SUMMER DESIGN PROJECT

A Report
on

HUMAN DETECTION AND RECOGNITION

Team members :
Arpit Solanki
Priyanka Singhal
Prashant Maurya
Krishnanunni Rajendran

Mentor :
Jignesh S. Bhatt

July 5, 2016

Indian Institute of Information Technology Vadodara

Abstract

In this project we have worked on the problem of human detection, face detection, face recognition and tracking an individual. Our project is capable of detecting a human and its face in a given video and storing Local Binary Pattern Histogram (LBPH) features of the detected faces. LBPH features are the key points extracted from an image which is used to recognize and categorize images. Once a human is detected in video, we have tracked that person assigning him a label. We have used the stored LBPH features of individuals to recognize them in any other videos. After scanning through various videos our program gives output like- person labeled as subject1 is seen in video taken by camera1, subject1 is seen in video by camera2. In this way we have tracked an individual by recognizing him/her in the video taken by multiple cameras. Our whole work is based on the application of machine learning and image processing with the help of openCV, an open source computer vision library. All the code is open sourced and available on github.

Contents

1 Introduction

The observation or monitoring of the activity, behavior and other information by a system which include several Closed Circuit Television (CCTV) cameras for observation and a set of algorithms to track a person is called Surveillance system. Technology has evolved a lot in the last few decades, previously there were no security cameras neither in Banks, Railway Station nor at other places. There were only security guards which protect these areas. Once the security cameras came into existence, it became easy to find people passing within the range of CCTV camera by simply searching through the videos recorded. Inventions increases people's expectations, although security camera reduces human effort but one has to search for an individual through entire video which takes a considerable amount of time. So people thought what if searching task can be accomplished by machine, it would save both human effort and time. A combination of machine learning with image processing is used to make a machine learn to recognize a person and track that person in the given footage. Our project is all about a system which have been designed to track human in the given videos. We have trained our machine for some people assigning label to each, so that when he/she have appeared in a video or more than one videos he/she have been recognized by our system assigning him/her label. In this way a person have been recognized and tracked in a given videos.

2 Problem Definition

The input is given in the form of a video or a sequence of videos. The main objective of our project is to detect and recognize person and track them in a given video sequence. We have considered a few constraints in the interest of providing a successful output. Firstly, our work is mostly restricted to homogeneous domain i.e. the videos provided by the user must be shot with a homogeneous background *i.e.* plain background. Secondly, our system is successful in detecting humans in majority of the cases but it recognizes only those individuals for which it is already trained. So if there is need of recognizing new individual for which machine is not trained, machine must be trained for that individual using our program.

There are some assumption taken by us. We have assumed that in most of the cases the person is facing the camera so that his/her face is successfully detected and recognized by our program.

There are certain major challenges that we faced while doing this project. Firstly, detecting humans in all possible postures like- standing, sitting, lying etc was a great problem. Secondly, detecting humans in all possible orientations *i.e.* when their front side is facing the camera, when their back side is facing the camera, when they are facing the cameras sideways etc. Reducing the false positive rate to minimum possible value is still a greater challenge.

3 Project Methodology

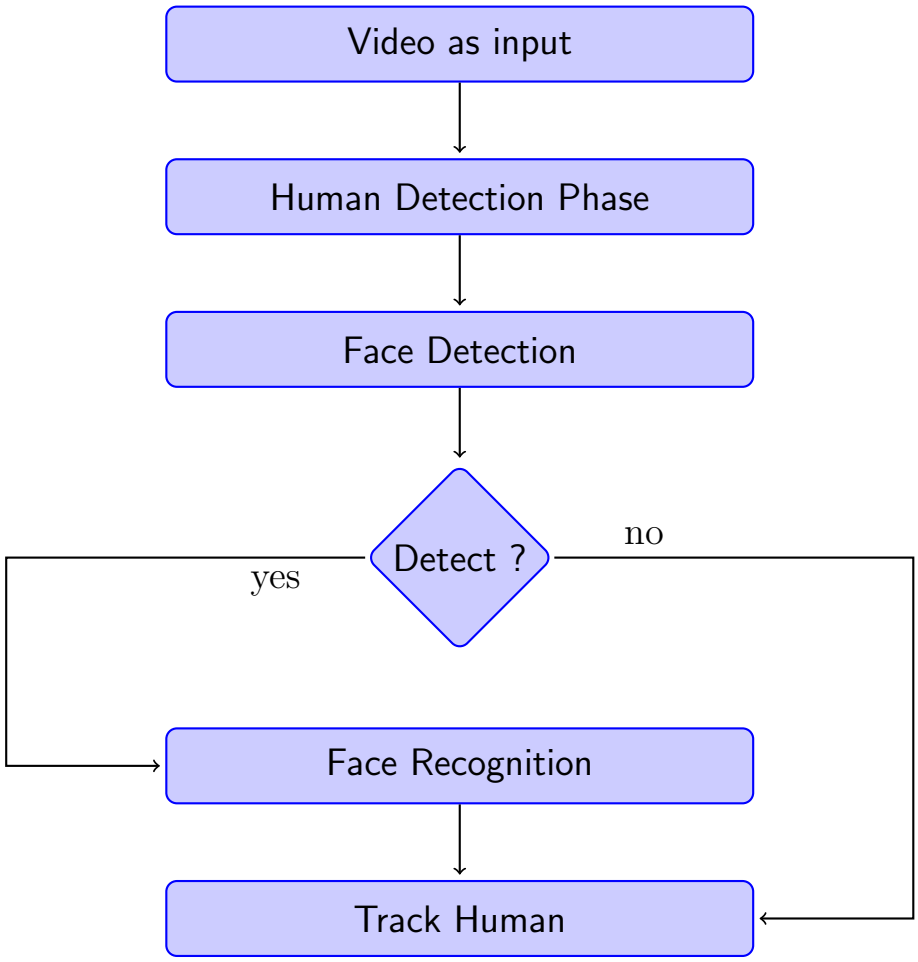
In this section, we will discuss about the different phases through which the system goes through before providing the output to the user. This provides an overview about how the system works and how the output is provided. The approach taken by the program is described using flow chart in section 3.1 approach.

