraction in the sense that the whole method is dynamic. This points towards a process where the feature extraction method is also trained with the help of back propagation and employed to get representation for each image. This does not lead to formation of set template from the training dataset. Instead it strives to provide important information from the subject image.

3.2 Convolution Filters

The first layer in a CNN is always a Convolutional Layer. Consider the case where the image input is an array of 36 x 36 x 3 pixel values. A convolutional layer can be thought of as a flashlight which is shining on the top left side of the input image covering an area of 7 x 7 in the input image. Now, imagine that the flashlight is moving across the whole input image. In technical terms, this flashlight is referred to as a filter(or sometimes a neuron) and the area that it is shining over is called the receptive field of the filter. The filter, or the neuron, in turn is an array of weights (i.e. numbers or parameters) having depth same as the depth of the input image, thus having the dimensions of this filter is 7 x 7 x 3, in this case. Now consider first position of the filter in top left corner of the image. As the filter is moving, or convolving, around the input image, values in the filter are convolved (or element wise multiplied) with the original pixel values of the image. These multiplications are then summed up (here this would be 75 multiplications in total, since we have 7 x 7 x 3) to obtain a single number which denotes the value when the particular filter is at the top left of the image. This process is then repeated for every location on the input volume. The stride, i.e. the number of elements by which filter

is moved, is usually kept as 1 but that can be varied accordingly.. Every position of filter on the input image generates a number as shown in Fig. 6. After moving the filter over all the positions, an array of 32 x 32 is obtained, which is known as an activation map (or feature map) for that filter.

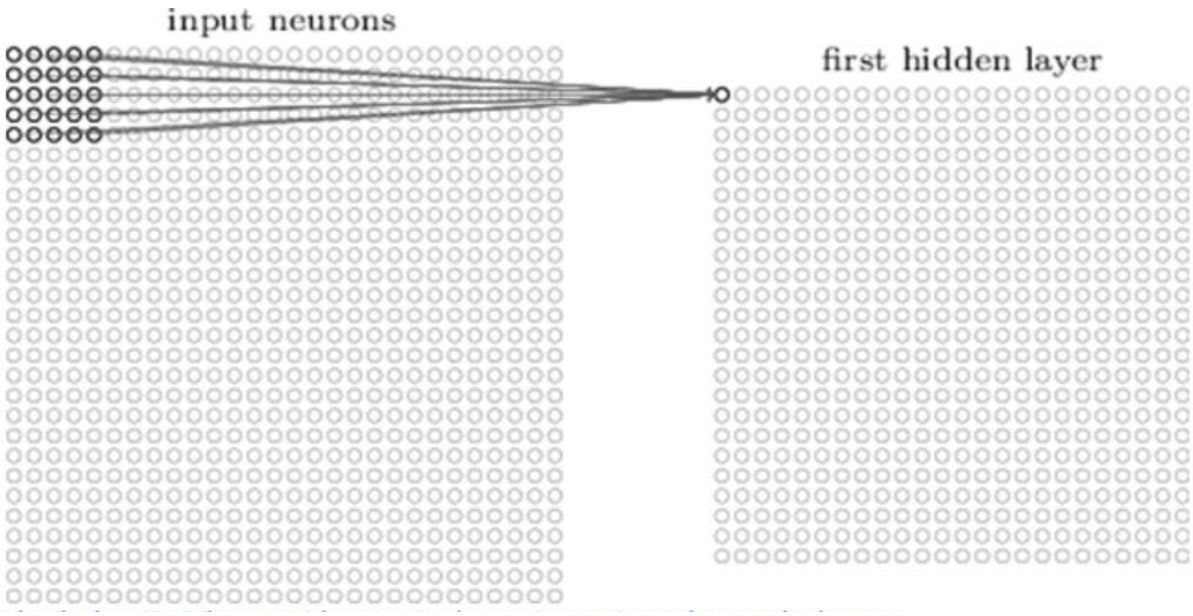
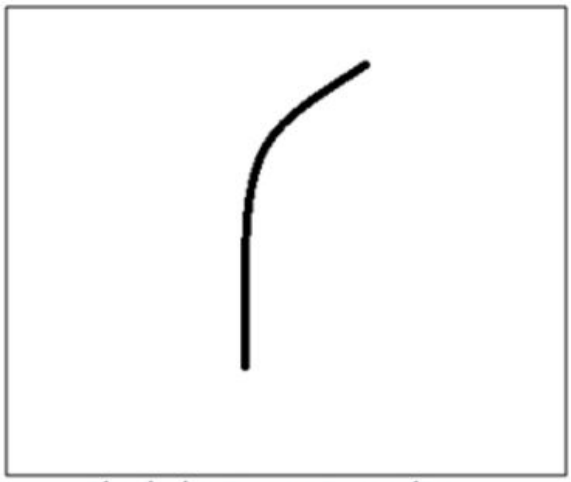


Fig. 6. Convolutional filter convolving around input volume to produce activation map [10]

Each of these filters act as feature identifiers which can be used to depict features edges, straight lines, simple colors, and curves. Imagine a fundamental characteristic that all images have in common with each other. Let the first filter be a curve detector of size 7 x 7 x 3 as shown in Fig. 7. For this curve detector, the filter will have a pixel structure in which there will be higher numerical values along the points that represent a shape of a curve.

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a Curve detector fitter

Fig. 7. Filter weights for detecting a curve [10]

3.3 Convolutional Network Structure

CNN [7] consists of multiple layers of filters. These small collection of neurons with their respective receptive fields process portions of the input image. The outputs of these neurons are then combined so that their input regions overlap with each other, to obtain a higher level representation of the input image. This process is iterated for every layer (Appendix B).

In a classic CNN architecture, apart from the convolutional layers, there are many other layers like the pooling layers and the Rectified Linear Units (ReLU) to account for the nonlinearities in the image and preserve the dimension. These layers help to improve the robustness of the network and control overfitting. A traditional CNN architecture would look like as shown in Fig. 8.

Fig. 8. Typical Structure of a Convolutional Neural Network

Now that the high level features are detected, a fully connected layer is connected to the end of the network. The function of the fully connected layer basically is to take an input volume (i.e. the output of the layer preceding it) and output an N dimensional vector, where N denotes the number of classes for that particular program. For example, for digit classification program, N would be 10 since there are 10 digits.

At the end of the CNN, the error is determined using a triplet loss function (depicted in Fig. 9 below). This error is then back propagated and the weights are updated accordingly. A triplet consists of three images:

1. An anchor image ***a***
2. A positive image of the same person ***p***
3. And a negative image of a different person ***n***.

To achieve the clustering, the distance between the anchor and positive should be less than the distance between the anchor and negative.

The training of CNN is done in the same way as that of feedforward neural network. Backpropagation is used to propagate the calculated triplet loss backwards and then weight updation is done for each filter. Here weights refer to the values associated with filters. These filters are same for each activation output but they are different for different layers.

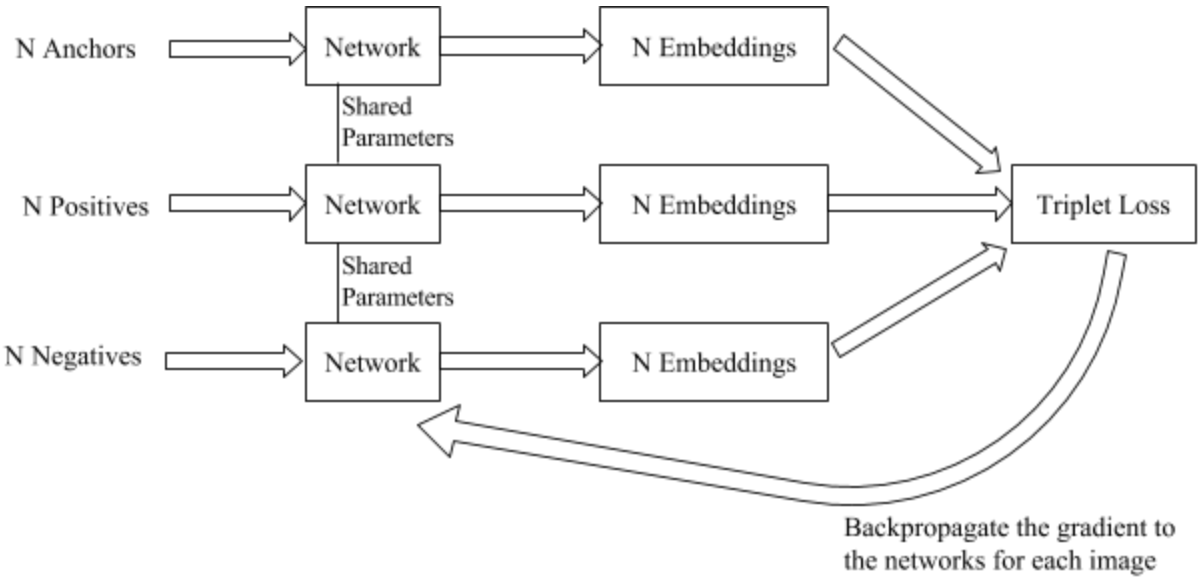


Fig. 9. Error calculation for CNN

3.4 Representation of Face

After feature extraction by CNN, a 128 dimensional feature vector is obtained which represents important information about the face. This information can be used at the later stage to classify the image. One example of feature representation can be given by the Fig. 10.

