Face Detection & Face Recognition Using Open Computer Vision

PRADHI ANIL KUMAR DAS

DECEMBER 26, 2019

INDEX

ABSTRACT

Identifying a person with an image has been popularized through the mass media. However, it is less robust to fingerprint or retina scanning. This report describes the face detection and recognition mini-project undertaken for the visual perception and autonomy module at Al Mulla Group. It reports the technologies available in the Open-Computer-Vision (OpenCV) library and methodology to implement them using Python. For face detection, Haar-Cascades were used and for face recognition Local binary pattern histograms was used. The methodology is described including flow charts for each stage of the system. Next, the results are shown including plots and screen-shots followed by a discussion of encountered challenges. The report is concluded with the authors' opinion on the project and possible applications.

Introduction

The following document is a report of the project. It involved building a system for face detection and face recognition using a classifier available in the open computer vision library(OpenCV). 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. The report begins with a brief history of face recognition. This is followed by the explanation of HAAR-cascades, Eigenface, Fisherface and Local Binary Pattern Histogram (LBPH) algorithms. Next, the methodology and the results of the project are described. Finally, a conclusion is provided on the pros and cons of the LBPH algorithm and possible implementation.

The History of Face Recognition

Face recognition began as early as 1977 with the first automated system being introduced by Kanade using a feature vector of human faces. In 1983, Sirovich and Kirby introduced the Principal Component **A**nalysis(PCA) for feature extraction. Using PCA, Turk and Pentland Eigenface was developed in 1991 and is considered a major milestone in technology.

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 Fisherface 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.

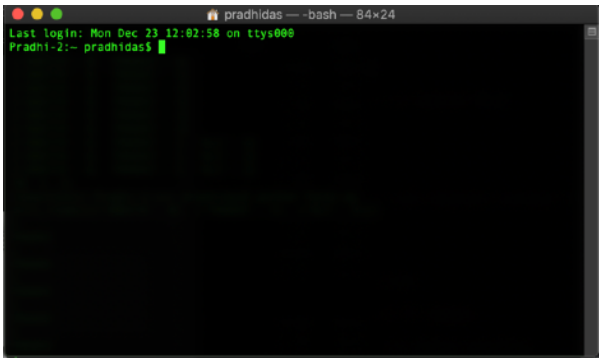
Viola and Jones introduced a face detection technique using HAAR cascades and ADABoost.

In 2007, a face recognition technique was developed by Naruniec and Skarbek using Gabor Jets that are similar to mammalian eyes.

In this project, HAAR cascades are used for face detection and LBPH are used for face recognition.

TOOLS USED

Sublime Text

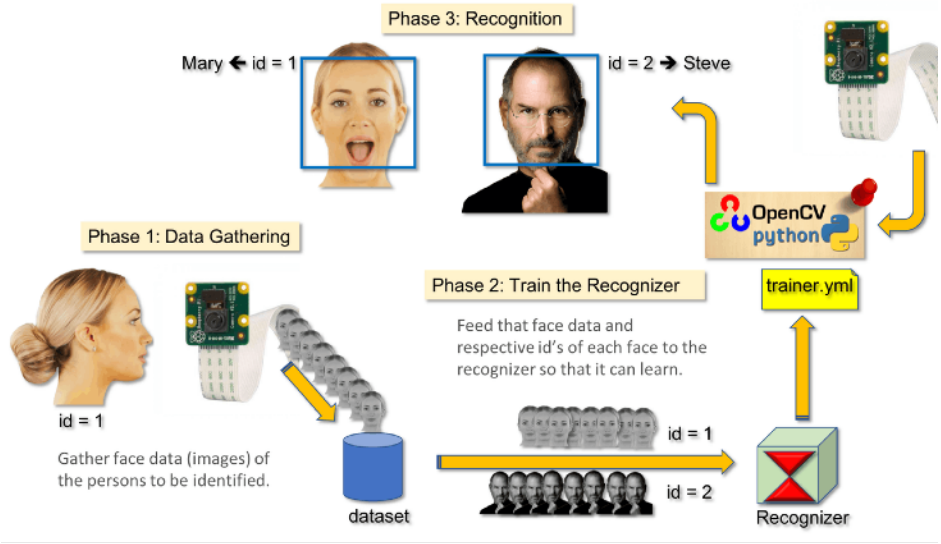


Three Phases

To create a complete project on Face Recognition, we must work on 3 very distinct phases:

- Face Detection and Data Gathering
- Train the Recognizer
- Face Recognition

The below block diagram resumes those phases:



Face Detection using Haar-Cascades

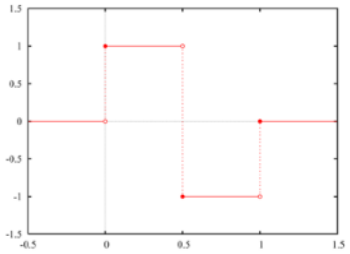
The problem of face recognition is all about face detection. This is a fact that seems quite bizarre to new researchers in this area. However, before face recognition is possible, one must be able to reliably find a face and its landmarks. This is essentially a segmentation problem and in practical systems, most of the effort goes into solving this task. In fact the actual recognition based on features extracted from these facial landmarks is only a minor last step.
There are two types of face detection problems:
1) Face detection in images and
2) Real-time face detection

In this project, **face detection in images** is used.