3.1 Approach

This section gives both overview and detailed information about our project. The flow chart shown on the next page is overall mapping of our project. The flow chart gives us idea about flow of our project and the stages involved in it.

As shown in the flow chart input to our program is a video or a sequence of videos. The program works in three stages the human detection stage, the face detection stage and the face recognition stage. In the first phase the program tries to detect humans in the input videos. If a human is detected, the program draws a rectangle around them and tracks the human in the video. In the second phase our program tries to detect the faces in the given video. Detection of faces in a given video is independent of detection of human in a video. Once face is detected by the program, the next phase is to correctly recognize the person. We had already trained our machine for certain number of people by extracting faces from some videos. Now with the help of trained machine our program tries to recognize individuals in the video. After face recognition, whether it is successful or not it goes to next step i.e. tracking human.

FLOW CHART



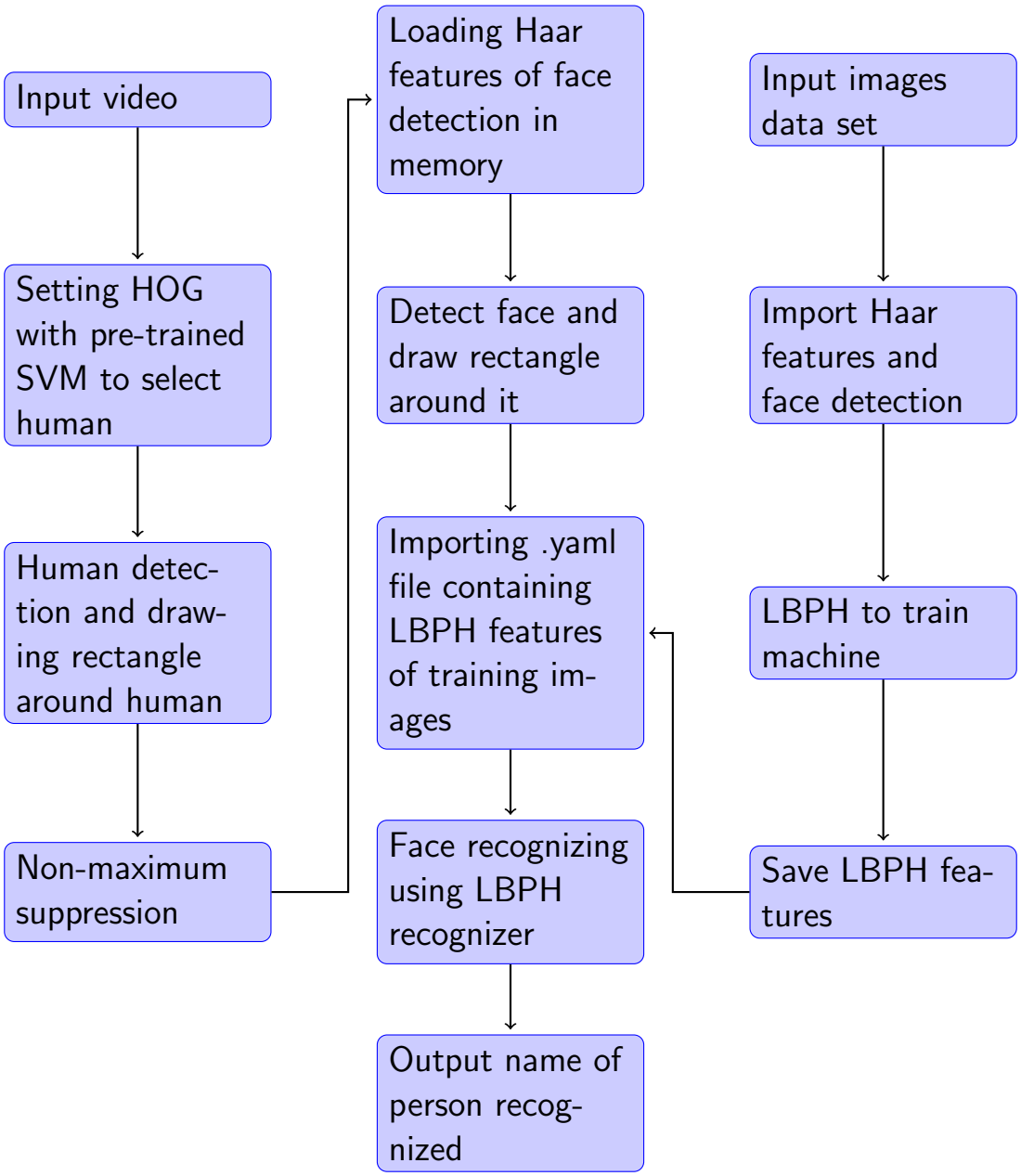
The data flow diagram gives a description about the various algorithms used in our project to manipulate input video for getting result. Our program works in three main phases human detection, face detection, face recognition as discussed earlier, but let's have a closer look to the processes and algorithms involved in these three phases.

For the first phase of human detection, the program takes a video or a sequence of videos as input. In the next step, it sets up a Histogram of Oriented Gradients (HOG)[1] descriptor which is a feature descriptor used for detecting objects with the help of a Support Vector Machine (SVM)[2]. HOG along with SVM is used in our program with the help of openCV library to detect humans. As the program then detects humans and a rectangle is drawn around the human. In some cases our program draws more than one rectangle for single person because of false positive cases, to minimize such false positive cases, in the next step our program uses non-maximum suppression to select a rectangle based on maximum overlapping criteria. The first phase of human detection gets completed here.