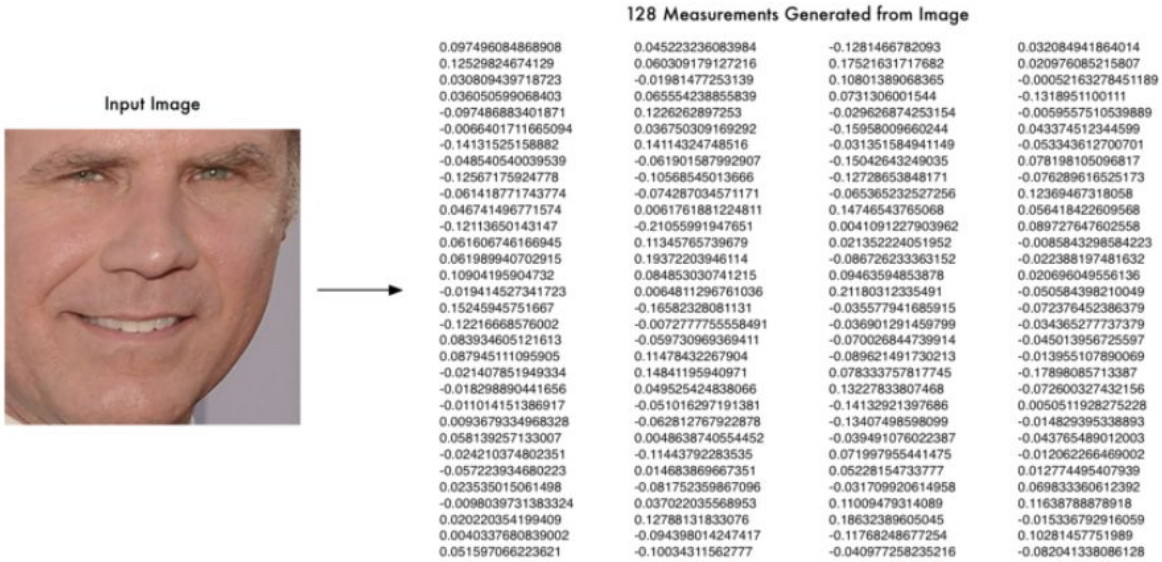


Fig. 10. Representation of a face after feature extraction [9]

3.5 Summary

The input for CNN is the 96x96 image provided by face detection. This image is an RGB image and this image is converted to a 128 dimensional array since feature extraction is being done here. These 128 numbers represent important information about the image.

Chapter 4

Classification of Faces

4.1 Introduction

Face recognition is a multi-class classification problem. In machine learning, multiclass or multinomial classification is the problem of classifying instances into one of the more than two classes. It can be categorized into One vs Rest and One vs One. The techniques employed here use the One vs Rest method to classify the images.

4.2 Classification Techniques Employed

Three algorithms are employed for classification in the proposed work. Support vector machine was selected since it is a large margin classifiers and provides best results. After much literature survey [1-7], three different kernels were selected which were able to provide desirable results. Other complex forms of SVMs were not considered since the time consumed for classification would increase which would make it undesirable for real time classification.

The SVM kernel results were also compared with decision tree and naive bayes classifier since these two were considered as benchmark models. The literature survey [1-7] indicated that these two algorithms were able to provide decent results in the past for the problem of support vector machines and if SVM provide better results than these algorithms, then efficient results can be obtained which are comparable to the state of the art methods currently being employed.

4.3 Support Vector Machines [11]

Support vector machines (SVMs) are supervised learning models that are used extensively for classification and regression analysis. SVM defines the decision boundary using the concept of decision hyperplanes in a multidimensional space. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier [11]. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is

as wide as possible.The SVM is a decision machine and so does not provide posterior probabilities [11].

SVM performs classification on the problems of the form given in (1),

$$y(x) = w^T.\ \phi(x) + b \tag{1}$$

where $\varphi(x)$ denotes the kernel transformation to transform data to feature space, and b is the bias parameter. The training data set comprises N input vectors $x_1, \ldots, x_N$ in an N dimensional space, with their corresponding target values $t_1, \ldots, t_N$ where $t_i \in \{-1, 1\}$, and new data points x are classified depending on the sign of $y(x)$. The support vector machines are maximal margin classifiers, i.e. the decision boundary corresponding to which the margin is maximized is selected.

The optimization problem thus reduces to maximising $1/w$, where w is the weight matrix, which is equivalent to minimizing $||w||^2$, and have to solve the problem given in (2)

$$\arg\min_{w,b} \frac{1}{2}\left||w|\right|^2 \tag{2}$$

subject to certain constraints. In order to solve this constrained optimization problem, we introduce Lagrange multipliers $a_n \geq 0$, with one multiplier a_n for each of the constraints giving the Lagrangian function

$$L(w, b, a) = \frac{1}{2}\left||w|\right|^2 - \sum_{n=1}^{N} a_n \{t_n\ (w^T \varphi(x_n) + b) - 1\} \tag{3}$$

where $a = (a_1, \ldots, a_N)^T$.

Subsequent problem is solved using Karush-Kuhn-Tucker (KKT) conditions [11].

Many linear parametric models can be changed into an equivalent 'dual representation' in which the predictions are done using linear combinations of a kernel function. The concept of a kernel formulated as an inner product in a feature space allows us to build interesting extensions of many well-known algorithms by making use of the kernel trick, also known as kernel substitution [12]. The general idea is that, for an algorithm designed in such a way that the input vector x is entered solely in the form of scalar products, then we can substitute that scalar product with some other choice of kernel. Three different SVM techniques were used for classification and their results were compared with that of Naive Bayes and Decision trees.

4.3.1 Linear SVM [11]

Linear SVM is the basic SVM where the input feature vector is not pre-processed before computation. A linear kernel for given feature vectors *f(u)* and *f(v)* is given in (4):

$$K(u,v) = f(u).f(v) \tag{4}$$

4.3.2 Grid Search SVM [13]

The grid search method provided by GridSearchCV exhaustively generates sample candidates from a parameter value grid specified with the param_grid parameter [13]. For instance, the following param_grid (in python):

```
param_grid = [{'C': [1, 10, 100, 1000], 'kernel': ['linear']},
              {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']}, ]
```

specifies that two grids should be considered: one with a linear kernel and C values in [1, 10,100, 1000], and the second one with an RBF kernel, and the cross-product of C values ranging in [1, 10, 100, 1000] and gamma values in [0.001, 0.0001].

4.3.3 Radial Basis Function Kernel SVM [11]

In machine learning, the (Gaussian) radial basis function kernel, or RBF kernel, is widely used in several kernelized learning algorithms. Particularly, it is commonly used in support vector machine classification. The kernel is given by the following equation where σ is a free parameter:

$$K(x, x') = \exp\left(-\frac{\left||x - x'|\right|^2}{2\sigma^2}\right) \tag{5}$$

4.4 Naive Bayes Classifier [12]

The Naive Bayes Classifier is a classifier which is based on statistical probability and uses Bayes' theorem for classification purposes. It applies to learning tasks where each instance ***x*** is described as a conjunction of attribute values and where the target function ***f(x)*** can take on any value a certain some finite set [12]. A set of training examples of the target function is provided, and a new instance is presented, described by the tuple of attribute values $(a_1, a_2 .. .a_n)$. The learner is asked to predict the target value, or classification, for this new instance. The Naive Bayes Learner is trained based on the Bayes Theorem given by (6):

$$P(A|B) = \frac{P(B|A).P(A)}{P(B)} \quad (6)$$

where *P(A)* is the probability of occurrence of event *A*
P(B) is the probability of occurrence of event *B*
P(A|B) is the probability of occurrence of event *A* given *B* has already occurred
P(B|A) is the probability of occurrence of event *B* given *A* has already occurred

4.5 Decision Trees [12]

This method approximates discretized target functions and the learned function is represented by a decision tree [12]. Learned trees can also be re-represented as cascading if-then statements to improve human readability. A decision tree is used as a predictive model which maps observations about an item represented in the branches to conclusions about the item's target value which are represented in the leaves [12].

A decision trees classifies a particular instance by sorting them down the tree from the root to some leaf node, leading to its final classification. Every node in the tree represents an if-then statement, and each branch descending from that node corresponds to one of the possible outcomes that can be achieved from this attribute [12]. An instance is classified by starting at the root of the tree, and progressively comparing each if-then comparison till we reach the leaf. One of the models for decision trees is shown in Fig. 11. Similar reasoning is used here.

