Real-time Facial Recognition

Utilization of face recognition algorithms in real-time environment

Project Report - CIS 667 – Introduction to Artificial Intelligence

Patel Mohammed Farees

Abstract

This report aims to define, implement and test different methods which accomplish the task of facial detection and recognition in real-time.

Face recognition is an important application of Image processing owing to its use in many fields. The project presented here was developed after study of various face recognition methods and their efficiencies. An effective and real time face recognition system based on OpenCV and Python is developed in the project. The system was tested on AT&T's database of faces and additionally on 25 different students of Syracuse university.

For facial detection within video (sequence of images), cascade classifiers are a popular choice for those who wish to emphasize speed in their detection. In this project, we will explore how detection is accomplished with the Haar-cascades method (also known as the Viola-Jones method). The Haar-cascades detection method is included within the OpenCV library, as well as many different pre-trained XML face classifiers which will assist in the detection process. Facial recognition is slightly more complex. Of the many algorithms available which are used to accomplish this task, we will inspect and implement three: Eigenfaces, Fisherfaces, and LBPH (Local binary patterns histograms). Luckily, all three algorithms are available in the OpenCV library. All three methods compare the input image (unknown face) with a training set of known faces. In the training set, known faces (20) are provided for each of the 25 people in our training set. When an algorithm is supplied an unknown face, it calls upon the training set to assist in recognition. The system responds to the face recognition queries in less than 200 milliseconds. Each of our three algorithms will accomplish this task in a different manner.

Contents

1. Introduction

Face detection has been a fascinating problem for image processing researchers during the last decade because of many important applications such as video face recognition at airports and security check-points, digital image archiving, etc. In this project, we attempt to detect faces in a digital image using various techniques such as Fisher's linear discriminant (FLD), Eigen-face decomposition, LBPH (Local binary patterns histograms). We determined that the more complex classifiers did not work as well as expected due to the lack of large databases for training.

1.1 Motivation

Face recognition has been a sought after problem of biometrics and it has a variety of applications in modern life. The problems of face recognition attract researchers working in biometrics, pattern- recognition field and computer vision. Several face recognition algorithms are also used in many different applications apart from biometrics, such as video compression, indexing etc. An efficient face recognition system can be of great help in forensic sciences, identification for law enforcement, surveillance, authentication for banking and security system, and giving preferential access to authorized users i.e. access control for secured areas etc. The problem of face recognition has gained even more importance after the recent increase in the terrorism related incidents. Use of face recognition for authentication is the state of art technique adopted by Apple for unlocking the IPhone X. This eliminates the need of remembering passwords and can provide a much greater security if face recognition is used in combination with other security measures for access control. The cost of the license for an efficient commercial Face recognition system ranges from 30,000 $ to 150,000 $ which shows the significant value of the problem. Though face recognition is considered to be a very crucial authentication system but even after two decades' continuous research and evolution of many face recognition algorithms, a truly robust and efficient system that can produce good results in real-time and normal conditions is still not available. The Face Recognition Vendor Test (FRVT) [8] that has been conducted by the National Institute of Standards and Technology (NIST), USA, has shown that the commercial face recognition systems do not perform well under the normal daily conditions. Some of the latest face recognition algorithm involving machine learning tools perform well but sadly the training period and processing time is large enough to limit its use in practical applications. Hence there is a continuous strife to propose an effective face recognition system with high accuracy and acceptable processing time.

1.2 Background

Facilitate people identification had been a crucial challenge for a long time. From ID papers such as ID card to recent biometric methods, the techniques are still ongoing with improvement. With new technologies, we still need to introduce safe, easy and reliable methods for people identification. The goal is to match each face with the corresponding person. We denote recent improvements in the field of computer science, specifically in the field of machine learning, deep learning and

computer vision that we think they could allow a lot of enhancement for face recognition. Some researchers are still done on the subject. We could relate for example the recent research work of Brandon Amos, Bartosz Ludwiczuk and Mahadev Satyanarayanan from School of Computer Science of Carnegie Mellon University, in June 2016. They demonstrated how their open source face recognition library OpenFace open the possibilities for face recognition with mobile applications. Their process was built around the LFW (Labeled Faces in the Wild), a public dataset of more than 13000 Labeled Faces images from about 6000 people.

My main goal with this project is to build a system for face recognition in video using various technologies such as machine learning, deep learning, and mobile application building in python. Generally, this requires to have the faces to recognize as input, but they may already be available in a given dataset (associated to a dataset like the LFW presented above) so that the system can compute necessary comparisons methods and give an efficient result. This project should also be applied to both live videos and recorded videos. At the end of the project, users can train the model on the fly and recognize faces included in their local database using a live video capture with their webcam.

2 Project Description

In this project, I have implemented a Haar feature-based cascade classifier along with three different appearance-based facial recognition algorithms, which applies holistic texture features to either entire face or specific regions in face image. Among many techniques that are appearance-based, three popular techniques for this purpose are Principal Component Analysis, Linear Discriminant Analysis and Local Binary Patterns Histogram (LPBH).

To summarize, the tasks required for this project can be divided into four primary goals:

1. Capture, read and manipulate sequences of image (video)
2. Face detection and feature extraction
3. Building facial recognition model
4. Recognition in a live video feed

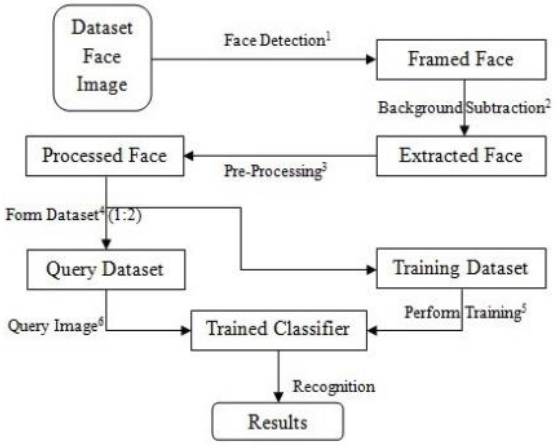


Fig (a): High-level overview of the system

This project allows user to either provide a set of images of a person or capture images of a person through a webcam (20 frame per 20 seconds). These images are stored in the database where, all the images of a single person are associated to the person's name, this itself is the class label that is used for recognizing that person in real-time.