Now, the program moves on to the second phase i.e. Face detection. For the detection of faces the program loads the Haar features[3] into the memory, which contains features of faces and using these features the program tries to detect human faces. After detection of human faces the program show detected faces in a separate window. Face detection is independent of human detection. There are certain cases in which human is not detected but face is detected, it happens because in some video only a part of human body is visible not complete body this phase gets completed here.

After the face detection phase, comes the phase of face recognition. But for this phase there is a prerequisite, to recognize a person firstly, we have to train our machine for certain images of that person. Here training is done using Haar cascades and Local Binary Pattern Histograms (LBPH)[4]. These algorithms are explained in the section 3.2. Firstly Haar features are used to detect faces in video frames, now each detected face is treated as a data set to train our machine and LBPH features corresponding to each faces are saved in a file. In our case it is yaml file.

DATA FLOW DIAGRAM



Face recognition is done by importing the saved features of yaml file into the memory, with the help of LPBH face recognizer program tries to recognize faces using LBPH features. Once faces are recognized the person is given the label of recognized face. So each recognized person is assigned a label. The name or label of recognized person is given as output along with the confidence factor. Confidence factor is a parameter whose value is zero if the images in training set and and test case are same, higher the value of confidence factor lower the given image is similar to that person. We have set threshold value of confidence factor, above which we can say the given image is not similar to the labeled person.

Thus, our program detects human, detect faces, recognize faces and track human by recognizing this individual has passed through this camera. So we have output like subject1 has passed through camera1 and he/she also passed camera2 and so on. In this way we have tracked person with help of multiple cameras.

3.2 Algorithms used in our project

3.2.1 Support Vector Machine (SVM) classifier

SVM is a set of Supervised learning method used for classification, regression. SVM can efficiently perform both linear and non-linear classification. When it is not possible to classify data linearly in the same dimensional space in which data point resides, SVM goes into higher dimensional space so that classification can be done, the same thing is shown in figure(1). Image at the left in figure(1) shows that the data to be classified is nonlinear and image at right of figure(1) shows that on going to higher dimension same data can be classified linearly. For non linear classification, regression and other task SVM construct a hyperplane or a set of hyperplanes in a high or infinite-dimensional spaces.

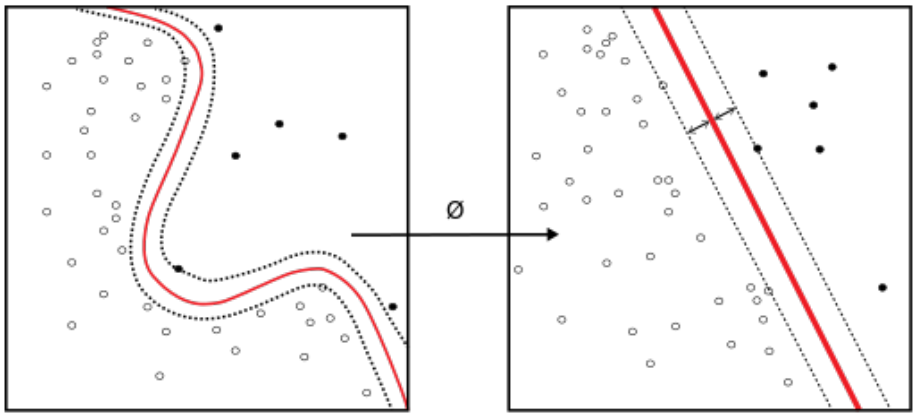


Figure 1: Nonlinear classification: To classify nonlinear data linearly in higher dimension.
source: Wikipedia[5]

Linear classification is a bit simpler task as compared to nonlinear,

but the resultant accuracy of linear classification is not good large data sets. Linear classification can be explained by equation 1.

Given training data (x_i, y_i) for $i = 1...N$, with $x_i \in R^d$ and $y_i \in \{1, -1\}$, learn a classifier f(x) such that

$$f(x_i) = \begin{cases} \geq 0 & y_i = +1 \\ < 0 & y_i = -1 \end{cases} \tag{1}$$

i.e. $y_i f(x_i) > 0$ for a correct classification.

3.2.2 Histogram Oriented Gradient (HOG)

It is one of the feature descriptor used in computer vision and image processing. Histogram of oriented gradients descriptor can be described by the distribution of intensity gradients or edge directions in an image. The image is divided into small connected region called as cells. A cell can contain several pixels and for each of the pixel a histogram of gradient is made. The descriptor contains histogram of gradient of each and every pixel. For better accuracy the HOG is contrast-normalized, this is done by calculating intensity over a larger area several cells known as block, then this value is used to normalize all cells within that block. The normalized result gives better performance on variation in illumination and intensity. HOG descriptors has some advantages over other descriptors such as it is invariant to geometric and photometric transformations except object orientation. It is particularly suited for human detection in images.

3.2.3 Haar Cascade and Cascade classifiers

Haar cascade is explained by two things haar like features and cascade classifiers. Haar-like feature considers adjacent rectangular regions at a specific location in a detection window, sums up the pixel intensities in each of the regions and then calculates the difference between these sums. Due to its intuitive similarity with haar-wavelets these features get their name as haar-features. These features are used for object detection. In the detection phase the window is moved over the input image and for each-subsection, haar-like features are calculated .This difference is then compared with a learned threshold value to detect the presence of an object. But for accurately detecting an object, a large number of haar-like features are required. To implement this algorithm, we need a large number of positive and negative samples where positive samples indicate the presence of a object and negative samples indicate the absence of the same object to train the classifier. For this, we need to apply each and every feature on all the training images and for each feature calculate the best threshold value which will classify the images as positive or negative. We select the features with minimum error rate, and this process is continued until required accuracy or required number of features are found.