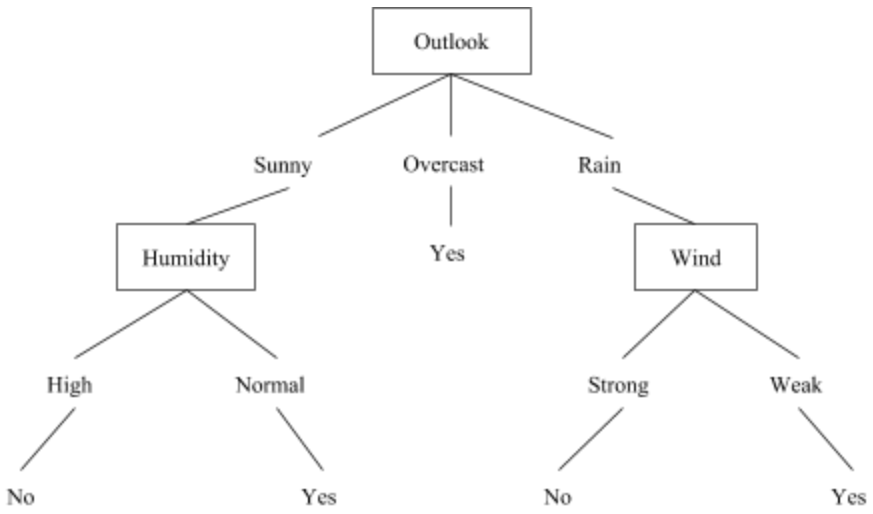


Fig. 11. Decision tree example [12]

4.6 Summary

Different classification techniques were introduced in this chapter. The script for classification of faces was written in python and the algorithms were imported in the script. The scripts for LFW experiment and real time classification were different since LFW experiment was a theoretical implementation to determine performance and real time implementation was a practical one.

The parameters for radial svm were C=1 for the arguments required in python method. Decision trees had a maximum depth of 20 and as a result pre-pruning was employed here. Naive Bayes algorithm does not require any parameters in python implementation and grid search SVM parameters have already been mentioned in the chapter 7.

The 128 dimensional vector serve as input features for decision trees such that the classification is done based on the individual pixel values. The feature selection is done using ID3 algorithm and depth is limited to 20 so as to avoid overfitting since decision trees are very prone to overfitting.

Chapter 5

Development of Web Application and Dataset Selection

5.1 Introduction

For the purpose of automating the attendance system, a real time application of the face recognition was needed. This led to the development of web application. The web application was developed using ***python*** and deployed using css and javascript. Also the deep learning framework requires large datasets for training. So, a large dataset was compiled and then testing was done on LFW dataset (Appendix A).

5.2 Development of Web application

Since a real time implementation of the proposed method was required, a web application was developed. The trained CNN model was already obtained by training it on the combined dataset of 500,000 images from CASIA and FaceScrub databases. The classifier was trained using real time images obtained by the frame by frame breakdown of live video recording. The modules written for preprocessing of images were provided in the code written for the application. The code for the application was written in python and the styling for the webpage of the application was done in CSS and HTML. After this the directory path for the modules was provided in the concerned arguments and the python code was hosted on local machine which can be transferred to a server at later stages. The application can be accessed by running the web address on the localhost.

Various steps involved in running the application are mentioned below:

1. Provide the name of the subject for training. Otherwise the application will display the person as unknown.
2. After addition of the subject, turn on the training button. Different frames will be obtained from the video feed. These frames will be used to train the current classifier.
3. Now the training can be turned off. After this the classifier will be saved which can later be trained again so as to add new subjects.

4. After addition of multiple subjects, the subjects can again appear in the video frame so that they can be identified. The entered name would appear on the frame around the face.

The labels for the identified subjects can be saved in a directory which can compile the attendance for the day. In this way, the objective for automating the attendance process can be easily obtained.

5.3 Training Dataset Compilation

Training a Deep Learning Network requires lots of data if efficient results are to be obtained. Most of the face recognition datasets are private and under the ownership of multinational corporations like Facebook and Google. The biggest publicly available datasets are CASIA and FaceScrub [7]. These two datasets have been combined to obtain 500,000 training images. The face recognition systems employed by Google and Facebook are trained on millions of images and therefore they are considered as the state of the art methods. The architecture developed here is based on the architecture of GoogleNet [5] but the number of layers have been reduced since the training dataset available is less as compared to that of Google. Still the proposed network has been able to provide adequate results even with reduced number of parameters.

5.4 LFW Dataset

The "Labeled Faces in the Wild" (LFW) image collection is a database of labeled, face images intended for studying face recognition in unconstrained images [7]. LFW is the de facto benchmark database for FRUE. Most existing CNNs train their networks on private databases and test the trained models on LFW.

The experiment setting can be explained as : Person i corresponds to the person in the LFW with the i^{th} most images. Then, 20 images are sampled from the first N people. If 20 images aren't provided of a person, all of their images are used. Next, the sampled data is split randomly ten times by putting 90% of the images in the training set and 10% of the images in the testing set. The random number seed should be initialized to the same value for sampling different experiments. A classifier is trained on the training set and the accuracy is obtained from predicting the identities in the testing set. The setup is explained through Fig. 12 below.

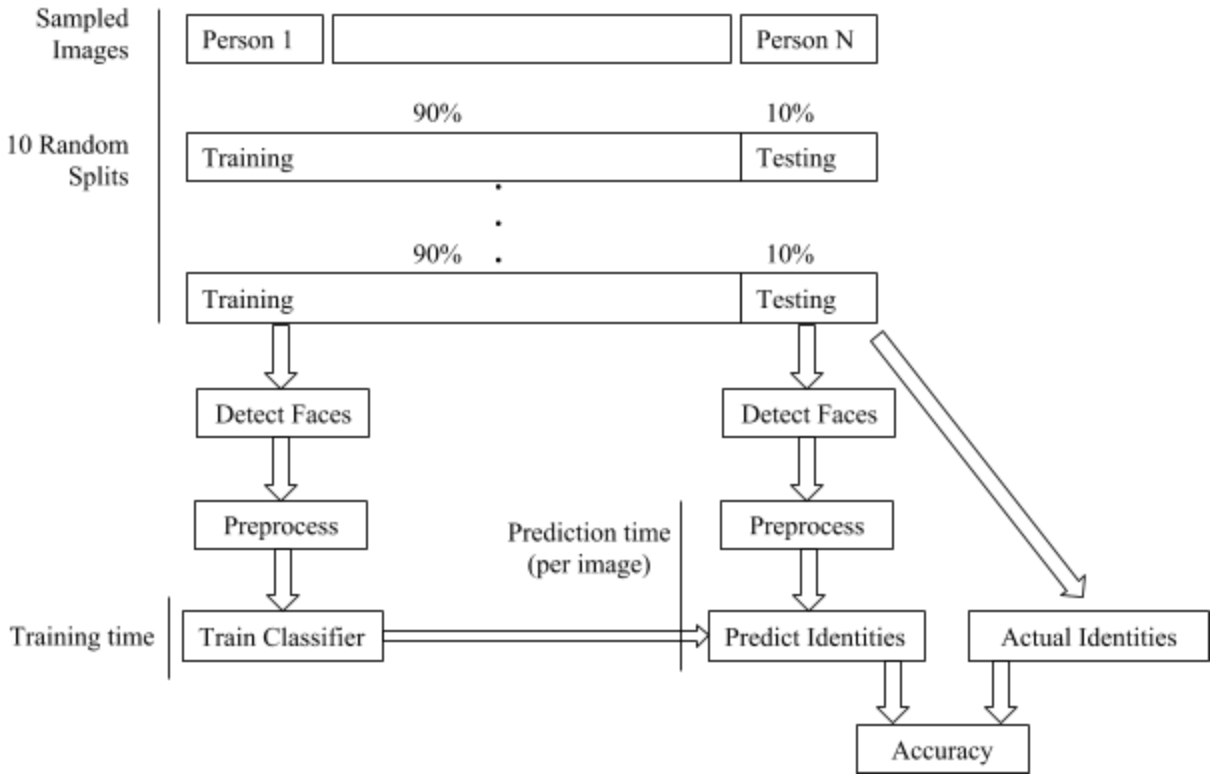


Fig. 12. LFW experiment

5.5 IIT Roorkee Students' Database

In addition to the LFW dataset, a dataset based on the students of IIT Roorkee was created using a web crawler. This method was used to obtain 10 images for each subject through their facebook profiles and 33 subjects were selected for this experiments. The remaining 67 subjects were considered from the LFW dataset for one of the experiments. The LFW experiment was performed on combined dataset [7] and the results were discussed in chapter 7. Since automated attendance recording requires multiple face detections in case of a group photo, several group photos were also used. These group photos were passed through the face detection module and multiple test subjects were obtained. These multiple test subjects were then passed through the remaining modules and the classification for each subject was obtained.

5.6 Summary

The input to the web application was a continuous video feed out of which frames were used for training. A 128 dimensional vector was provided to the classifier in testing phase of LFW experiment and output was one of the 100 classes to which it may belong.

Chapter 6

Methodology

6.1 Introduction

This chapter connects the chapter 2, 3, 4 and 5 and provides an overall understanding of how the whole automated attendance system works. The benchmark models have been explained in this section and the flow of the system has also been summarised.