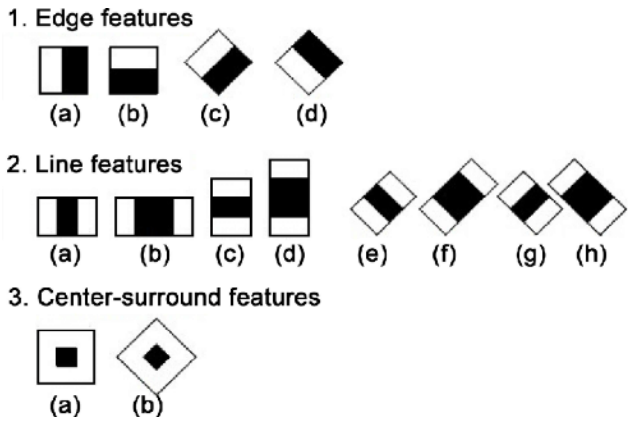
Most face detection systems attempt to extract a fraction of the whole face, thereby eliminating most of the background and other areas of an individual's head such as hair that are not necessary for the face recognition task. With static images, this is often done by running a across the image. The face detection system then judges if a face is present inside the window. Unfortunately, with static images there is a very large search space of possible locations of a face in an image.
There is another technique for determining whether there is a face inside the face detection system's window - using Template Matching. The difference between a fixed target pattern (face) and the window is computed and thresholded. If the window contains a pattern which is close to the target pattern(face) then the window is judged as containing a face.

A Haar wavelet is a mathematical fiction that produces square-shaped waves with a beginning and an end and used to create box shaped patterns to recognize signals with sudden transformations. By combining several wavelets, a cascade

can be created that can identify edges, lines and circles with different color intensities. These sets are used in Viola Jones face detection technique in 2001 and since then more patterns are introduced for object detection.To analyze an image using Haar cascades, a scale is selected smaller than the target image. It is then placed on the image, and the average of the values of pixels in each section is taken. If the difference between two values pass a given threshold, it is considered a match. Face detection on a human face is performed by matching a combination of different Haar-like-features. For example, forehead, eyebrows and eyes contrast as well as the nose with eyes. A single classifier is not accurate enough. Several classifiers are combined as to provide an accurate face detection system.