After capturing and saving the image in the database, our next objective will be to locate and preserve the face region from the captured image and discard the background. For achieving this goal, we will be using Haar feature-based cascade classifier for face detection and extracting features that contributes to the face region. We will be discussing the whole process of feature extraction in detail in the next section.

The next goal would be to manipulate the face regions and prepare data for facial recognition.

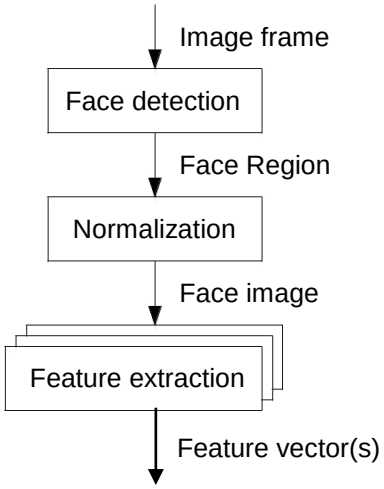


Fig (b): Face detection and Feature extraction process

The frame with annotated face regions is further processed with pre-processing part of the recognition pipeline. Here the face region is normalized and improper lighting conditions may be compensated. This normalization process is also explained in detail in the next section.

Finally, we implement a facial recognition model using PCA, LDA and Local Binary Patterns Histogram approach.

PCA is a dimension reduction method which projects "n" dimensional data into k dimensional subspace where k is small than n, and the k dimensional subspace is defined leading eigenvectors of the data's covariance matrix. PCA achieves facial recognition by projecting face images onto a feature space called "Eigen faces" through PCA. We are going to discuss how this performs on images and how PCA works from a mathematical standpoint in the next section.

LDA on the other hand, is a supervised learning algorithm. The aim of LDA is to find a linear combination of features which separate different classes of objects. In the other words, LDA is supposed to find a subspace on which objects classes are far away from each other while requiring objects in the same class are close to each other.

The third approach for facial recognition is using LBPH. This feature extraction method describes the texture and the shape of a digital image. This is done by dividing an image into several image regions from which the features are extracted. These features consist of binary patterns that describe the surrounding of pixels in the regions. The obtained features from the regions are concatenated into a single feature histogram, which forms a representation of the image. Images can then be compared by measuring the similarity (distances) between their histogram. All of the above mentioned three approaches are described in detail in the next section.

Finally, after training our model on the training images using the above mentioned three approaches, we then recognize faces in real-time or in a live video feed.

3 Face Detection and Recognition Pipeline

3.1 Capture, Read and Manipulate sequence of Images

My application enables users to either provide a database that comprises of set of images of many people or let user build his own database on the fly by capturing 20 images of a person (1 frame per second) to help the classifier train the model.

Each captured image acts as an input to face detection and normalization model. Surprisingly, images captured through OpenCV are not in the traditional RGB format, instead is captured in the BGR format. Hence, as a preprocessing step the captured image is first converted into RGB format and then the RGB image is converted into a matrix that comprises of pixel values. The below is just an overview of the transformations performed on the captured image before it is fed to the face detection and normalization model.

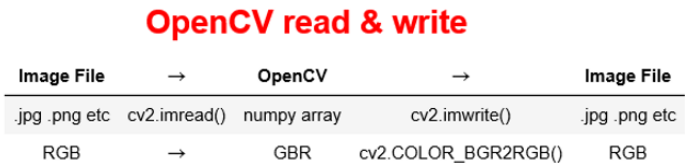


Fig (c): Image transformations

3.2 Face Detection and Feature Extraction

We implement an Independent Viola-Jones Haar feature-based cascade-classifier for face detection and localization in each frame. This application implements the Haar feature calculation and cascade of classifiers for each frame, independent of the previous or future frames. The application does not

maintain state about any located face, since faces are detected only for a given frame as it is given to the algorithm.

Face detection uses the Viola-Jones face tracking system. At a high level, for a given image frame, the system searches many regions of interest in the frame. For each region of interest, the system computes a set of Haar features on the frame, and uses a cascade of classifiers to determine whether or not to reject this region of interest (i.e. determine that it contains no faces) or determine that it does contain a face [?].

A Haar feature is a way to compress spatial information from a region of interest into a scalar value. In Figure 2, we observe that a Haar feature can be expressed as a binary image mask. To compute the Haar feature for a given region of interest, we compute:

$$Output = \sum P_{White} - \sum P_{Black}$$

Where P_{White} and P_{Black} are the set of pixels in the region of interest under a white area and black area in the Haar feature mask, respectively. For a given region of interest and Haar feature, this gives us a scalar value to use as input for classification. For this project, we use a pre-trained set of Haar features from Viola-Jones that are optimized for facial detection.

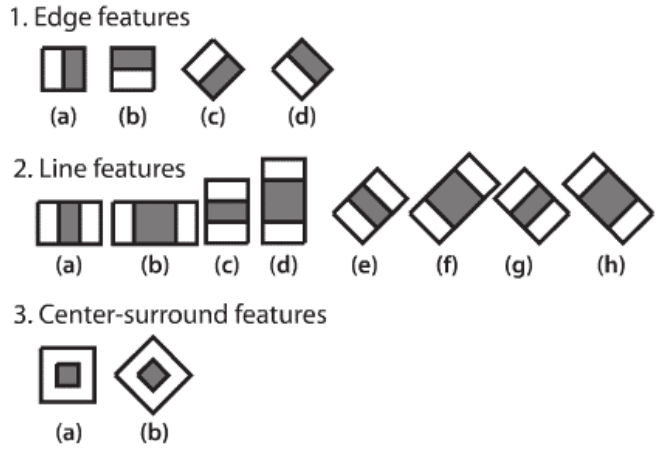


Fig (d): Example of Haar feature masks, which are used to compute scalar feature values for a region of interest.

To determine whether a region of interest contains a face or not, we use a computationally efficient system called a cascade of classifiers. Because computing the entire feature set for every candidate region of interest would be computationally very expensive (even for a machine with more resources than our target mobile platform), we would like to reject a region of interest early, before computing all features.

