Face Recognition

Abstract— Identifying a person with an image has been popularised through the mass media. However, it is less robust to ﬁngerprint or retina scanning. The goal of this report is the presentation of our biometry and sect- rite course’s project: Face recognition for Labelled Faces in the Wild dataset using Convolutional Neural Network technology with Graph Lab Framework. Deep learning is the new big trend in machine learning. It had many recent successes in computer vision, automatic speech recognition and natural land- gauge processing. We will go through general ideas and structures of face recognition using deep learning, important issues to load, summarize and visualize the dataset, we will explain the critical techniques and algorithms, and ﬁnally give some predictions and present our results.

1. Introduction

In this document we'll show you how to implement the Eigenfaces and Fisher faces method with Python, so you'll

understand the basics of Face Recognition. All concepts are explained in detail, but a basic knowledge of Python is assumed. Originally this document was a Guide to Face Recognition with OpenCV. Since OpenCV now comes with the cv: Face Recognizer, this document has been reworked into the official OpenCV documentation I am doing all this in my spare time and I simply can't maintain two separate documents on the same topic any more. So I have decided to turn this document into a guide on Face Recognition with Python only. Face recognition is a non-invasive identiﬁcation system and faster than other systems since multiple faces can be analysed at the same time. The diﬀerence between face detection and identiﬁcation is, face detection is to identify a face from an image and locate the face. Face recognition is making the decision ”whose face is it ? ”, using an image database. In this project both are accomplished using diﬀerent techniques and are described below. The report begins with a brief history of face recognition. This is followed by the explanation of HAAR-cascades, Eigenface, Fisher face and Local binary pattern histogram (LBPH) algorithms. Next, the methodology and the results of the project are described. A discussion regarding the challenges and the resolutions are described.

1. The History of Face Recognition

A Face recognition began as early as 1977 with the ﬁrst automated system being introduced By Mandamusing a feature vector of human faces . In 1983, Service and Kirby introduced the principal component analysis(PCA) for feature extraction. Using PCA, Turk and Pentland Eigenface was developed in1991 and is considered a major milestone in technology [3]. Local binary pattern analysis for texture recognition was introduced in 1994 and is improved upon for facial recognition later by incorporating Histograms(LBPH)]. In 1996 Fisher face was developed using Linear discriminant analysis (LDA)for dimensional reduction and can identify faces in diﬀerent illumination conditions, which was an issue in Eigenface method [6]. Viola and Jones introduced a face detection technique using HAAR cascades and Adobos. In 2007, A face recognition technique was developed by Narnia and Skarbek using Gabor Jets that are similar to mammalian eyes. In This project, HAAR cascades are used for face detection and Eigenface, Fisher face and LBPH are used for face recognition Face Detection & Face Recognition.

1. Face Recognition

The following sections describe the face recognition algorithms Eigenface, Fisher face, Local binary pattern histogram and how they are implemented in OpenCV.

1. Methodology

Below are the methodology and descriptions of the applications used for data gathering, face detection, training and face recognition. The project was coded in Python using a mixture of IDLE and PYCharm IDEs.

1. Face Detection

First stage was creating a face detection system using Haar-cascades. Although, training is required for creating new Haar-cascades, OpenCV has a robust set of Haar-cascades that was used for the project. Using face-cascades alone caused random objects to be identiﬁed and eye cascades were incorporated to obtain stable face detection. Face and eye classiﬁer objects are created using classiﬁer class in OpenCV through the cv2.CascadeClassiﬁer() and loading the respective XML ﬁles. A camera object is created using the cv2.VideoCapture() to capture images. By using the Cascade Classiﬁer.detectMultiScale() object of various sizes are matched and location is returned. Using the location data, the face is cropped for further veriﬁcation. Eye cascade issued to verify there are two eyes in the cropped face. If satisﬁed a marker is placed around the face to illustrate a face is detected in the location.

1. Face Recognition Process

B For this project three algorithms are implemented independently. These are Eigenface, Fisher face and Linear binary pattern histograms respectively. All three can be implemented using OpenCV libraries. There are three stages for the face recognition as follows:

1. Collecting images IDs.

2. Extracting unique features, classifying them and storing in XML ﬁles.

3. Matching features of an input image to the features in the saved XML ﬁles and predict identity.

1. Collecting the image data

The Collecting classiﬁcation images is usually done manually using a photo editing software to crop and resize photos. Furthermore, PCA and LDA requires the same number of pixels in all the images for the correct operation. This time consuming and a laborious task is automated through an application to collect 50 images with diﬀerent expressions. The application detects suitable expressions between 300ms,straightens any existing tilt and save them. Application starts with a request for a name to be entered to be stored with the ID in a text ﬁle. The face detection system starts the ﬁrst half. However, before the capturing begins, the application check for the brightness levels and will capture only if the face is well illuminated. Furthermore, after the face is detected, the position of the eyes are analysed. If the head is tilted, the application automatically corrects the orientation. These two additions were made considering the requirements for Eigenface algorithm. The Image is then cropped and saved using the ID as a ﬁlename to be identiﬁed later. A loop runs this program until 50 viable images are collected from the person. This application made data collection eﬃcient.