These features cannot detect the object on their own but when used together they form a strong classifier. So let's consider a scenario in which the task is to detect a face. Major part of the image is a non-face region. For this, the concept of cascade of classifiers was introduced. Instead of applying all the features on a window at once, group the features into different stages and apply them one-by-one. So if a region fails in the first stage itself, we don't need to apply the remaining classifiers. Only on those regions which pass the first stage,

we apply the second stage and so on. Thus, the region which passes all these stages contains the object or in this case a face.

3.2.4 Linear Binary Pattern Histogram (LBPH) face recognizer:

The basic theme on which Local Binary Pattern works is to achieve a local structure of an image by comparing each pixel with it's neighborhood. A pixel is taken as center and it is compared with all it's neighborhood pixels if the value of the neighborhood pixel is greater than the value of pixel at center then it is assign value 1 otherwise 0. In this way shifting the center each time we end up with an binary matrix of image containing relative values. For example if we are using 3×3 neighborhood like the matrix shown in example(1), we have 8 neighbors and after comparison each time we will end up with 8 digit binary number corresponding to each center, for example(1) shown below comparing the value at center to neighborhood values results in binary matrix with values 01001110. In this way for 8 digits there are 2^8 combinations known as Local Binary Patterns.

Example 1:

0	8	3
4	7	1
12	9	13

$\xrightarrow{Threshold}$

0	1	0
0		0
1	1	1

A more formal description of the LBP operator can be given as:

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

where in equation(2), (x_c, y_c) as central pixel with intensity i_c ; and i_p being the intensity of the the neighbor pixel and s is the sign function defined as:

$$s(x) = \begin{cases} 1 & if \geq 1 \\ 0 & else \end{cases} \tag{3}$$

The value of i_c =7 in example(1) and after the application of equation(2) and equation(3) on the given matrix, we got binary value=01001110 corresponding to center.

4 Result

We have already discussed the program methodology and how the program works in the previous sections. In this section, we have provided the details about the results obtained by us while using this program against some of the test cases. We have explained the output of each and every test case using the screen shots of the output provided by our program. While making this project, we faced a lot of challenges and we have countered majority of these problems. There are some flaws in the output provided by the program but we have tried to minimize it as much as possible. The program requires a lot of computations so the output of the program is also system dependent. We have tried the program on two systems which were available with us. One of which had the following specifications: AMD A4 2.2 GHz dual-core processor and 4GB RAM. And the other system had the following specifications: Intel i5 2.4GHz Quad-core processor 8GB RAM. For an input video of 30 Frames per second(fps) the 1st system gave an output speed of just 4-5 fps. On the other hand, the 2nd system gave an output with a speed of 10-12 fps.

As mentioned before, we have considered a constraint that the input video must be shot in a homogenous domain. And also that the system will recognize only those faces for which it is initially trained. So we have the trained the machine for two people. One of them is labeled as 1 and the other one is labeled as 2. We have provided the output for four test videos where each of the test videos represent different scenarios. We have tried to represent the different stages of human detection, face detection and face recognition in the results provided by the program. Following are the screen shots of different stages of the output provided by the program when the test cases were given as input videos.

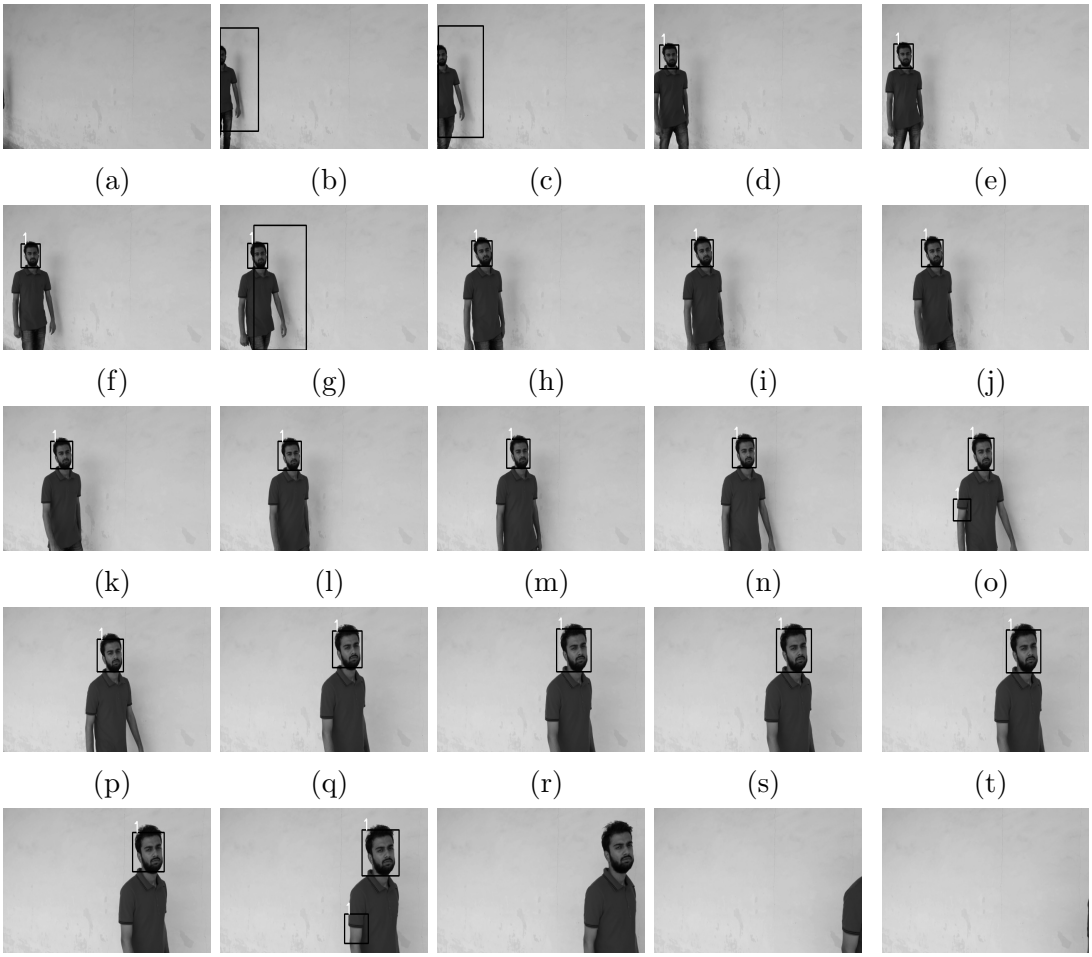


Figure 2: Output of video 1: single person video; fig. o,v are false face detected cases, no face detected in fig. w, in many images human is not detected due to absence of full human body.