The cascade of classifier works as follows: for a given region of interest, we repeatedly apply a stage (i.e. a small subset) of our Haar features. Using these feature values and their corresponding weights pre-trained by the classifier, we

compute a score for the region of interest. If the score is below a certain threshold, we can immediately reject this region of interest and avoid computing more features for it. If the score is at least the threshold, we continue to apply more stages on this region of interest. If a region of interest passes all stages, we determine that it contains a face.

The stage sizes increase as they progress (e.g. the one used in this project goes from 1, 10, 25, 25, 50 features in the first five stages). The intuition behind this is that we would like to reject the low-hanging fruit (i.e. regions of interest that can be easily eliminated using one Haar feature) before spending more computational resources calculating additional Haar features. As a region of interest passes more stages, we apply a stricter (i.e. larger) set of features so we can be con dent that resulting regions of interest contain faces.

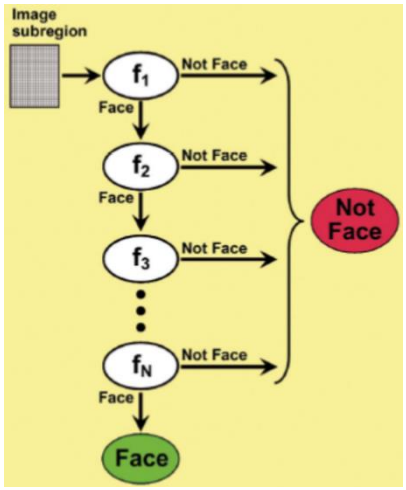


Fig (e): General schematic of cascade of classifiers, during which regions of interests are iteratively evaluated with a stricter set of Haar features to determine the presence of a face.

Now once we have detected the face region, the image goes through a sequence of transformations which we described as the normalization process. We first preserve the face region in the image and discard the background. In other words, we cut the image to retain only the region of interest i.e. the face region. Secondly, we normalize the pixel intensities so that it uses the complete greyscale spectrum [0-255]. In other words, the contrast is enhanced by normalizing the image. We can see how this normalization works below for a random image.

Fig (f): An un-equalized image vs Histogram equalized image

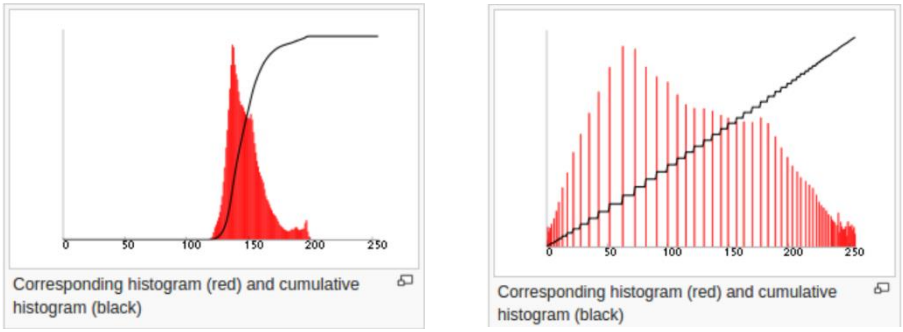


Fig (g): Histogram comparison of un-equalized image vs Histogram equalized image

4. Building Facial Recognition Models

This section will include a detailed explanation of different facial recognition models we used for recognizing people in a live video feed. I implemented three different appearance-based facial recognition algorithms, which applies holistic texture features to the face image. Among many techniques that are appearance-based, three popular techniques for this purpose are Principal Component Analysis, Linear Discriminant Analysis and Local Binary Patterns Histogram (LPBH). Each of them are explained in detail below.

4.1 Face recognition using Eigen-face Algorithm

The Eigen-face approach to facial recognition relies in performing what's called Principle Component Analysis (PCA), finding directions with the greatest variance within the data. This is easily done after each image is converted to grayscale; essentially turning each image into a vector within a vector space. After a PCA is performed on these vectors, we obtain the eigenvectors which make up the basis of the vector space.

The eigenvectors we obtain are very important to us since they usually represent the most prominent features of the data in question (faces). This is especially crucial when attempting to identify an unknown face. In this situation, we can decompose the unknown face and assess which eigenvectors most closely correlate to the image in question. We are then able to compare the similar eigenvectors to our training set, and attempt to find a successful recognition

Algorithmic implementation:

Let $X = \{x_1, x_2, \ldots, x_n\}$ be a random vector with observations $x_i \in R^d$.

1. The mean (u) is computed for each vector

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

2. Covariance matrix S is calculated for each vector

$$S = \frac{1}{n}\sum_{i=1}^{n} (x_i - \mu)(x_i - \mu)^T$$

3. The eigenvalues and eigenvectors are calculated for each vector

$$Sv_i = \lambda_i v_i, i = 1, 2, \ldots, n$$

4. The k principle components are found according to the largest calculated eigenvalues. For any vector x, its k principle components are given by:

$$y = W^T(x - \mu)$$

5. We can then reconstruct the projection by:

$$x = Wy + \mu$$

6. Recognition is then performed by projecting all training samples (and the input image in question) into the PCA subspace.

7. The nearest neighbor (eigenvalues) are found between the projected training samples and the projected input image.

4.2 Face recognition using Fisher-face Algorithm

Where the Eigen-face method utilizes PCA to project the image space to a low dimensional subspace and maximize the total scatter across all classes (faces), the Fisher-face method uses LDA (Linear Discriminant Analysis) to maximize the ratio of between-class scatter to that of within-class scatter.

The Fisher-faces method attempts to make classes cluster tightly together, while different classes are as far away as possible from each other in the lower-dimensional representation. It is important to note that since we only identified the features to distinguish between subjects, our reconstruction will not be as detailed as in the Eigen-faces method.