1. Training the Classiﬁers

OpenCV enables the creation of XML ﬁles to store features extracted from datasets using the Face Recognizer class. The stored images are imported, converted to grayscale and saved with IDs in two lists with same indexes. Face Recognizer objects are created using face recogniser class. Each recogniser can take in parameters that are described below:

below:cv2.face.createEigenFaceRecognizer()

1. Takes in the number of components for the PCA for crating Eigenfaces. OpenCV documentation mentions 80 can provide satisfactory reconstruction capabilities.

2. Takes in the threshold in recognising faces. If the distance to the likeliest Eigenface is above this threshold, the function will return a -1, that can be used state the face is unrecognisable.

cv2.face.createFisherfaceRecognizer()

1. The ﬁrst argument is the number of components for the LDA for the creation of Fisherfaces. OpenCV mentions it to be kept 0 if uncertain.

2. Similar to Eigenface threshold. -1 if the threshold is passed.

cv2.face.createLBPHFaceRecognizer()

1. The radius from the centre pixel to build the local binary pattern.

2. The Number of sample points to build the pattern. Having a considerable number will slow down the computer.

3. The Number of Cells to be created in X axis.

4. The number of cells to be created in Y axis.

5. A threshold value similar to Eigenface and Fisherface. if the threshold is passed the object will return -1.

Recogniser objects are created and images are imported, resized, converted into numpy arrays and stored in a vector. The ID of the image is gathered from splitting the ﬁle name, and stored in another vector. By using FaceRecognizer.train(NumpyImage, ID) all three of the objects are trained. It must benoted that resizing the images were required only for Eigenface and Fisherface, not for LBPH. Next, the conﬁguration model is saved as a XML ﬁle using FaceRecognizer.save(FileName).

1. The Face Recognition

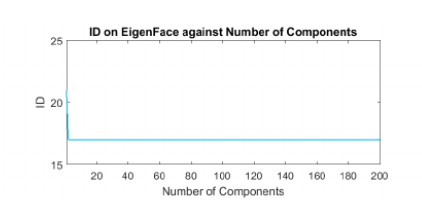
Face recogniser object is created using the desired parameters. Face detector is used to detect faces in the image, cropped and transferred to be recognised. This is done using the same technique used for the image capture application. For each face detected, a prediction is made using FaceRecognizer.predict() which return the ID of the class and conﬁdence. The process is same for all algorithms and if the condenses higher than the set threshold, ID is -1. Finally, names from the text ﬁle with IDs are used to display the name and conﬁdence on the screen. If the ID is -1, the application will print unknown face without the conﬁdence level.

1. Recognising a diﬀerent subject

For a secondary test, authors’ colleague provided a photo which was 7 years old. Although the photo has two people, plots only contain data on Mr. Westlake for clarity. The photo is shown below in ﬁgure 23.The idea was to observe if the program can identify a younger face from the data-set of an older face. The same program is used as above and plots for Eigenface, Fisher face and three separate pairs for the LBPH are below in ﬁgures 1, 2, 3, 4, 5.

Eigenface and Fisher face IDs and conﬁdence was as expected. Note that Mr. Westlake’s’ ID is 17. The LBPH ID ﬂuctuated when the pixels were above 38. Same can be noticed when cells are increased and low cells are unstable. The calibration details are detailed after the plots.

.



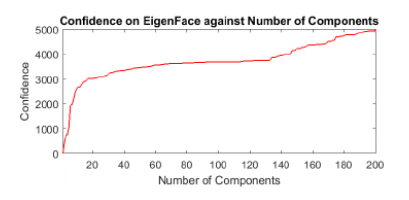
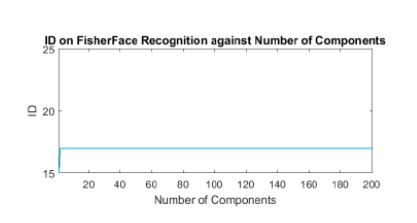


Fig. 1 ID and Conﬁdence for Eigenface



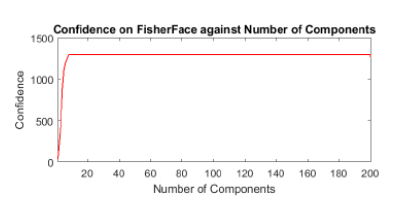
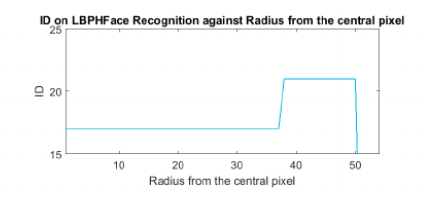


Fig. 2 ID and Conﬁdence for Eigenface



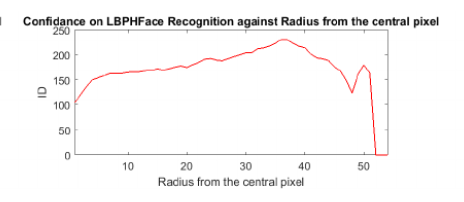
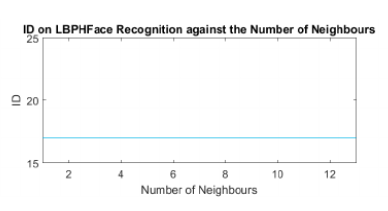


Fig. 3 ID and Conﬁdence for LBPA neighbours.