6.2 Block Diagram and Flowchart

The conventional face recognition pipeline consists of four stages: face detection, face alignment, feature extraction (or face representation) and classification. Perhaps the single most important stage is feature extraction. The pipeline is represented as in figure 13:

Fig. 13. Block Diagram for face recognition

The flowchart for the proposed method can be given by figure 14:

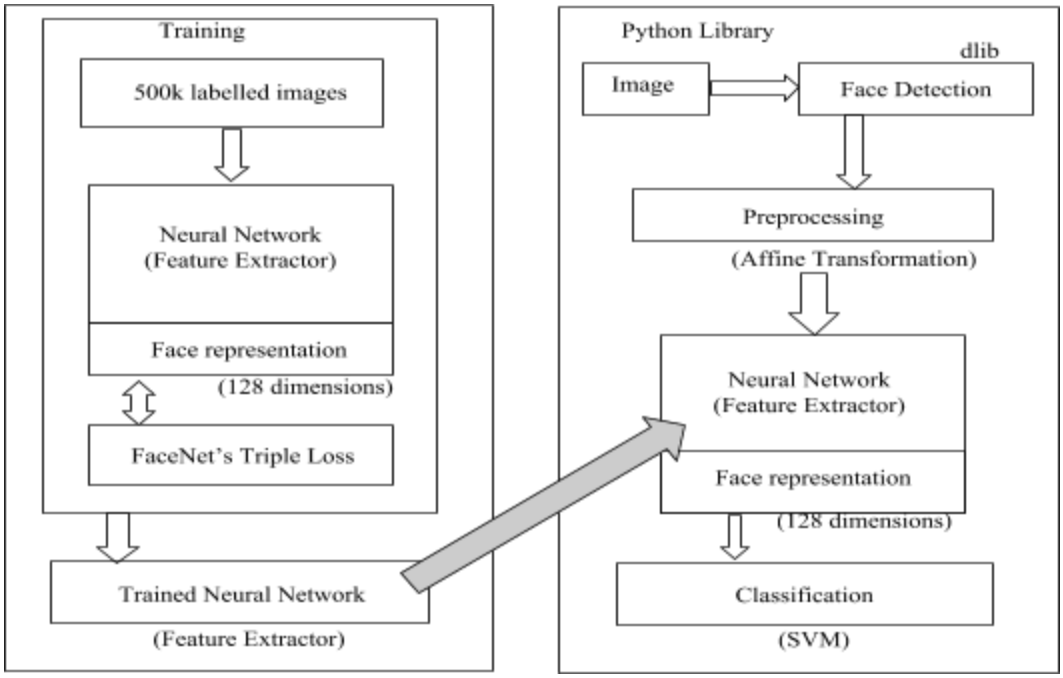


Fig. 14. Flow chart of the proposed method

6.3 Benchmark Models

Three different benchmark models were considered so that the performance of the proposed can be compared with these methods. The three benchmark methods are the standard techniques used for face recognition and therefore they can provide a good sense of how the proposed method performs in the problem of face detection and face recognition.

6.3.1 Eigenfaces

The Eigenface is one of the earliest successful methods of face recognition. This method uses Principal Component Analysis (PCA) for dimensionality reduction by linearly projecting the image space to a lower dimensional feature space.

6.3.2 Fisherfaces

To compute the Fisherfaces, an assumption is made on the normal distribution of the data in each class. This method upgrades the PCA method used in Eigenface and replaces it with an enhanced Linear Discriminant Analysis (LDA) called Fisher's Linear Discriminant Analysis (FLDA) for dimensionality reduction. This method maximizes the ratio of between-class scatter to that of the within-class scatter. Hence, this method achieves better results than PCA in discrimination or classification problems. The Fisherface is especially useful when facial images have large variations in facial expression and illumination.

6.3.3 Local Binary Pattern Histograms (LBPH)

In LBPH [8], face area is divided into small square regions from which LBP histograms are extracted using the LBP operator as shown in Fig. 15 and concatenated into enhanced feature histogram, thereby providing an efficient representation of the face.

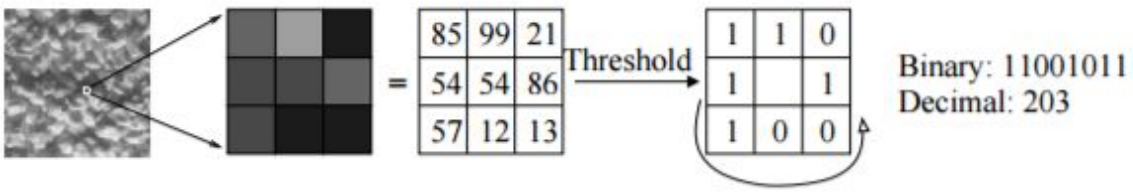


Fig. 15. Basic LBP operator [6]

6.4 Summary

The face detection was done using dlib which is a face recognition library written in C++. After obtaining the detected face, the alignment is performed using OpenCV where implementation of affine transformation is provided.

The CNN architecture has been implemented in Torch which is a deep learning framework written in lua. This network is trained using publicly available datasets and the weights values are stored for future use.

The classification script has been written in python where all the classification models are imported and results have been obtained. The eigenfaces, fisherfaces and LBPH implementation is provided in OpenCV and same has been used for benchmark purposes.

Chapter 7

Results and Discussions

7.1 Introduction

Different results have been discussed here with the experiments being conducted on the standard LFW dataset and custom dataset. Also confidence measure for some of the images have been produced so as to prove that a correct implementation of the algorithm has been obtained. Then the working of the web application has been explained.

7.2 Accuracy on the LFW Dataset

Dataset consists of around 13000 images of about 5000 people. Out of these 100 people with the most number of images were selected and a maximum of 20 images were selected for each individual. The data was divided into 9:1 ratio with 90 percent for training and 10 percent for testing [8]. Around 10 experiments were completed with random splitting being done in each case and the mean of the accuracies and training time was considered (Appendix A). Accuracy is calculated using (7).

Accuracy (ratio) = No. of Correctly classified images / No. of Total images (7)

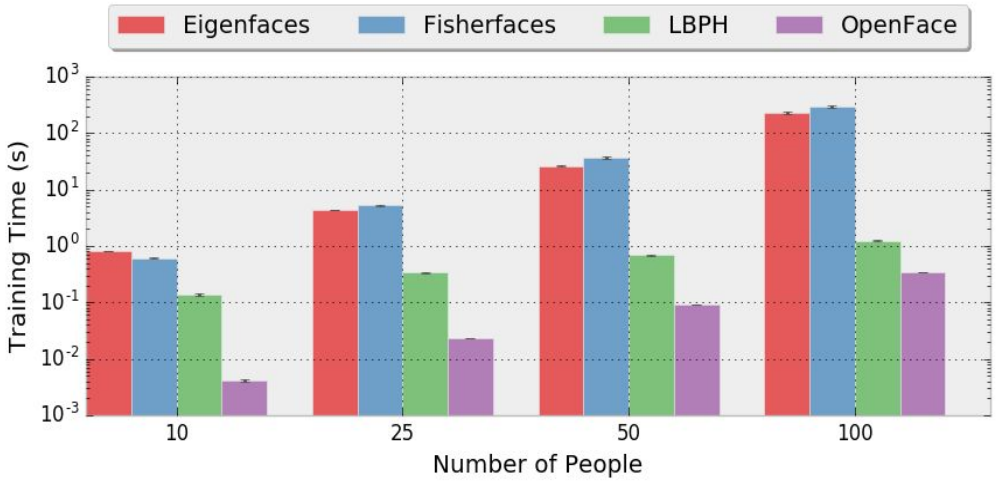


Fig 16. Comparison of the training times (LFW dataset)

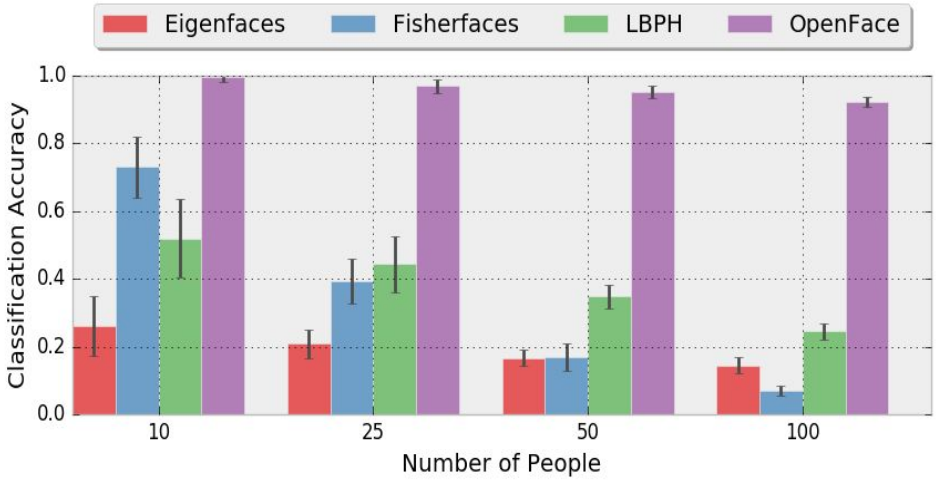


Fig 17. Comparison of the classification accuracy(LFW dataset)

7.3 Accuracy on the custom dataset

This dataset contains face images of 33 students of IIT Roorkee and the experiment is performed in the similar manner as explained in 8.1 and Appendix A.