Algorithmic implementation:

1. For any vector X with samples taken from c classes:

$$X = \{X_1, X_2, \ldots, X_c\}$$
$$X_i = \{x_1, x_2, \ldots, x_n\}$$

2. Scatter matrices S_B and S_W are then found by:

$$S_B = \sum_{i=1}^{c} N_i(\mu_i - \mu)(\mu_i - \mu)^T$$
$$S_W = \sum_{i=1}^{c} \sum_{x_j \in X_i} (x_j - \mu_i)(x_j - \mu_i)^T$$

3. The total mean (u) is calculated by:

$$\mu = \frac{1}{N} \sum_{i=1}^{N} x_i$$

4. The mean of each class (person) is calculated (u_i):

$$\mu_i = \frac{1}{|X_i|} \sum_{x_j \in X_i} x_j$$

5. The algorithm now finds a projection W which maximizes separability between classes:

$$W_{opt} = \arg\max_W \frac{|W^T S_B W|}{|W^T S_W W|}$$

6. An optimized solution is found by:

$$S_B v_i = \lambda_i S_w v_i$$
$$S_W^{-1} S_B v_i = \lambda_i v_i$$

7. And then rewritten as:

$$W_{pca} = \arg\max_W |W^T S_T W|$$
$$W_{fld} = \arg\max_W \frac{|W^T W_{pca}^T S_B W_{pca} W|}{|W^T W_{pca}^T S_W W_{pca} W|}$$

4.3 Face recognition using Local Binary Pattern Histogram Algorithm (LBPH)

In contrast to the previous two methods, LBPH analyzes each image independently rather than inspecting the whole dataset. The LBPH method characterizes each image in the dataset locally. When an unknown input image is provided, we perform the same analysis on it and compare the result to each of the images in the dataset. By characterizing the local patterns found in specific image locations, we can successfully analyze each image based on these results.

Algorithmic implementation:

1. We can describe the LBP operator as:

$$LBP(x_c, y_c) = \sum_{p=0}^{P-1} 2^p s(i_p - i_c)$$

2. Where (x_c,y_c) is the central pixel with intensity i_c , and i_n representing the intensity of a neighbor pixel. With that, our sign function can be defined as:

$$s(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{else} \end{cases}$$

3. For any given point (x_c,y_c), its neighbor's position (x_p,y_p), where p is an element of P, is found by:

$$x_p = x_c + R\cos(\frac{2\pi p}{P})$$

$$y_p = y_c - R\sin(\frac{2\pi p}{P})$$

4.4 K-Nearest Neighbor Classification Algorithm

The k-nearest neighbor algorithm is a method for classifying objects based on closest training objects: an object is classified according to a majority vote of its neighbors. In this project, KNN plays an important role in recognition phase. The steps of performing KNN are as follows:

Find K nearest neighbors.

For a specific test face, it is first projected onto the subspace defined by W, and W is calculated by any methods mentioned above such as PCA, LDA and LBPH. Afterward, an array with size is created to keep the K nearest neighbors, and the way to achieve this goal is to go through the whole training set and

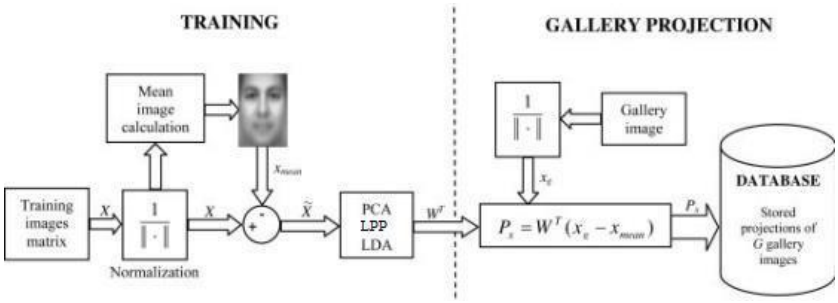
update the array if there is a training face whose distance is smaller than the largest distance in the original array. At the end of this process, the K nearest neighbors are found. Also, please notice that the training set has already been projected onto the same subspace defined by W.

Classify according to weights.

After the K nearest neighbors are found, a HashMap is created to keep the pairs <label, weight> by going through all the neighbors. If the HashMap happens to meet a new label, <label, 1 / distance> is directly added into the map. Otherwise, an updated version which adds the original weight with 1 / distance is put into the map. After the HashMap is correctly constructed, this program will go through the whole HashMap and find the label associated with the largest weight. This class label will be the label recognized by this system.

5. Face Recognition Methodology

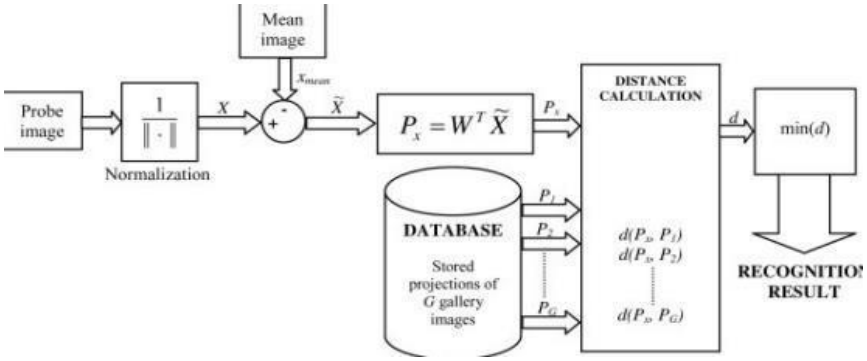
After introducing PCA, LDA, LBPH and KNN, now it is the time to combine them together to build the face recognition system. The methodology used in this project can be divided into the following three parts:



Fig(h): An illustration of the training process

As Fig(h) shows, there are two major tasks: training and gallery projection. For the training part, the training face set is normalized followed by subtracting the mean face. The projection matrix is then calculated using PCA, LBPH or LDA. For the gallery projection, faces with known labels are projected onto the subspace defined by, and their projections are stored in the database for the next step.

2. Recognizing



Fig(i): An illustration of the recognizing process

Now it is the time to recognize an unknown face. As Fig(i) illustrates, the probe image is normalized followed by subtracting the mean face of the original training set. Next it is projected onto subspace, and the projected data is calculated. The last step is find 's K nearest neighbors among the database constructed in step 2.