The screen shots of the output provided by the program for the first video is shown in figure 2. In this video, only one person will be seen. We have trained the machine to recognize him and label him as 1. Initially, the frame is empty as seen in figure 2(a). When the person enters the frame, the program correctly detects the presence of a person and draws a rectangle around him. The program has correctly recognized the presence of person 1 as seen in figure 2(c) and it continues to track his face and correctly label it as 1. A false positive is triggered due to the sleeve of person 1 and the program has incorrectly detected it as a face and labeled as 1 as it can be seen in figure 2(o). Then again, the program has removed the false positive and correctly recognized person 1 in the upcoming figures till the person 1 goes out of the frame in figure 2(y). In most of the cases human body is not detected because full human body is not present in those frames.

The screen shots of the output provided by the program for the second video are given in figure 3. Similar to video 1 in this video too, only one person will be seen. We have trained the machine to recognize him and label him as 2.Initially, the frame is empty seen in figure 3(a). As the person enters the frame, the program correctly detects the presence of a person and draws a rectangle around him. And it continues to detect the person in the upcoming frames as it can be seen in the figures. The program recognizes the person 2 and labels him for the first time in the figure 3(m). The program continues to correctly label the face of person 2 as 2 till the person moves out of the frame.

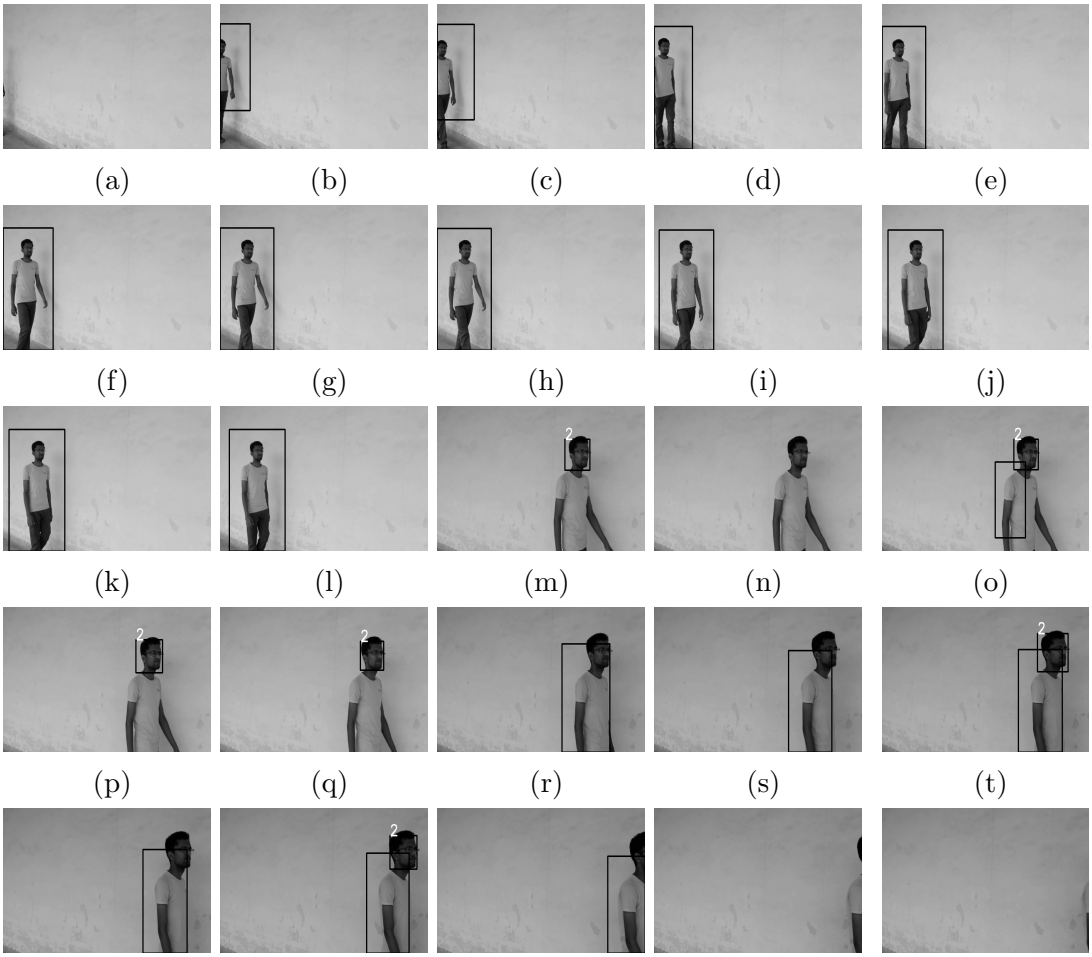


Figure 3: Output of video 2: single person video; no face detected in many images due to unclear face, human is detected in most of cases due to presence of full human body.