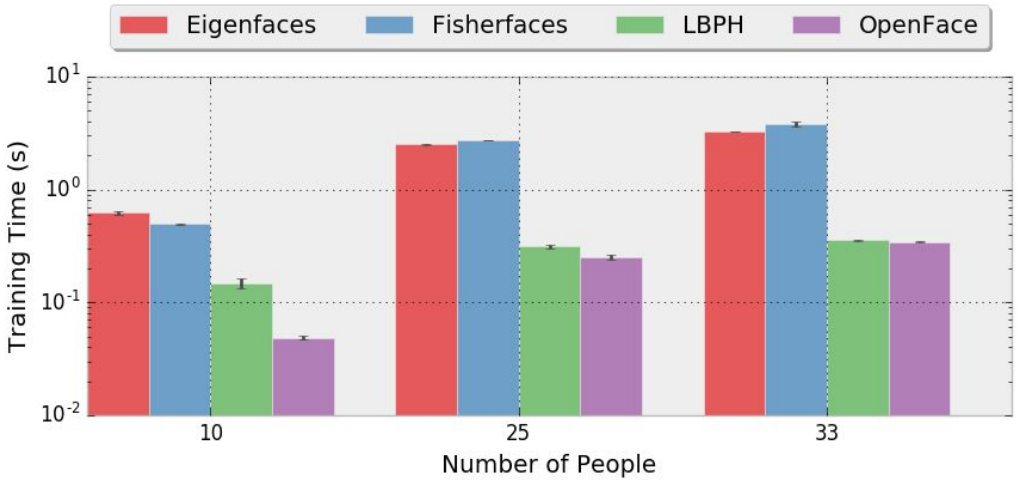


Fig 18. Comparison of training times for each method (custom dataset)

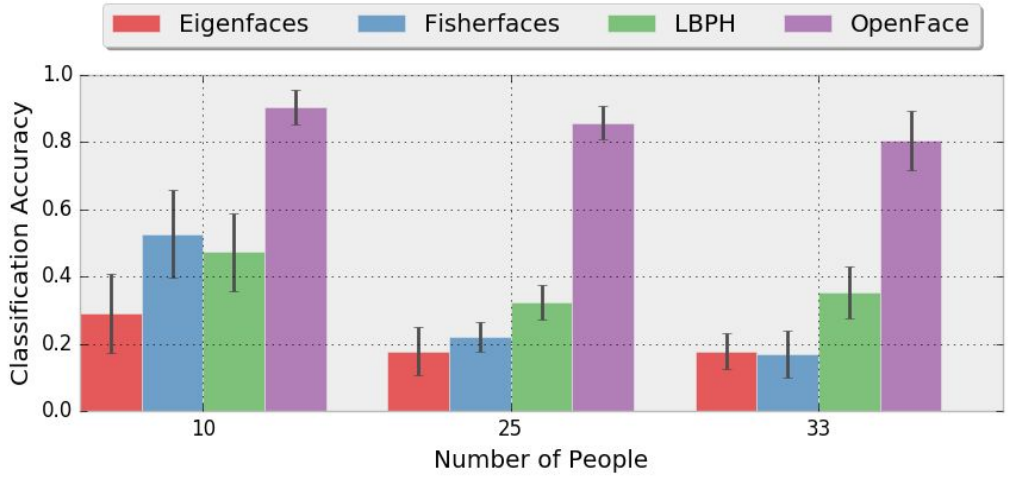


Fig 19. Comparison of the classification accuracy (custom dataset)

7.4 Accuracy on the combined dataset

A total of 100 people were considered in this dataset. We took all the 33 people of our custom dataset and 67 people from the LFW dataset performing the same experiment as explained in 8.1.

Table-I. A comparison of the accuracies (in %) of the different classifiers

Index	No. of people / Methods	10	25	50	100
1a	**Linear SVM**	99.5	97.4	95.1	90.8
1b	**RBF SVM**	99.5	97.8	95.4	**91.2**
1c	**Grid search**	100	97.4	95.1	90.9
2	**Decision trees**	90.5	72	50.4	38.1
3	**Naive Bayes**	99.5	97.4	95.1	89.7

The RBF SVM was chosen for future classification purposes as the best accuracies were obtained overall using this method for a reasonable number of subjects. Also it was observed that decision trees were not a good candidate for classification. Gridsearch SVM performed similar to Linear SVM and thus extra computations associated with the gridsearch method can be avoided.

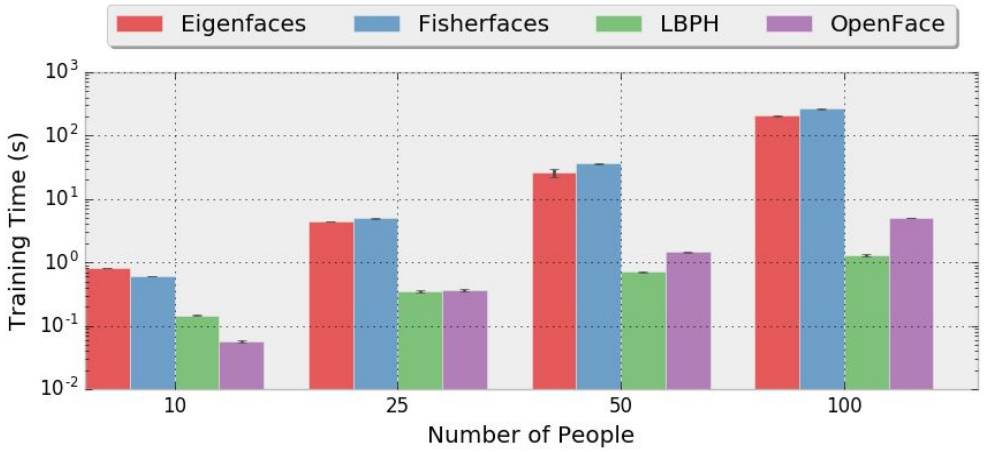


Fig 20. Comparison of the training times (combined dataset)

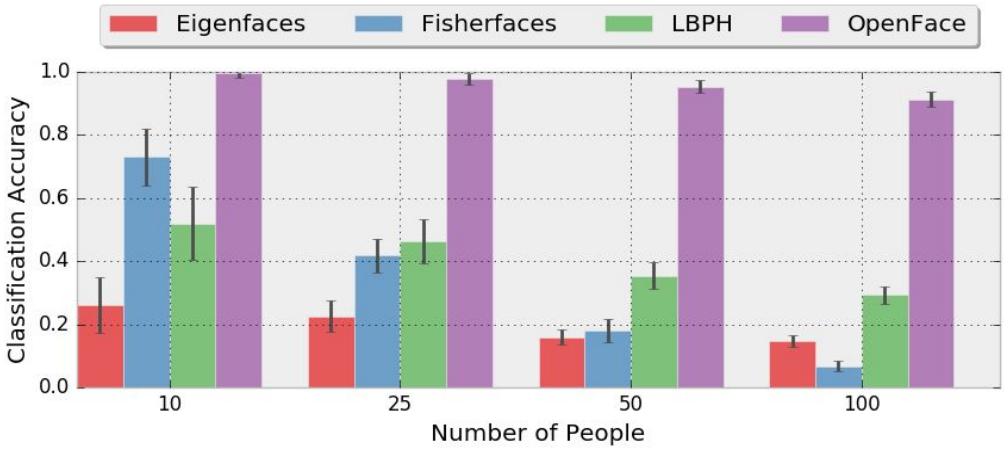


Fig 21. Comparison of the classification accuracies (combined dataset)

7.5 Face detection for images containing single and multiple faces

- Single face detection (Appendix C)

Test image Classified image

Fig 22. George W Bush was predicted with **0.99** confidence

Remarks - George W Bush has more than 500 photos in this instance

Test image Classified image

Fig 23. George W Bush was predicted with **0.32** confidence

Remarks - George W Bush has 20 photos in this instance

- Multiple faces detection

Test image Classified images

Fig 24. **Chitra** @x=646 (**0.51% confidence)** and **Shruti** @x=1135 were predicted (**0.40% confidence**)

[Note: @x denotes the position of image when going from left to right]

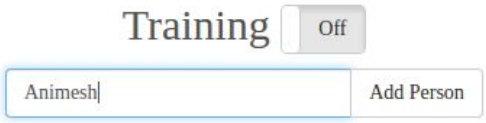
7.6 Demo of Web Application

Step 1: Enter the name of the subject whose image is to be included in the training dataset. Since we want to predict multiple subjects in a single image, we will enter the names of two subjects here: Animesh and Angad.

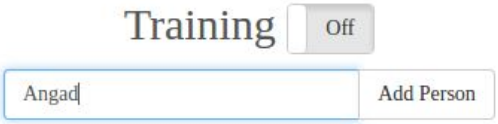
Automatic attendance marking using face recognition

Official Web application

EEN-400B: Group 2



(a) Animesh

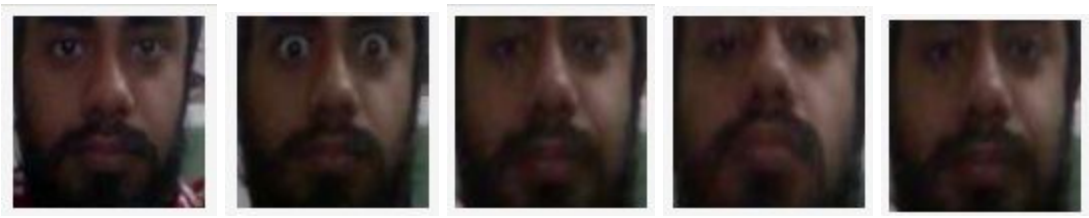


(b) Angad

Fig 25. Addition of subjects

Step 2: The training is made on for both the subjects one by one so that the classifier can be trained on the subjects.