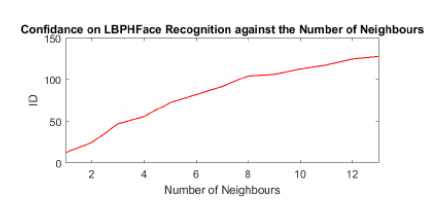
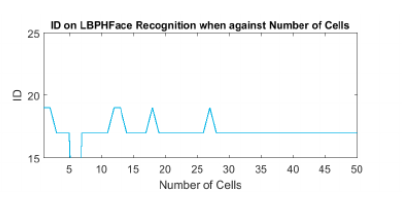


Fig. 4 ID and Conﬁdence for LBPA neighbours.



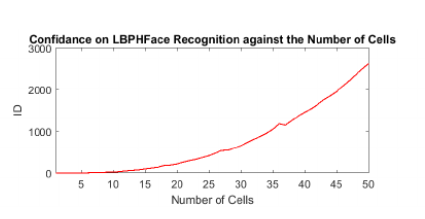


Fig. 5 ID and Conﬁdence for LBPA cells.

1. Face Database

We don't want to do a toy example here. We are doing face recognition, so you'll need some faceimages! You can either create your own database or start with one of the available databases, face-

rec.org/databases gives an up-to-date overview. Three interesting databases are:

AT&T Face database The AT&T Face database, sometimes also known as ORL Database of Faces,

contains ten di  
event images of each of 40 distinct subjects. For some subjects, the images were

taken at di  
event times, varying the lighting, facial expressions (open / closed eyes, smiling /

not smiling) and facial details (glasses / no glasses). All the images were taken against a dark

homogeneous background with the subjects in an upright, frontal position (with tolerance for

some side movement).

Yale Face database A The AT&T Face database is good for initial tests, but it's a fairly easy

database. The Eigenfaces method already has a 97% recognition rate, so you won't see any

improvements with other algorithms. The Yale Face database A is a more appropriate dataset

for initial experiments, because the recognition problem is harder. The database consists of

15 people (14 male, 1 female) each with 11 grayscale images sized 320 \* 243 pixel. There are changes in the light conditions (centre light, left light, right light), facial expressions (happy,

normal, sad, sleepy, surprised, wink) and glasses (glasses, no-glasses).

The original images are not cropped or aligned. I've prepared a Python script available in

src/py/crop\_face.py, that does the job for you.

Extended Yale Face database B The Extended Yale Face database B contains 2414 images of 38

di  
event people in its cropped version. The focus is on extracting features that are robust to

illumination, the images have almost no variation in emotion/occlusion/: : :. I personally think,

that this dataset is too large for the experiments I perform in this document, you better use

the AT&T Facedatabase. A rest version of the Yale Face database B was used in to see how

the Eigenfaces and Fisher faces method perform under heavy illumination changes. Used the same setup to take 16128 images of 28 people. The Extended Yale Face database B is the merge of the two databases, which is now known as Extended Yalefacedatabase B.

1. Discussion

Early attempts on Eigenface and Fisherface was disappointing since the LBPH alone recognised a face. By designing an application to test the algorithms, and after calibration with new data, both algorithms performed well. The tester applications also allowed accurate threshold settings. Another problem was people tilting their head when images taken for the data. This was ﬁxed with an application that dentists locations of the eyes and rotate the image to correct the oﬀ-set. It was noticed that some early-stage images in the data-set diﬀerent brightness’s. To resolve this, before taking an image, the brightness was averaged to prevent dark images. These changes to the system improved the performance noticeably.

1. Conclusion

This paper describes the mini-project for visual perception and autonomy module. Next, it explains the technologies used in the project and the methodology used. Finally, it shows the results, discuss the challenges and how they were resolved followed by a discussion. Using Haar-cascades for face detection worked extremely well even when subjects wore spectacles. Real time video speed was satisfactory aswell devoid of noticeable frame lag. Considering all factors, LBPH combined with Haar-cascades canbe implemented as a cost eﬀective face recognition platform. An example is a system to identify known troublemakers in a mall or a supermarket to provide the owner a warning to keep him alert or for automatic attendance taking in a class .

References

1. S. M. Metev and V. P. Veiko, *Laser Assisted Microtechnology*, 2nd ed., R. M. Osgood, Jr., Ed. Berlin, Germany: Springer-Verlag, 1998.
2. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification*, IEEE Std. 802.11, 1997.