Finally, the label of the probe face is predicted according to the majority vote among the K nearest neighbors.

This is a general description about the face recognition system built in this project.

6 Results

Our training set is comprised of images from the AT&T Face database. This database contains 40 separate people (classes), each with 10 facial images each. This makes for total of 400 images within the database. These images contain various facial expressions, lighting, and facial accessories (glasses). This experiment was performed just to evaluate the facial recognition performance of our model. Below is just a glimpse of our dataset.

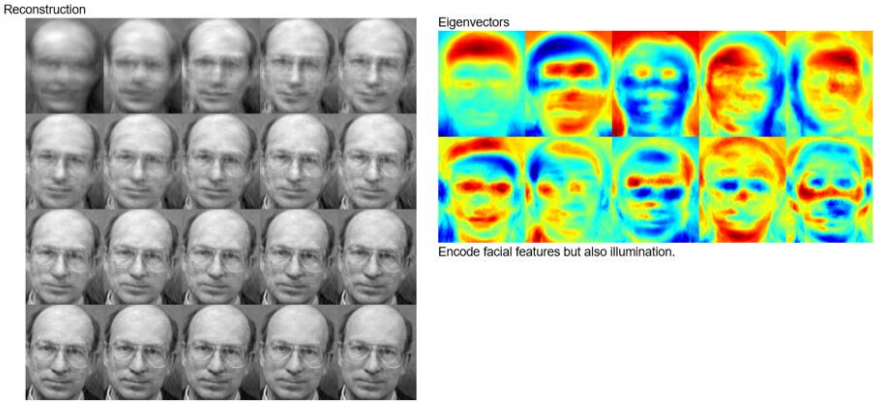
Fig (j): AT&T Face Database

Facial recognition with Eigen-faces

An eigen-face model will be trained on the AT&T database of 399 images except for the image shown below. This image will be used as the input within the trained model, and it will attempt to successfully match it to the correct class. The image belongs to class 13. The model will calculate all the eigenvalues. But we can see that, if we consider the first 10 eigenvalues of an image, it is more than enough to recognize or reconstruct the face

Input Image:

Reconstruction:

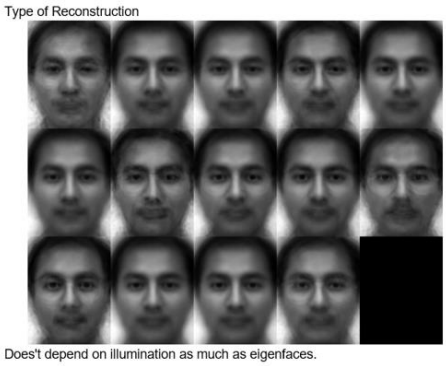


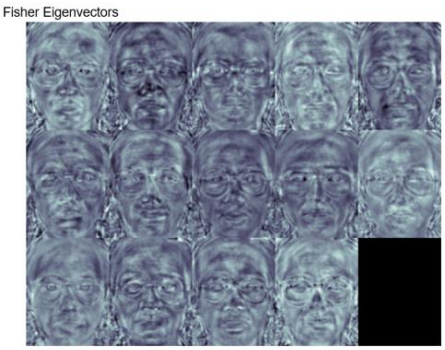
Facial recognition with Fisher-faces

A fisher-face model will be trained on the AT&T database of 399 images except for the image shown below. This image will be used as the input within the trained model, and it will attempt to successfully match it to the correct class. The image belongs to class 13. The model will calculate the first 16 eigenvalues.

Input Image:

Reconstruction



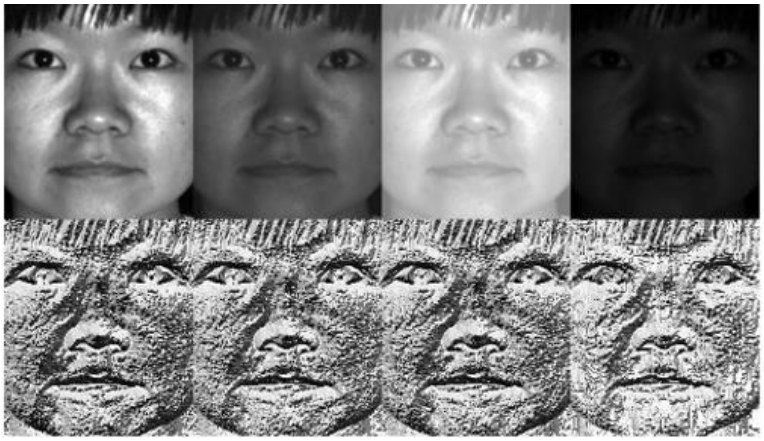


Facial recognition with LBPH

An eigen-face model will be trained on the AT&T database of 399 images except for the image shown below. This image will be used as the input within the trained model, and it will attempt to successfully match it to the correct class. The image belongs to class 13. The method will use a radius of 1, and a neighborhood of 8 neighbors.

Input Image:

Reconstruction:



Now for real-time face recognition, I captured images of 25 people. (20 images per person) for training my recognition model. I achieved an accuracy of 100% i.e. my model correctly classified every single person in the real-time video feed. I have also calculated the confidence value for every prediction. The confidence value id the difference between facial features of the predicted and the actual face. Below ids the results that I obtained for predicting real-time faces of 25 people.

```
Actual Class = 40 | Predicted Class = 40
==================================================
Image 1: Eigen-face confidence: 1022.24| Fisher-face confidence: 1850.33| LBPH confidence: 2040.59
Image 2: Eigen-face confidence: 1478.45| Fisher-face confidence: 1068.53| LBPH confidence: 1190.76
Image 3: Eigen-face confidence: 1243.23| Fisher-face confidence: 1310.12| LBPH confidence: 1099.23
Image 4: Eigen-face confidence: 1209.31| Fisher-face confidence: 1590.45| LBPH confidence: 1909.67
Image 5: Eigen-face confidence: 2045.00| Fisher-face confidence: 1046.77| LBPH confidence: 1320.07
Image 6: Eigen-face confidence: 2345.12| Fisher-face confidence: 1009.09| LBPH confidence: 1987.65
Image 7: Eigen-face confidence: 2111.32| Fisher-face confidence: 1938.14| LBPH confidence: 1476.31
Image 8: Eigen-face confidence: 2097.50| Fisher-face confidence: 1002.95| LBPH confidence: 1323.18
Image 9: Eigen-face confidence: 1432.09| Fisher-face confidence: 1120.23| LBPH confidence: 2877.09
Image 10: Eigen-face confidence: 1908.24| Fisher-face confidence: 1008.01| LBPH confidence: 2844.40
Image 11: Eigen-face confidence: 1378.15| Fisher-face confidence: 1980.15| LBPH confidence: 1002.68
Image 12: Eigen-face confidence: 1028.03| Fisher-face confidence: 2345.16| LBPH confidence: 1304.09
Image 13: Eigen-face confidence: 1647.12| Fisher-face confidence: 1353.19| LBPH confidence: 2300.65
Image 14: Eigen-face confidence: 2073.23| Fisher-face confidence: 1890.23| LBPH confidence: 2470.15
Image 15: Eigen-face confidence: 2098.04| Fisher-face confidence: 1233.45| LBPH confidence: 1043.04
Image 16: Eigen-face confidence: 2123.65| Fisher-face confidence: 1098.78| LBPH confidence: 1465.23
Image 17: Eigen-face confidence: 2009.09| Fisher-face confidence: 1235.09| LBPH confidence: 1034.01
Image 18: Eigen-face confidence: 1324.68| Fisher-face confidence: 2345.12| LBPH confidence: 1221.52
Image 19: Eigen-face confidence: 1098.29| Fisher-face confidence: 2909.30| LBPH confidence: 1421.93
Image 20: Eigen-face confidence: 2009.14| Fisher-face confidence: 1112.13| LBPH confidence: 1002.19
Image 21: Eigen-face confidence: 2109.13| Fisher-face confidence: 1001.33| LBPH confidence: 2667.91
Image 22: Eigen-face confidence: 1390.35| Fisher-face confidence: 2197.67| LBPH confidence: 1884.50
Image 23: Eigen-face confidence: 1095.20| Fisher-face confidence: 1293.14| LBPH confidence: 1374.13
Image 24: Eigen-face confidence: 1256.04| Fisher-face confidence: 1995.30| LBPH confidence: 1413.45
Image 25: Eigen-face confidence: 2011.15| Fisher-face confidence: 1435.93| LBPH confidence: 2147.06
```

Fig (k): Performance measure comparison for Eigen-face, Fisher-face and LBPH for face recognition of 25 people

7 Conclusion and Future Work

From the data, we can see some very interesting things. First, and most importantly, all three of our recognition models predicted the correct class for all the 25 people that were selected for evaluating our model. We can observe from the results that each recognition model has a different confidence value for a predicted image. This tells us that the way Eigen-face, Fisher-face and LPBH recognize faces are different. Eigen-faces and Fisher-faces find a mathematical description of the most dominant features of the training set as a whole whereas, LPBH analyzes each face in the training set separately and independently.

Additionally, we see some major differences in the reconstruction of the input image between the Eigen-face and Fisher-face methods. The Eigen-face method, using PCA, with its goal of maximizing scatter across all classes, is able to provide a relatively detailed reconstruction of the decomposed input image. The fisher-face method, using LDA instead of PCA, which aims to maximize between class scatter rather than total scatter. Because of this, reconstruction of the decomposed input image will not provide any helpful information to us.

Some improvements could have been made in my methods and experiments. In the LBPH model, a better representation of the histograms

should have been added to better visualize what was occurring during recognition. It is helpful having the size of the histograms, which was included, but having a graph would have been more descriptive for the user.

Additionally, I would love to continue working on this project and explore and implement neural network and deep learning based facial recognition system and compare how it will perform against our proposed method of Eigen-face, Fisher-face and LBPH methods.

8 References

Delac, K., Grgic, M., & Grgic, S. (2005). Independent comparative study of PCA, ICA, and LDA on the FERET data set. *International Journal of Imaging Systems and Technology*, *15*(5), 252-260.

Turk, M., & Pentland, A. (1991). Eigenfaces for recognition. *Journal of cognitive neuroscience*, *3*(1), 71-86.

Belhumeur, P. N., Hespanha, J. P., & Kriegman, D. J. (1997). Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *19*(7), 711-720.

He, X., Yan, S., Hu, Y., Niyogi, P., & Zhang, H. J. (2005). Face recognition using laplacian-faces. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *27*(3), 328-340.

9 Appendix

1. Revised Project Proposal
2. Source Code Listing

Appendix Part A: Revised project Proposal

Real Time Face recognition system

The goal of this project is to implement a reliable real-time automated face recognition system using three different appearance-based facial recognition approaches along with the Open Source Computer Vision (OpenCV) library in Python.

Problem Description

Face tracking and detection is an important and fundamental problem defined in AI, especially in computer vision. This area of research has a lot of applications in face identification systems, model based coding, gaze detection, human computer interaction, teleconferencing, etc. The objective of this project is to go through general ideas and structures of face recognition using deep learning, important issues to load, summarize and visualize the data. I will explain the critical techniques and algorithms, and finally give some predictions and present my results. I will be referring to the Section VI of our AI textbook, that highlights image processing and object recognition from structural information. I will be implementing an AI algorithm of computer vision. I am hoping to implement Eigen-face, Fisher-face and Local Binary Pattern Histogram (LBPH) approaches to recognize human faces in real time.

Motivations

My motivation for doing this project is to build a security product using an AI implementation. I am interested in image processing especially in this project because I believe that facial recognition is used in a wide variety of industries such as security, marketing, and entertainment. In many security applications, users often need to detect and locate faces in real-time in surveillance footage. Additionally, applications often use facial recognition in order to deduce a person's identity from an image of their face.

Nature of Proposed work:

This is an applied project because it involves solving machine learning problem of predicting facial identity using machine learning framework of computer vision. My project will also include comparison of an Eigen-face, Fisher-face and Local Binary Pattern Histogram techniques
for face detection from an accuracy standpoint.