(a) Some of the training images for Animesh



(b) Some of the training images for Angad

Fig 26. Training on the obtained images

Step 3: After the trained classifier has been saved, the subjects can reappear in the frame so that their faces can be predicted. The labels entered for both the subjects would appear on top of the bounding boxes around their faces.

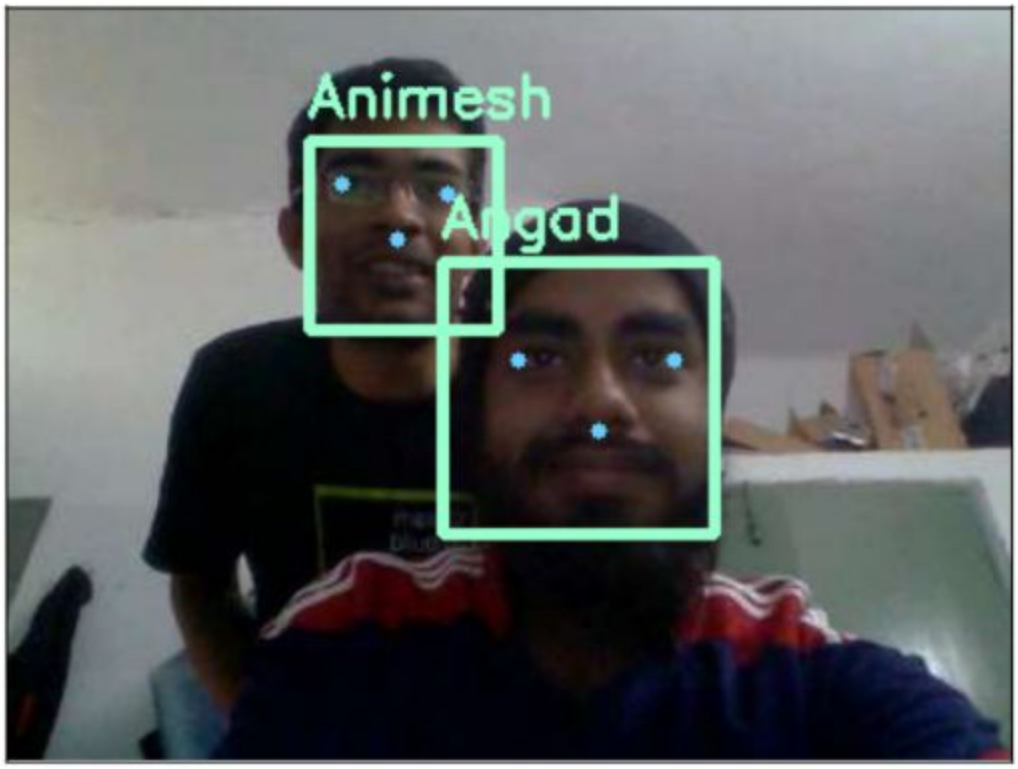


Fig 27. Testing of web application

Chapter 8

Conclusions and Future Work

8.1 Conclusions

- The accuracy of the faces predicted in case of both single and multiple face cases are fairly satisfactory. However, the confidence percentage of the detection increases as the number of sample images of the test subject increases.
- Among the classification techniques considered in the project, best accuracy was obtained with the help of RBF kernel in SVM method with 91% accuracy in LFW accuracy. This can confirm that classification can be done with adequate accuracy.
- A web application has been prepared for real time face recognition which can be used for automated attendance system efficiently.

8.2 Future Work

- Different CNN frameworks can be explored so as to get an improved performance with respect to feature extraction.
- Also more classification techniques and face detection techniques can be experimented with in order to get better performance. The main aim is to get improved performance on detection of multiple faces in a photo.
- A mobile application can also be prepared so as to easily operate the automatic attendance system and make it portable.

References

[1] Takeo Kanade, “Picture processing system by computer complex and recognition of human faces,” Doctoral dissertation, Kyoto University, pp. 83–97, 1973.

[2] Matthew Turk and Alex Pentland, “Eigenfaces for recognition,” Journal of cognitive neuro-science, vol. 3, no. 1, pp. 71–86, 1991.

[3] Peter N Belhumeur, Joa’o P Hespanha, and David J Kriegman, “Eigenfaces vs. fisherfaces: Recognition using class specific linear projection,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, no. 7, pp. 711–720, 1997.

[4] Yaniv Taigman, Ming Yang, Marc’Aurelio Ranzato, and Lior Wolf. Deepface, “Closing the gap to human-level performance in face verification,” 2014 IEEE Conference on In Computer Vision and Pattern Recognition (CVPR), Columbus, USA, pp. 1701–1708, 2014.

[5] Florian Schroff, Dmitry Kalenichenko, and James Philbin, “Facenet: A unified embedding for face recognition and clustering,” In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 815–823, 2015.

[6] Yoshua Bengio, Ian J. Goodfellow, and Aaron Courville, “Deep learning,” MIT Press, 2015.

[7] B. Amos, B. Ludwiczuk, M. Satyanarayanan, "Openface: A general-purpose face recognition library with mobile applications," CMU-CS-16-118, CMU School of Computer Science, Tech. Rep., 2016.

[8] Timo Ahonen, Abdenour Hadid, and Matti Pietikainen, “Face recognition with local binary patterns,” In Computer vision-eccv 2004, pages 469–481. Springer, 2004.

[9] Available:https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning

[10] Available: https://adeshpande3.github.io/adeshpande3.github.io

[11] Christopher Bishop, “Pattern Recognition and Machine Learning,” Springer, 1st edition, 2007.

[12] Tom Mitchell, “Machine Learning,” McGraw-Hill, 1st edition, 1997.

[13] Jianfeng Wang and Lei Zhang, "A parameter optimization method for an SVM based on improved grid search algorithm," Applied Science and Technology, vol. 39, no. 3, 2012.