The screen shots of the output provided by the program for the third video are given in figure 4. Unlike previous videos, person 1 and person 2 both appear in this video. But they enter the frame one after the other and not together. Initially the frame is empty as seen in figure 4(a). Then the person who is labeled as 1 in our training samples enters the frame. In figure 4(d), the program has recognized the presence of person 1 for the first time and has even successfully detected the face of person 1. Since in the frame the entire body of person 1 is visible the program has correctly drawn a rectangle around him as seen in figure 4(e). The system is successful in detecting the face of person 1 and labeling it as 1 till the time he is present in the frame. Although, a false positive was detected by the system in figure 4(j) where the system false fully detects the sleeve of person 1 as a face. In the same frame, since the person 2 also enters the frame a rectangle is drawn around him. Till figure 4(l), the presence of a person is correctly detected by the program and is shown by drawing a rectangle around that person. Now in figure 4(m), we see that the person 1 is completely out of frame and person 2 is correctly detected. The program was successful in detecting a face in figure 4(n) and has correctly labeled it as 2. The program then again fired a false positive as we can see in figure 4(q) it has incorrectly labeled person 2 as 1. In figure 4(r),the program has recognized and labeled the person 2 correctly as person 2 rectifying the false positive. The program continues to label person 2 correctly till the person goes out of the frame in figure 4(y). In most of the cases human body is not detected because full human body is not present in those frames.

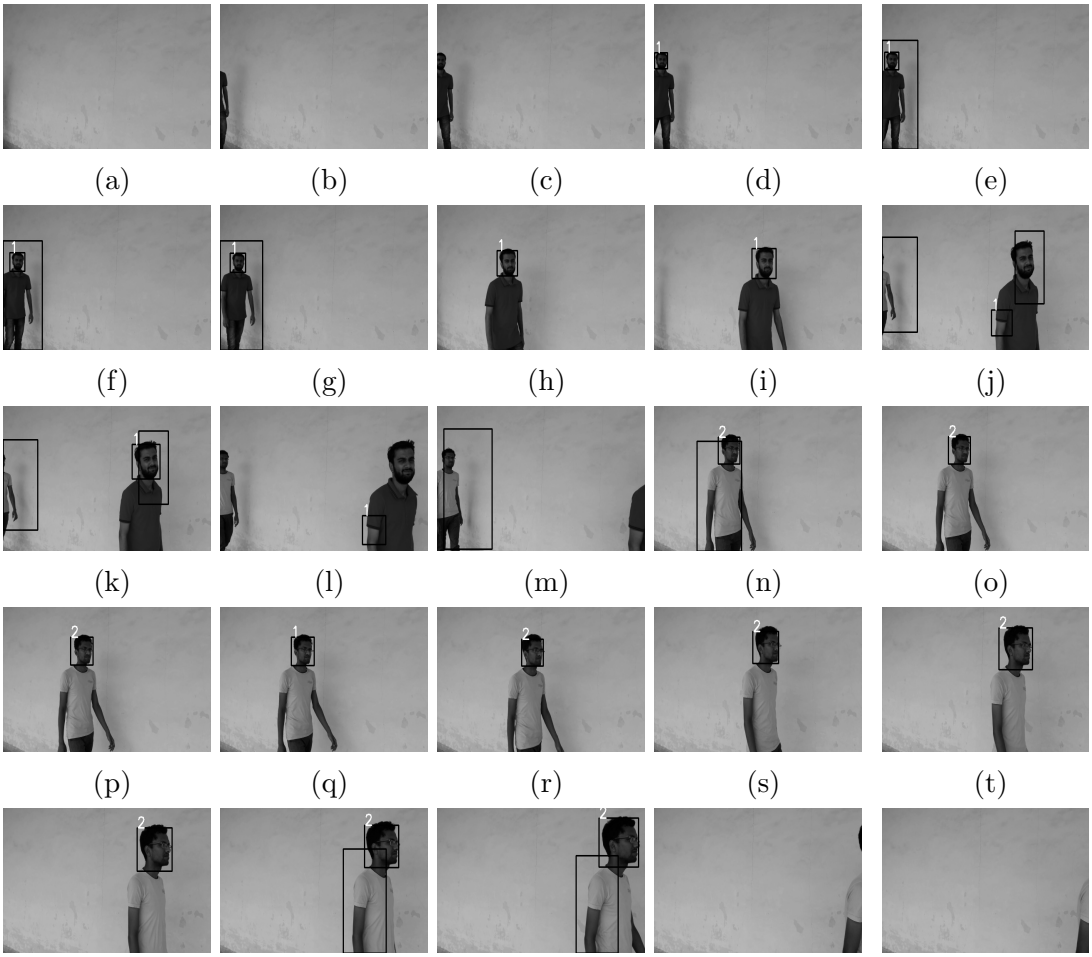


Figure 4: Output of video 3: multiple person video, walking one after other; false face detected in fig. j,l, fig. q is wrongly detected as subject 1, human is not detected in most of cases due to absence of full human body.