Method

The objective is to build a face recognition model to detect faces in a live video feed. The major technical challenges for this project are locating and detecting faces in real-time. Detecting faces poses the challenge of localizing faces while being robust to varying lighting, angle, pose, and size. Finally, running this in real-time requires efficiency measures - for example, we can utilize flow tracking to avoid re-detecting faces in each frame. To summarize, the tasks required for this application can be divided into three primary goals:

1. Face Localization: The input to this system is a video stream with zero or more faces per frame. I will be using the OpenCV library in python to extract facial features and bound boxes around each detected face.

2. Classification: The face detected should be associated to a label and these should be used to train the neural network model. For detecting faces, I will be using variants of appearance-based approaches such as EigenFaces algorithm and Adaboost to build a strong classifier to detect faces in real time.

3. Real-time performance: The goal for this application is to run in real time on a laptop (i5 3.2 GHz CPU). For this project, the video capturing device is the webcam installed on the laptop.

Time-line and evaluation criteria:

Task Name	Duration	Self-assessment criteria	Finish Date
Explore ideas and collect image data	10 days	The criteria for this phase involves exploring ideas and learning various implementation algorithms for face detection and collecting image data for training models	11/10
Detect faces from image	10 days	Criteria is to complete using open cv library along with numpy and pandas to detect faces from an image in Python	11/20
Building neural network and other ML modes for facial recognition	15 days	The criteria for this phase involves extracting facial features, building eigenFaces algorithm, training recognition models and Adaboost to detect faces in real time	12/5
Changes and debugging	5 days	Validating source code and making any changes if required	12/10
Final Report	4 days	Completion of project report	12/14

The final report will consist of project design, source code listings, details and walk through of project implementation. Further my work can be assessed based on oral presentation and project demo.

Reference Paper:

Viola,P., Jones,M.(2014). Robust Real-Time Face Detection, International Journal of Computer Vision. Retreived from: http://www.vision.caltech.edu/html-files/EE148-2005-Spring/pprs/viola04ijcv.pdf