Appendix

A: Program for LFW experiment

```
import cv2
import numpy as np
import pandas as pd

from sklearn.svm import SVC
from sklearn.grid_search import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.cross_validation import ShuffleSplit
from sklearn.metrics import accuracy_score

import matplotlib as mpl
mpl.use('Agg')
import matplotlib.pyplot as plt
plt.style.use('bmh')

import operator
import os
import pickle
import sys
import time

import argparse

import openface

sys.path.append("..")
from openface.helper import mkdirP

fileDir = os.path.dirname(os.path.realpath(__file__))
modelDir = os.path.join(fileDir, '..', 'models')
dlibModelDir = os.path.join(modelDir, 'dlib')
openfaceModelDir = os.path.join(modelDir, 'openface')

nPplVals = [10, 25, 50, 100]
nImgs = 20

cmap = plt.get_cmap("Set1")
colors = cmap(np.linspace(0, 0.5, 5))
alpha = 0.7


def main():
    parser = argparse.ArgumentParser()
    lfwDefault = os.path.expanduser("~/openface/data/lfw/aligned")
    parser.add_argument('--lfwAligned', type=str,
                        default=lfwDefault,
                        help='Location of aligned LFW images')
    parser.add_argument('--networkModel', type=str, help="Path to Torch network
model.",
                        default=os.path.join(openfaceModelDir, 'nn4.small2.v1.t7'))
    parser.add_argument('--largeFont', action='store_true')
    parser.add_argument('workDir', type=str,
                        help='The work directory where intermediate files and results
are kept.')
    args = parser.parse_args()
    # print(args)

    if args.largeFont:
        font = {'family': 'normal', 'size': 20}
        mpl.rc('font', **font)
```

```
    mkdirP(args.workDir)

    print("Getting lfwPpl")
    lfwPplCache = os.path.join(args.workDir, 'lfwPpl.pkl')
    lfwPpl = cacheToFile(lfwPplCache)(getLfwPplSorted)(args.lfwAligned)

    print("Eigenfaces Experiment")
    cls = cv2.createEigenFaceRecognizer()
    cache = os.path.join(args.workDir, 'eigenFacesExp.pkl')
    eigenFacesDf = cacheToFile(cache)(opencvExp)(lfwPpl, cls)

    print("Fisherfaces Experiment")
    cls = cv2.createFisherFaceRecognizer()
    cache = os.path.join(args.workDir, 'fisherFacesExp.pkl')
    fishFacesDf = cacheToFile(cache)(opencvExp)(lfwPpl, cls)

    print("LBPH Experiment")
    cls = cv2.createLBPHFaceRecognizer()
    cache = os.path.join(args.workDir, 'lbphExp.pkl')
    lbphFacesDf = cacheToFile(cache)(opencvExp)(lfwPpl, cls)

    print("OpenFace CPU/SVM Experiment")
    net = openface.TorchNeuralNet(args.networkModel, 96, cuda=False)
    #cls = SVC(kernel='linear', C=1, probability=True)

    #param_grid = [
    #       {'C': [1, 10, 100, 1000],
    #         'kernel': ['linear']},
    #        {'C': [1, 10, 100, 1000],
    #         'gamma': [0.001, 0.0001],
    #         'kernel': ['rbf']}
    #    ]
    #cls = GridSearchCV(SVC(C=1, probability=True), param_grid, cv=5)

    cls = SVC(C=1, kernel='rbf', probability=True, gamma=2)

    #cls = DecisionTreeClassifier(max_depth=20)

    #cls = GaussianNB()

    cache = os.path.join(args.workDir, 'openface.cpu.svm.pkl')
    openfaceCPUsvmDf = cacheToFile(cache)(openfaceExp)(lfwPpl, net, cls)

    print("OpenFace GPU/SVM Experiment")
    net = openface.TorchNeuralNet(args.networkModel, 96, cuda=True)
    cache = os.path.join(args.workDir, 'openface.gpu.svm.pkl')
    openfaceGPUsvmDf = cacheToFile(cache)(openfaceExp)(lfwPpl, net, cls)

    plotAccuracy(args.workDir, args.largeFont,
                 eigenFacesDf, fishFacesDf, lbphFacesDf,
                 openfaceCPUsvmDf, openfaceGPUsvmDf)
    plotTrainingTime(args.workDir, args.largeFont,
                     eigenFacesDf, fishFacesDf, lbphFacesDf,
                     openfaceCPUsvmDf, openfaceGPUsvmDf)
    plotPredictionTime(args.workDir, args.largeFont,
                       eigenFacesDf, fishFacesDf, lbphFacesDf,
                       openfaceCPUsvmDf, openfaceGPUsvmDf)

# http://stackoverflow.com/questions/16463582


def cacheToFile(file_name):
    def decorator(original_func):
        global cache
        try:
            cache = pickle.load(open(file_name, 'rb'))
        except:
            cache = None
```

```
        def new_func(*param):
            global cache
            if cache is None:
                cache = original_func(*param)
                pickle.dump(cache, open(file_name, 'wb'))
            return cache
        return new_func

    return decorator


def getLfwPplSorted(lfwAligned):
    lfwPpl = {}
    for person in os.listdir(lfwAligned):
        fullPath = os.path.join(lfwAligned, person)
        if os.path.isdir(fullPath):
            nFiles = len([item for item in os.listdir(fullPath)
                          if os.path.isfile(os.path.join(fullPath, item))])
            lfwPpl[fullPath] = nFiles
    return sorted(lfwPpl.items(), key=operator.itemgetter(1), reverse=True)


def getData(lfwPpl, nPpl, nImgs, mode):
    X, y = [], []

    personNum = 0
    for (person, nTotalImgs) in lfwPpl[:nPpl]:
        imgs = sorted(os.listdir(person))
        for imgPath in imgs[:nImgs]:
            imgPath = os.path.join(person, imgPath)
            img = cv2.imread(imgPath)
            img = cv2.resize(img, (96, 96))
            if mode == 'grayscale':
                img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            elif mode == 'rgb':
                img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            else:
                assert 0

            X.append(img)
            y.append(personNum)

        personNum += 1

    X = np.array(X)
    y = np.array(y)
    return (X, y)


def opencvExp(lfwAligned, cls):
    df = pd.DataFrame(columns=('nPpl', 'nImgs', 'trainTimeSecMean',
'trainTimeSecStd',
                               'predictTimeSecMean', 'predictTimeSecStd',
                               'accsMean', 'accsStd'))

    df_i = 0
    for nPpl in nPplVals:
        print(" + nPpl: {}".format(nPpl))
        (X, y) = getData(lfwAligned, nPpl, nImgs, mode='grayscale')
        nSampled = X.shape[0]
        ss = ShuffleSplit(nSampled, n_iter=10, test_size=0.1, random_state=0)

        allTrainTimeSec = []
        allPredictTimeSec = []
        accs = []

        for train, test in ss:
            start = time.time()
            cls.train(X[train], y[train])
            trainTimeSec = time.time() - start
```

```
            allTrainTimeSec.append(trainTimeSec)

            y_predict = []
            for img in X[test]:
                start = time.time()
                (label, score) = cls.predict(img)
                y_predict.append(label)
                predictTimeSec = time.time() - start
                allPredictTimeSec.append(predictTimeSec)
            y_predict = np.array(y_predict)

            acc = accuracy_score(y[test], y_predict)
            accs.append(acc)

        df.loc[df_i] = [nPpl, nImgs,
                        np.mean(allTrainTimeSec), np.std(allTrainTimeSec),
                        np.mean(allPredictTimeSec), np.std(allPredictTimeSec),
                        np.mean(accs), np.std(accs)]
        print(" Accuracy: {}".format(np.mean(accs)))
        df_i += 1

    return df


def openfaceExp(lfwAligned, net, cls):
    df = pd.DataFrame(columns=('nPpl', 'nImgs',
                               'trainTimeSecMean', 'trainTimeSecStd',
                               'predictTimeSecMean', 'predictTimeSecStd',
                               'accsMean', 'accsStd'))

    repCache = {}

    df_i = 0
    for nPpl in nPplVals:
        print(" + nPpl: {}".format(nPpl))
        (X, y) = getData(lfwAligned, nPpl, nImgs, mode='rgb')
        nSampled = X.shape[0]
        ss = ShuffleSplit(nSampled, n_iter=10, test_size=0.1, random_state=0)

        allTrainTimeSec = []
        allPredictTimeSec = []
        accs = []

        for train, test in ss:
            X_train = []
            for img in X[train]:
                h = hash(str(img.data))
                if h in repCache:
                    rep = repCache[h]
                else:
                    rep = net.forward(img)
                    repCache[h] = rep
                X_train.append(rep)

            start = time.time()
            X_train = np.array(X_train)
            cls.fit(X_train, y[train])
            trainTimeSec = time.time() - start
            allTrainTimeSec.append(trainTimeSec)

            start = time.time()
            X_test = []
            for img in X[test]:
                X_test.append(net.forward(img))
            y_predict = cls.predict(X_test)
            predictTimeSec = time.time() - start
            allPredictTimeSec.append(predictTimeSec / len(test))
            y_predict = np.array(y_predict)

            acc = accuracy_score(y[test], y_predict)
```

```
            accs.append(acc)

        df.loc[df_i] = [nPpl, nImgs,
                        np.mean(allTrainTimeSec), np.std(allTrainTimeSec),
                        np.mean(allPredictTimeSec), np.std(allPredictTimeSec),
                        np.mean(accs), np.std(accs)]
        print(" Accuracy: {}".format(np.mean(accs)))
        df_i += 1

    return df


def plotAccuracy(workDir, largeFont, eigenFacesDf, fishFacesDf, lbphFacesDf,
                 openfaceCPUsvmDf, openfaceGPUsvmDf):
    indices = eigenFacesDf.index.values
    barWidth = 0.2

    if largeFont:
        fig = plt.figure(figsize=(10, 5))
    else:
        fig = plt.figure(figsize=(10, 4))
    ax = fig.add_subplot(111)
    plt.bar(indices, eigenFacesDf['accsMean'], barWidth,
            yerr=eigenFacesDf['accsStd'], label='Eigenfaces',
            color=colors[0], ecolor='0.3', alpha=alpha)
    plt.bar(indices + barWidth, fishFacesDf['accsMean'], barWidth,
            yerr=fishFacesDf['accsStd'], label='Fisherfaces',
            color=colors[1], ecolor='0.3', alpha=alpha)
    plt.bar(indices + 2 * barWidth, lbphFacesDf['accsMean'], barWidth,
            yerr=lbphFacesDf['accsStd'], label='LBPH',
            color=colors[2], ecolor='0.3', alpha=alpha)
    plt.bar(indices + 3 * barWidth, openfaceCPUsvmDf['accsMean'], barWidth,
            yerr=openfaceCPUsvmDf['accsStd'], label='OpenFace',
            color=colors[3], ecolor='0.3', alpha=alpha)

    box = ax.get_position()
    if largeFont:
        ax.set_position([box.x0, box.y0 + 0.07, box.width, box.height * 0.83])
        plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1.25), ncol=4,
                   fancybox=True, shadow=True, fontsize=16)
    else:
        ax.set_position([box.x0, box.y0 + 0.05, box.width, box.height * 0.85])
        plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1.25), ncol=4,
                   fancybox=True, shadow=True)
    plt.ylabel("Classification Accuracy")
    plt.xlabel("Number of People")

    ax.set_xticks(indices + 2 * barWidth)
    xticks = []
    for nPpl in nPplVals:
        xticks.append(nPpl)
    ax.set_xticklabels(xticks)

    locs, labels = plt.xticks()
    plt.ylim(0, 1)
    plt.savefig(os.path.join(workDir, 'accuracies.png'))


def plotTrainingTime(workDir, largeFont, eigenFacesDf, fishFacesDf, lbphFacesDf,
                     openfaceCPUsvmDf, openfaceGPUsvmDf):
    indices = eigenFacesDf.index.values
    barWidth = 0.2

    fig = plt.figure(figsize=(10, 4))
    ax = fig.add_subplot(111)
    plt.bar(indices, eigenFacesDf['trainTimeSecMean'], barWidth,
            yerr=eigenFacesDf['trainTimeSecStd'], label='Eigenfaces',
            color=colors[0], ecolor='0.3', alpha=alpha)
    plt.bar(indices + barWidth, fishFacesDf['trainTimeSecMean'], barWidth,
            yerr=fishFacesDf['trainTimeSecStd'], label='Fisherfaces',
```

```
            color=colors[1], ecolor='0.3', alpha=alpha)
    plt.bar(indices + 2 * barWidth, lbphFacesDf['trainTimeSecMean'], barWidth,
            yerr=lbphFacesDf['trainTimeSecStd'], label='LBPH',
            color=colors[2], ecolor='0.3', alpha=alpha)
    plt.bar(indices + 3 * barWidth, openfaceCPUsvmDf['trainTimeSecMean'], barWidth,
            yerr=openfaceCPUsvmDf['trainTimeSecStd'],
            label='OpenFace',
            color=colors[3], ecolor='0.3', alpha=alpha)

    box = ax.get_position()
    if largeFont:
        ax.set_position([box.x0, box.y0 + 0.08, box.width, box.height * 0.83])
        plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1.27), ncol=4,
                   fancybox=True, shadow=True, fontsize=16)
    else:
        ax.set_position([box.x0, box.y0 + 0.05, box.width, box.height * 0.85])
        plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1.25), ncol=4,
                   fancybox=True, shadow=True)
    plt.ylabel("Training Time (s)")
    plt.xlabel("Number of People")

    ax.set_xticks(indices + 2 * barWidth)
    xticks = []
    for nPpl in nPplVals:
        xticks.append(nPpl)
    ax.set_xticklabels(xticks)
    locs, labels = plt.xticks()
    # plt.setp(labels, rotation=45)
    # plt.ylim(0, 1)

    ax.set_yscale('log')
    plt.savefig(os.path.join(workDir, 'trainTimes.png'))


def plotPredictionTime(workDir, largeFont, eigenFacesDf, fishFacesDf, lbphFacesDf,
                       openfaceCPUsvmDf, openfaceGPUsvmDf):
    indices = eigenFacesDf.index.values
    barWidth = 0.15

    fig = plt.figure(figsize=(10, 4))
    ax = fig.add_subplot(111)
    plt.bar(indices, eigenFacesDf['predictTimeSecMean'], barWidth,
            yerr=eigenFacesDf['predictTimeSecStd'], label='Eigenfaces',
            color=colors[0], ecolor='0.3', alpha=alpha)
    plt.bar(indices + barWidth, fishFacesDf['predictTimeSecMean'], barWidth,
            yerr=fishFacesDf['predictTimeSecStd'], label='Fisherfaces',
            color=colors[1], ecolor='0.3', alpha=alpha)
    plt.bar(indices + 2 * barWidth, lbphFacesDf['predictTimeSecMean'], barWidth,
            yerr=lbphFacesDf['predictTimeSecStd'], label='LBPH',
            color=colors[2], ecolor='0.3', alpha=alpha)
    plt.bar(indices + 3 * barWidth, openfaceCPUsvmDf['predictTimeSecMean'], barWidth,
            yerr=openfaceCPUsvmDf['predictTimeSecStd'],
            label='OpenFace CPU',
            color=colors[3], ecolor='0.3', alpha=alpha)
    plt.bar(indices + 4 * barWidth, openfaceGPUsvmDf['predictTimeSecMean'], barWidth,
            yerr=openfaceGPUsvmDf['predictTimeSecStd'],
            label='OpenFace GPU',
            color=colors[4], ecolor='0.3', alpha=alpha)

    box = ax.get_position()
    if largeFont:
        ax.set_position([box.x0, box.y0 + 0.11, box.width, box.height * 0.7])
        plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1.45), ncol=3,
                   fancybox=True, shadow=True, fontsize=16)
    else:
        ax.set_position([box.x0, box.y0 + 0.05, box.width, box.height * 0.77])
        plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1.37), ncol=3,
                   fancybox=True, shadow=True)
    plt.ylabel("Prediction Time (s)")
    plt.xlabel("Number of People")
```

```
    ax.set_xticks(indices + 2.5 * barWidth)
    xticks = []
    for nPpl in nPplVals:
        xticks.append(nPpl)
    ax.set_xticklabels(xticks)
    ax.xaxis.grid(False)
    locs, labels = plt.xticks()
    # plt.setp(labels, rotation=45)
    # plt.ylim(0, 1)

    ax.set_yscale('log')
    plt.savefig(os.path.join(workDir, 'predictTimes.png'))

if __name__ == '__main__':
    main()
```