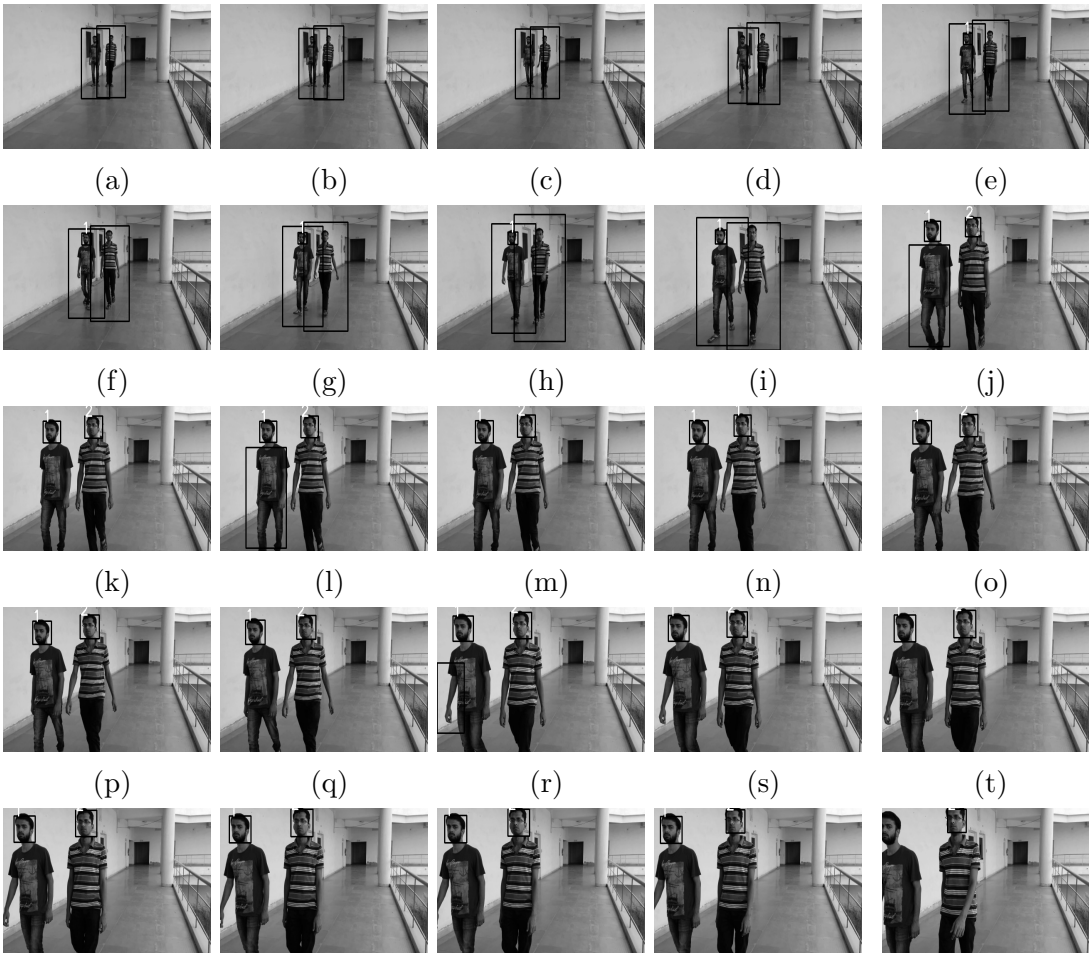


Figure 5: Output of video 4: multiple person video, walking together; no face detected in fig. a-d due to larger distance from camera, human is detected in stating fig. a-i due to presence of full human body.

The screen shots of the output provided by the program for the video 4 is shown in figure 5. In this video both person 1 and person 2 appear together in the same frame. We can see here since the first frame, the program has correctly detected both the persons. It has drawn a rectangle around both the humans as it can be seen in the figures given above. As they move in towards the camera the rectangles drawn around them also move with them in one way tracking them. In figure 5(e), the program has successfully recognized person 1 and labeled him as 1. And it continues to track both of them till finally recognizes the person 2 also as seen in figure 5(j). In this and the following figures, it can be seen how the program has successfully recognized both the persons and has correctly labeled them. As it can be seen in figure 5(n), the program false fully recognized the person 2 as person 1 and thus labeled person 2 also as 1. But in the following figures the program has correctly recognized person 1 as 1 and person 2 as 2 and it continues to recognize them till the last frame. In this video we get better human detection because in many frames full body of human is present.

5 Conclusion

The project mainly focuses on recognizing and tracking a person in a video. The program takes a video or a sequence of videos as input and outputs the names of the people who are seen in the videos. The assumption is that the videos are shot in a homogeneous environment. Another limitation that the user has to face is that he should train the machine beforehand, only then will the program recognize the person. Another program named train.py has been provided using which the user can train the machine for the people that are to be tracked. The program needs some processing time so for a input video of 30 frames per second the output comes out to be a video of 10-15 fps i.e. the program is system dependent. The program tries to avoid the false positives as much as possible but for certain textures it still becomes confused. The program successfully detects the presence of a person in a video and is also capable in detecting the faces of these people. The program uses the pretrained model to recognize humans found in a video and if the program succeeds in recognizing the person the name of the person is outputted on the terminal thus helping the user to track any person given a sequence of videos.

We are planning to add the feature that the program itself learns from the input video when it founds a new person and label it with a label. So when the next time, the person is seen in the video or any future videos the program can recognize that person on its own. With this feature ,the user doesn't need to train the machine but it will train by itself. This will make the life of the user much easier while using this program. The focus will also be on reducing the amount of false positives and making the program robust for any kind of environment.

6 References

References

[1] Navneet Dalal and Bill Triggs. Histogram of oriented gradients for human detection, 2005.

[2] http://docs.opencv.org/3.1.0/d4/db1/tutorial_py_svm_basics.html

[3] http://docs.opencv.org/master/d7/d8b/tutorial_py_face_detection.html

[4] Ahonen, T., Hadid, A., and Pietikainen, M. Face Recognition with Local Binary Patterns. Computer Vision - ECCV 2004 (2004), 469481.