Appendix Part B: Source Code Listing

```
# In[1]:


from IPython.display import YouTubeVideo
import cv2
import numpy as np
import os
import math
from matplotlib import pyplot as plt
from IPython.display import clear_output

get_ipython().magic('matplotlib inline')
# Open a new thread to manage the external cv2 interaction
cv2.startWindowThread()

def plt_show(image, title=""):
    if len(image.shape) == 3:
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    plt.axis("off")
    plt.title(title)
    plt.imshow(image, cmap="Greys_r")
    plt.show()

class FaceDetector(object):
    def __init__(self, xml_path):
        self.classifier = cv2.CascadeClassifier(xml_path)

    def detect(self, image, biggest_only=True):
        scale_factor = 1.2
        min_neighbors = 5
        min_size = (30, 30)
        biggest_only = True
        flags = cv2.CASCADE_FIND_BIGGEST_OBJECT |
cv2.CASCADE_DO_ROUGH_SEARCH if biggest_only else
cv2.CASCADE_SCALE_IMAGE
        faces_coord = self.classifier.detectMultiScale(image,
                                                       scaleFactor=scale_factor,

minNeighbors=min_neighbors,
                                                       minSize=min_size,
                                                       flags=flags)
        return faces_coord
```

```
class VideoCamera(object):
    def __init__(self, index=0):
        self.video = cv2.VideoCapture(index)
        self.index = index
        print self.video.isOpened()

    def __del__(self):
        self.video.release()

    def get_frame(self, in_grayscale=False):
        _, frame = self.video.read()
        if in_grayscale:
            frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        return frame

def cut_faces(image, faces_coord):
    faces = []

    for (x, y, w, h) in faces_coord:
        w_rm = int(0.3 * w / 2)
        faces.append(image[y: y + h, x + w_rm: x + w - w_rm])

    return faces

def normalize_intensity(images):
    images_norm = []
    for image in images:
        is_color = len(image.shape) == 3
        if is_color:
            image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        images_norm.append(cv2.equalizeHist(image))
    return images_norm

def resize(images, size=(50, 50)):
    images_norm = []
    for image in images:
        if image.shape < size:
            image_norm = cv2.resize(image, size,
                                    interpolation=cv2.INTER_AREA)
        else:
            image_norm = cv2.resize(image, size,
                                    interpolation=cv2.INTER_CUBIC)
        images_norm.append(image_norm)
```

```
    return images_norm

def normalize_faces(frame, faces_coord):
    faces = cut_faces(frame, faces_coord)
    faces = normalize_intensity(faces)
    faces = resize(faces)
    return faces

def draw_rectangle(image, coords):
    for (x, y, w, h) in coords:
        w_rm = int(0.2 * w / 2)
        cv2.rectangle(image, (x + w_rm, y), (x + w - w_rm, y + h),
                      (150, 150, 0), 8)




def collect_dataset():
    images = []
    labels = []
    labels_dic = {}
    people = [person for person in os.listdir("people/")]
    for i, person in enumerate(people):
        labels_dic[i] = person
        for image in os.listdir("people/" + person):
            images.append(cv2.imread("people/" + person + '/' +
image,
                                     0))
            labels.append(i)
    return (images, np.array(labels), labels_dic)


# ### Collect image data and train models

# In[3]:


images, labels, labels_dic = collect_dataset()

rec_eig = cv2.face.createEigenFaceRecognizer()
rec_eig.train(images, labels)

# needs at least two people
rec_fisher = cv2.face.createFisherFaceRecognizer()
rec_fisher.train(images, labels)
```

```
rec_lbph = cv2.face.createLBPHFaceRecognizer()
rec_lbph.train(images, labels)

print "Models Trained Succesfully"



# ### Get a sample picture

# In[9]:


filepath="C:\\Anaconda3\\envs\\py27\\Library\\etc\\haarcascades
\\haarcascade_frontalface_alt2.xml"
webcam = VideoCamera()
frame = webcam.get_frame()
detector = FaceDetector(filepath)
frame = webcam.get_frame()
faces_coord = detector.detect(frame)
faces = normalize_faces(frame, faces_coord)
face = faces[0]
plt_show(face)


# In[11]:


del webcam


# ### Make Predictions with the three models

# In[12]:


collector = cv2.face.MinDistancePredictCollector()

rec_eig.predict(face, collector)
conf = collector.getDist()
pred = collector.getLabel()
print "Eigen Faces -> Prediction: " + labels_dic[pred].capitalize()
+"    Confidence: " + str(round(conf))

rec_fisher.predict(face, collector)
```

```
conf = collector.getDist()
pred = collector.getLabel()
print "Fisher Faces -> Prediction: " +labels_dic[pred].capitalize()
+ "    Confidence: " + str(round(conf))

rec_lbph.predict(face, collector)
conf = collector.getDist()
pred = collector.getLabel()

print "LBPH Faces  -> Prediction: " + labels_dic[pred].capitalize()
+"    Confidence: " + str(round(conf))


# coding: utf-8

# In[9]:


from IPython.display import YouTubeVideo
import cv2
import numpy as np
import os
import math
from matplotlib import pyplot as plt
from IPython.display import clear_output

get_ipython().magic('matplotlib inline')
# Open a new thread to manage the external cv2 interaction
cv2.startWindowThread()

def plt_show(image, title=""):
    if len(image.shape) == 3:
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    plt.axis("off")
    plt.title(title)
    plt.imshow(image, cmap="Greys_r")
    plt.show()

class FaceDetector(object):
    def __init__(self, xml_path):
        self.classifier = cv2.CascadeClassifier(xml_path)

    def detect(self, image, biggest_only=True):
        scale_factor = 1.2
        min_neighbors = 5
```

```
        min_size = (30, 30)
        biggest_only = True
        flags = cv2.CASCADE_FIND_BIGGEST_OBJECT |
cv2.CASCADE_DO_ROUGH_SEARCH if biggest_only else
cv2.CASCADE_SCALE_IMAGE
        faces_coord = self.classifier.detectMultiScale(image,
                                        scaleFactor=scale_factor,

minNeighbors=min_neighbors,
                                        minSize=min_size,
                                        flags=flags)
        return faces_coord

class VideoCamera(object):
    def __init__(self, index=0):
        self.video = cv2.VideoCapture(index)
        self.index = index
        print self.video.isOpened()

    def __del__(self):
        self.video.release()

    def get_frame(self, in_grayscale=False):
        _, frame = self.video.read()
        if in_grayscale:
            frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        return frame

def cut_faces(image, faces_coord):
    faces = []

    for (x, y, w, h) in faces_coord:
        w_rm = int(0.3 * w / 2)
        faces.append(image[y: y + h, x + w_rm: x + w - w_rm])

    return faces

def normalize_intensity(images):
    images_norm = []
    for image in images:
        is_color = len(image.shape) == 3
        if is_color:
            image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        images_norm.append(cv2.equalizeHist(image))
    return images_norm
```

```
def resize(images, size=(50, 50)):
    images_norm = []
    for image in images:
        if image.shape < size:
            image_norm = cv2.resize(image, size,
                                    interpolation=cv2.INTER_AREA)
        else:
            image_norm = cv2.resize(image, size,
                                    interpolation=cv2.INTER_CUBIC)
        images_norm.append(image_norm)

    return images_norm

def normalize_faces(frame, faces_coord):
    faces = cut_faces(frame, faces_coord)
    faces = normalize_intensity(faces)
    faces = resize(faces)
    return faces

def draw_rectangle(image, coords):
    for (x, y, w, h) in coords:
        w_rm = int(0.2 * w / 2)
        cv2.rectangle(image, (x + w_rm, y), (x + w - w_rm, y + h),
                      (150, 150, 0), 8)

def collect_dataset():
    images = []
    labels = []
    labels_dic = {}
    people = [person for person in os.listdir("people/")]
    for i, person in enumerate(people):
        labels_dic[i] = person
        for image in os.listdir("people/" + person):
            images.append(cv2.imread("people/" + person + '/' +
image, 0))
            labels.append(i)
    return (images, np.array(labels), labels_dic)


# In[10]:


images, labels, labels_dic = collect_dataset()
```

```
rec_eig = cv2.face.createEigenFaceRecognizer()
rec_eig.train(images, labels)

# needs at least two people
rec_fisher = cv2.face.createFisherFaceRecognizer()
rec_fisher.train(images, labels)

rec_lbph = cv2.face.createLBPHFaceRecognizer()
rec_lbph.train(images, labels)

print "Models Trained Succesfully"


# In[17]:


filepath="C:\\Anaconda3\\envs\\py27\\Library\\etc\\haarcascades
\\haarcascade_frontalface_alt2.xml"
detector = FaceDetector(filepath)
webcam = VideoCamera(0)


# In[18]:


cv2.namedWindow("Face Recognition Model",
cv2.WINDOW_NORMAL)
while True:
   frame = webcam.get_frame()
   faces_coord = detector.detect(frame, True) # detect more than
one face
   if len(faces_coord):
      faces = normalize_faces(frame, faces_coord) # norm
pipeline
      for i, face in enumerate(faces): # for each detected face
         collector = cv2.face.MinDistancePredictCollector()
         rec_lbph.predict(face, collector)
         conf = collector.getDist()
         pred = collector.getLabel()
         threshold = 140
         print "Prediction: " + labels_dic[pred].capitalize() +
"\nConfidence: " + str(round(conf))
         cv2.putText(frame, labels_dic[pred].capitalize(),
                  (faces_coord[i][0], faces_coord[i][1] - 10),
                  cv2.FONT_HERSHEY_PLAIN, 3, (66, 53, 243), 2)
```

```
        clear_output(wait = True)
        draw_rectangle(frame, faces_coord) # rectangle around
face
    cv2.putText(frame, "ESC to exit", (5, frame.shape[0] - 5),
                cv2.FONT_HERSHEY_PLAIN, 1.3, (66, 53, 243), 2,
cv2.LINE_AA)
    cv2.imshow("Face Recognition Model", frame) # live feed in
external
    if cv2.waitKey(40) & 0xFF == 27:
        cv2.destroyAllWindows()
        break


# In[19]:


del webcam
#webcam = VideoCamera(0)
```