B: Program for CNN architecture

```
imgDim = 96

function createModel()
   local net = nn.Sequential()

   net:add(nn.SpatialConvolutionMM(3, 64, 7, 7, 2, 2, 3, 3))
   net:add(nn.SpatialBatchNormalization(64))
   net:add(nn.ReLU())

   -- The FaceNet paper just says `norm` and that the models are based
   -- heavily on the inception paper (http://arxiv.org/pdf/1409.4842.pdf),
   -- which uses pooling and normalization in the same way in the early layers.
   --
   -- The Caffe and official versions of this network both use LRN:
   --
   --   + https://github.com/BVLC/caffe/tree/master/models/bvlc_googlenet
   --   + https://github.com/google/inception/blob/master/inception.ipynb
   --
   -- The Caffe docs at http://caffe.berkeleyvision.org/tutorial/layers.html
   -- define LRN to be across channels.
   net:add(nn.SpatialMaxPooling(3, 3, 2, 2, 1, 1))
   net:add(nn.SpatialCrossMapLRN(5, 0.0001, 0.75))

   -- Inception (2)
   net:add(nn.SpatialConvolutionMM(64, 64, 1, 1))
   net:add(nn.SpatialBatchNormalization(64))
   net:add(nn.ReLU())
   net:add(nn.SpatialConvolutionMM(64, 192, 3, 3, 1, 1, 1))
   net:add(nn.SpatialBatchNormalization(192))
   net:add(nn.ReLU())

   net:add(nn.SpatialCrossMapLRN(5, 0.0001, 0.75))
   net:add(nn.SpatialMaxPooling(3, 3, 2, 2, 1, 1))

   -- Inception (3a)
   net:add(nn.Inception{
     inputSize = 192,
     kernelSize = {3, 5},
     kernelStride = {1, 1},
     outputSize = {128, 32},
```

```
  reduceSize = {96, 16, 32, 64},
  pool = nn.SpatialMaxPooling(3, 3, 1, 1, 1, 1),
  batchNorm = true
})

-- Inception (3b)
net:add(nn.Inception{
  inputSize = 256,
  kernelSize = {3, 5},
  kernelStride = {1, 1},
  outputSize = {128, 64},
  reduceSize = {96, 32, 64, 64},
  pool = nn.SpatialLPPooling(256, 2, 3, 3, 1, 1),
  batchNorm = true
})

-- Inception (3c)
net:add(nn.Inception{
  inputSize = 320,
  kernelSize = {3, 5},
  kernelStride = {2, 2},
  outputSize = {256, 64},
  reduceSize = {128, 32, nil, nil},
  pool = nn.SpatialMaxPooling(3, 3, 2, 2, 1, 1),
  batchNorm = true
})

-- Inception (4a)
net:add(nn.Inception{
  inputSize = 640,
  kernelSize = {3, 5},
  kernelStride = {1, 1},
  outputSize = {192, 64},
  reduceSize = {96, 32, 128, 256},
  pool = nn.SpatialLPPooling(640, 2, 3, 3, 1, 1),
  batchNorm = true
})

-- Inception (4b)
net:add(nn.Inception{
  inputSize = 640,
  kernelSize = {3, 5},
  kernelStride = {1, 1},
  outputSize = {224, 64},
  reduceSize = {112, 32, 128, 224},
  pool = nn.SpatialLPPooling(640, 2, 3, 3, 1, 1),
  batchNorm = true
})

-- Inception (4e)
net:add(nn.Inception{
  inputSize = 640,
  kernelSize = {3, 5},
  kernelStride = {2, 2},
  outputSize = {256, 128},
  reduceSize = {160, 64, nil, nil},
  pool = nn.SpatialMaxPooling(3, 3, 2, 2, 1, 1),
```

```
      batchNorm = true
    })

    -- Inception (5a)
    net:add(nn.Inception{
                inputSize = 1024,
                kernelSize = {3},
                kernelStride = {1},
                outputSize = {384},
                reduceSize = {192, 128, 384},
                pool = nn.SpatialLPPooling(960, 2, 3, 3, 1, 1),
                batchNorm = true
    })
    -- Inception (5b)
    net:add(nn.Inception{
                inputSize = 896,
                kernelSize = {3},
                kernelStride = {1},
                outputSize = {384},
                reduceSize = {192, 128, 384},
                pool = nn.SpatialMaxPooling(3, 3, 1, 1, 1, 1),
                batchNorm = true
    })
    net:add(nn.SpatialAveragePooling(3, 3))
    -- Validate shape with:
    -- net:add(nn.Reshape(896))
    net:add(nn.View(896))
    net:add(nn.Linear(896, opt.embSize))
    net:add(nn.Normalize(2))
    return net
end
```

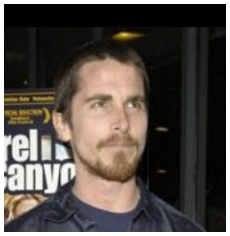
C: Some of the images used for testing


Aaron Eckhart


Al Pacino


Aishwarya Rai


Brad Pitt


Christian Bale


Avishek De


Gaurav Waghmare


Chitra Kumari


Richa Jain


Sankalp Wagh