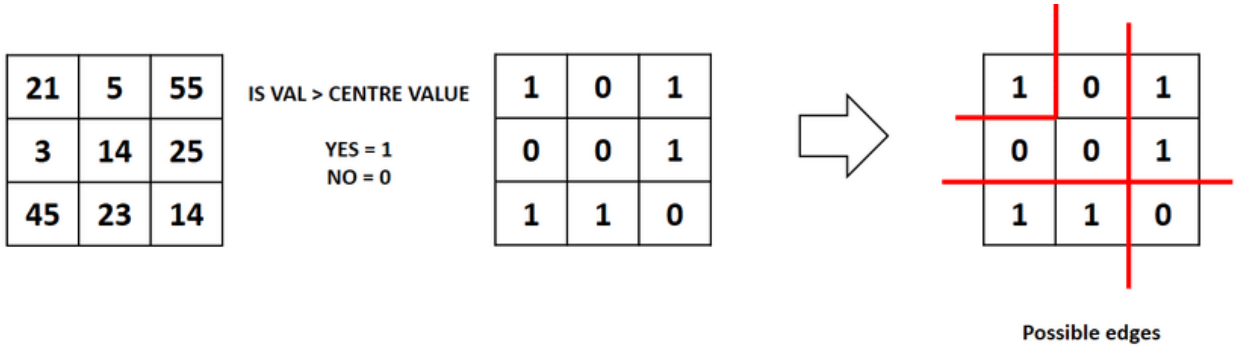
Haar wavelet



Face Recognition Using Local Binary Pattern Histogram (LBPH) Algorithm

Local binary patterns were proposed as classifiers in computer vision and in 1990 By Li Wang. The combination of LBP with histogram oriented gradients was introduced in 2009 that increased its performance in certain datasets [5].
For feature encoding, the image is divided into cells (4 x 4 pixels).
Using a clockwise or counter-clockwise direction surrounding pixel values are compared with the central. The value of intensity or luminosity of each neighbor is compared with the centre pixel. Depending if the difference is higher or lower than 0, a 1 or a 0 is assigned to the location.
The result provides an 8-bit value to the cell. The advantage of this technique is even if the luminosity of the image is changed as in figure 7, the result is the same as before. Histograms are used in larger cells to find the frequency of occurrences of values making process faster. By analyzing the results in the cell, edges can be detected as the values change. By computing the values of all cells and concatenating the histograms, feature vectors can be obtained.
Images can be classified by processing with an ID attached.
Input images are classified using the same process and compared with the dataset and distance is obtained. By setting up a threshold, it can be identified if it is a known or unknown face. Eigenface and Fisherface compute the dominant features of the whole training set while LBPH analyze them individually.

Local Binary Pattern generating 8-bit number



Increase Brightness yet, same results

42	10	110
6	28	50
90	46	28

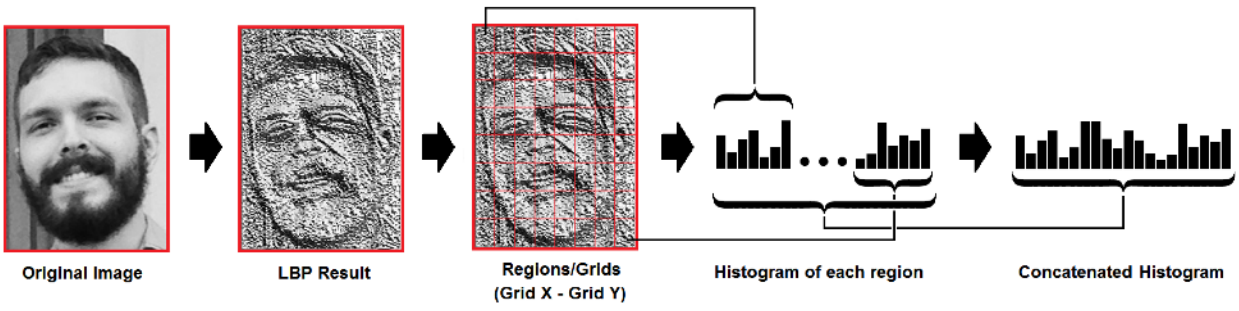
IS VAL > CENTRE VALUE

YES = 1
NO = 0

1	0	1
0	0	1
1	1	0

In this project, face recognition is done using template matching.

This is similar the template matching technique used in face detection, except here we are not trying to classify an image as a 'face' or 'non-face' but are trying to recognize a face. Whole face, eyes, nose and mouth regions which could be used in a template matching strategy.The basis of the template matching strategy is to extract whole facial regions (matrix of pixels) and compare these with the stored images of known individuals. Once again Euclidean distance can be used to find the closest match. The simple technique of comparing grey-scale intensity values for face recognition was used by Baron (1981). However there are far more sophisticated methods of template matching for face recognition. These involve extensive pre- processing and transformation of the extracted grey-level intensity values.



BRIEF OUT LINE OF THE IMPLEMENTED SYSTEM

Fully automated face detection of frontal view faces is implemented using a deformable template algorithm relying on the image invariants of human faces. This was chosen because a similar neural-network based face detection model would have needed far too much training data to be implemented and would have used a great deal of computing time. The main difficulties in implementing a deformable template based technique were the creation of the bright and dark intensity sensitive templates and designing an efficient implementation of the detection algorithm.

A manual face detection system was realized by measuring the facial proportions of the average face, calculated from 3-6 test subjects. To detect a face, a human operator would identify the locations of the subject's eyes in an image and using the proportions of the average face, the system would segment an area from the image

A template matching based technique was implemented for face recognition. This was because of its increased recognition accuracy when compared to geometrical features based techniques and the fact that an automated geometrical features based technique would have required complex feature detection pre-processing.

FACE RECOGNITION DIFFICULTIES

1. Identify similar faces (inter-class similarity)

2. Accommodate intra-class variability due to

2.1 head pose

2.2 illumination conditions

2.3 expressions

2.4 facial accessories

2.5 aging effects

3. Cartoon faces

FACE RECOGNITION USES

1. Accessing personal devices
2. Social robot interactions
3. Online purchases
4. Smart Advertising
5. Prevent Retail Crime
6. Aid Forensic Investigation
7. To find missing people

PROJECT

Face Trainer

faces.py | faces-train.py | labels.json

```python
import cv2   # importing opencv
import os    # importing os for path
import numpy as np  # importing the NumPy library
from PIL import Image  # importing the image library
import json  # importing json library

BASE_DIR = os.path.dirname(os.path.abspath(__file__))  # Absolute directory of the path of image
image_dir = os.path.join(BASE_DIR, "images")  # Image directory
# Import HAAR Cascades for face detection
face_cascade = cv2.CascadeClassifier('cascades/data/haarcascade_frontalface_alt.xml')

current_id = 0
label_ids = {} # Dictionary of names and ids
y_labels = []  # List of names and ids
x_train = []   # training list

for root, dirs, files in os.walk(image_dir):    # Traversal of the path directory
    for file in files:
        if file.endswith(".jpeg"):
            path = os.path.join(root, file)     # joining the file found with the root(name)
            label = os.path.basename(root).replace(" ","-").upper()  # extracting the name from directory

            if label in label_ids:
                pass
            else:
                label_ids[label] = current_id
                current_id += 1
            id_ = label_ids[label]
            print(label_ids)
            pil_image = Image.open(path).convert("L") # L-grayscale
            size = (550, 550)
            final_image = pil_image.resize(size, Image.ANTIALIAS)   # resize the image so that LBPH Recognizer can be trained
            image_array = np.array(final_image, "uint8")            # Converting the image to NumPy array
            faces2 = face_cascade.detectMultiScale(image_array)     # Detect the faces and store the positions

            for (x,y,w,h) in faces2:                                # Frames LOCATION X,Y,Width,Height
                if id_ not in y_labels:
                    ROI = image_array[y:y+h, x:x+w]
                    x_train.append(ROI)      # Append the NumPy array to the list
                    y_labels.append(id_)     # Append the ids to the list
print(y_labels)
recognizer = cv2.face.LBPHFaceRecognizer_create()   # LBPH Face recognizer object
with open("labels.json",'w') as f:
    json.dump(label_ids, f)

recognizer.train(x_train, np.array(y_labels))       # The recognizer is trained using the images
recognizer.save("trainer.yml")                      # Save the training data from the trainer
```

Face Detector and Recognizer

faces.py | faces-train.py | labels.json

```python
import numpy as np   # importing NumPy - a Python library used for scientific calculations
import cv2  # importing opencv
import json # importing json (JavaScript Object Notation)
# Import HAAR Cascades for face detection
face_cascade = cv2.CascadeClassifier('cascades/data/haarcascade_frontalface_alt.xml')
recognizer = cv2.face.LBPHFaceRecognizer_create()     # LBPH Face recognizer object
recognizer.read("trainer.yml")                        # Load the training data from the trainer to recognize the faces
labels = {"person_name": 2}                           # Declaring the name of the person as label
with open("labels.json",'r') as f:
    o_labels = json.load(f)                           # Loading the trained data
    labels = {v:k for k,v in o_labels.items()}
print(o_labels.items())

cap = cv2.VideoCapture(0)                             # Camera object
while(True):
    ret, frame = cap.read()                           # Read the camera object
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)    # Convert the Camera to gray
    faces2 = face_cascade.detectMultiScale(gray)      # Detect the faces and store the positions

    for (x, y, w, h) in faces2:                       # Frames LOCATION X,Y,Width,Height
        ROI_gray = gray[y:y+h, x:x+w]                 # The Face is isolated and cropped
        ROI_color = frame[y:y+h, x:x+w]

        id_, conf = recognizer.predict(ROI_gray)      # Determine the id of the photo
        if conf>40:
            print(id_)
            print(labels[id_])
        img_item = "img.png"
        cv2.imwrite(img_item, ROI_color)
        font = cv2.FONT_HERSHEY_TRIPLEX               # Font of the text
        name= labels[id_]
        color = (255,0,0)
        stroke = 2
        cv2.putText(frame, name, (x,y-15), font, 2, color, stroke, cv2.LINE_AA)     # Name of the face recognized

        color = (0,255,255) #BGR 0-255
        stroke = 2
        end_cord_x = x+w
        end_cord_y = y+h
        cv2.rectangle(frame, (x,y), (end_cord_x, end_cord_y), color, stroke)       # Reactangle around the face to show that face is detected
    # Display the resulting frame
    cv2.imshow('frame',frame)                         # show the captured image
    if cv2.waitKey(20) & 0xFF == ord('q'):            # Quit if the key is q
        break
# When everything done, release the capture
cap.release()
cv2.destroyAllWindows()

```

References

1. https://github.com/codingforentrepreneurs
2. https://www.codingforentrepreneurs.com/blog/install-opencv-3-for-python-on-mac/
3. https://kirr.co/o51uoz
4. https://kirr.co/0l6qmh
5. https://kirr.co/531875
6. https://stackoverflow.com/questions/45655699/attributeerror-module-cv2-face-has-no-attribute-createlbphfacerecognizer