[5] https://en.wikipedia.org/wiki/Support$_v ector_m achine/media/File : Kernel_M achine.s$

7 Appendix

In this section we are dealing with the topics mentioned below:

1. basic transformation on images
2. working with videos
3. argument parsers python
4. background subtraction openCV
5. non max suppression openCV
6. face recognition openCV
7. face detection using haar cascades
8. face recognition using scikit-learn

7.1 basic transformation on images openCV

In this subsection we talk about applying basic transformation like changing colorspaces,rotating an image etc on images. changing colorspaces: openCV provides a simple interface for changing colorspaces. In the sample code given below we use cv:: cvtcolor(image, *colorspace)

```
>>> import cv2
>>> image = cv2.imread("image.jpg")
>>> gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
>>> cv2.imshow("gray image",gray)
>>> cv2.waitkey(0)
```

Rotating an image: To rotate an image we have to first find a transformation matrix for this.opencv comes with a function named cv::getRotationMatrix2D (Point2f center, double angle, double scale). Now we have applied this matrix to image using function
cv::warpAffine(src, M, dsize[, dst[, flags[, borderMode[, borderValue]]]]) in the sample code given below.

```
>>> import cv2
>>> img = cv2.imread('messi5.jpg',0)
>>> rows,cols = img.shape
>>> M = cv2.getRotationMatrix2D((cols/2,rows/2),90,1)
>>> dst = cv2.warpAffine(img,M,(cols,rows))
```

Finding and drawing contours in an image:Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. OpenCV provides a function to find contours and draw contours. In the sample code given below we first find contours on an image then we draw it on black image.

```
>>> import numpy as np
>>> import cv2
>>> im = cv2.imread('test.jpg')
>>> imgray = cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
>>> ret,thresh = cv2.threshold(imgray,127,255,0)
>>> im2, contours, hierarchy= cv2.findContours(thresh, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
>>> cv2.drawContours(img, contours, -1, (0,255,0), 3)
>>> cv2.imshow("window",img)
>>> cv2.waitkey(0)
```

7.2 Working with videos

Videos are basically sequence of frames or images. OpenCV provides a simple interface to read and write videos. To read a video first we have to create an instance of Video Capture class. As mentioned in the sample code given below we first create an instance of Video Capture class then we read each frame in sequence.

```
>>> import numpy as np
>>> import cv2
>>> cap = cv2.VideoCapture(0)
>>> while(True):
 .    # Capture frame-by-frame
 .    ret, frame = cap.read()
  .
 .    # Our operations on the frame come here
 .    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
  .
 .    # Display the resulting frame
 .    cv2.imshow('frame',gray)
>>>      if cv2.waitKey(1) & 0xFF == ord('q'):
 .     break
>>> cap.release()
>>> cv2.destroyAllWindows()
```

7.3 Argument parser python

argparse library provides simple functions to create user friendly command line interfaces.Here we will talk about a simple example about using argparse library.For this we first have to create an instance of ArgumentParser class then we will add argument(to be given to program) with certain parameters.As mentioned in the code we can add additional parameters like what is the type of argument,is it a required argument etc.

```
>>> import argparse
>>> parser = argparse.ArgumentParser(description='Process some integers.')
>>> parser.add_argument('integers', metavar='N', type=int, nargs='+',
    help='an integer for the accumulator')
>>> parser.add_argument('--sum', dest='accumulate', action='store_const',
      const=sum, default=max, help='sum the integers (default: find the max)')
>>> args = parser.parse_args()
>>> print(args.accumulate(args.integers)
```

7.4 background subtraction openCV

Background subtraction means removing the background from an image while saving the foreground to extract foreground's information. opencv provides implementation of some algorithms regarding background subtraction.One of the popular algorithm for background subtraction is "Gaussian Mixture-based Background/Foreground Segmentation Algorithm".It was introduced in the paper "An improved adaptive background mixture model for real-time tracking with shadow detection" by P.KadewTraKuPong and R. Bowden in 2001.To use it we have to create an object of function cv2.BackgroundSubtractorMOG and then apply it to the frame. As shown in the code below we first read a video and for each frame we apply background subtraction as mentioned above.

```
>>> import numpy as np
    >>> import cv2
    >>> cap = cv2.VideoCapture('vtest.avi')
    >>> fgbg = cv2.createBackgroundSubtractorMOG()
    >>> while(1):
     . ret, frame = cap.read()
     .   fgmask = fgbg.apply(frame)
     .   cv2.imshow('frame',fgmask)
     .   k = cv2.waitKey(30) & 0xff
   >>> cap.release()
    >>> cv2.destroyAllWindows()
```

7.5 Non maximum suppression

This algorithm is used just after phase of human detection and face detection, when rectangles are drawn around human body or face. In many cases we found that there are a bunch of rectangle around human body and faces, to remove false drawn rectangles and get exactly one rectangle non maximum suppression is used. Non maximum suppression works on the principle of overlapping criteria. Overlapping threshold value is set and then overlapping parameter is calculated for all the rectangles, the rectangle passing threshold is accepted and drawn.

7.6 face recognition openCV

Face recognition is an easy task for humans.experiments shows that even a three days old baby can distinguish between known faces so a computer can also do it with no difficulties.The very basic approach for face recognition is based on detecting geometric features of face and then comparison of those features with different approaches. OpenCV provides implementation of many face recognition algorithms. Here we will explain Local Binary Patterns Histograms (LBPH) face recognition algorithm. The sample code is given below:

```
#import necessary modules
#!/usr/bin/python
import cv2
import os
import sys
import numpy as np
from PIL import Image
import imutils
#haar cascade file to detect faces
cascadePath = "haarcascade_profileface.xml"
#cascade classifier object to detect faces
faceCascade = cv2.CascadeClassifier(cascadePath)
#LBPH face recognizer for face recognition
recognizer = cv2.face.createLBPHFaceRecognizer()
# function to get labels for training images
def get_images_and_labels(path):
i=0
image_paths = [os.path.join(path, f) for f in os.listdir(path) if not f.endswith('.sad')]
images = []
labels = []
for image_path in image_paths:
image_pil = Image.open(image_path).convert('L')
image = np.array(image_pil, 'uint8')
image = imutils.resize(image, width=min(500, image.shape[1]))
nbr = int(os.path.split(image_path)[1].split(".")[0].replace("subject", ""))
# here detecting faces from images to append to training set
faces = faceCascade.detectMultiScale(image)
for (x, y, w, h) in faces:
images.append(image[y: y + h, x: x + w])
# cv2.imwrite("subject02."+str(i)+".jpg",image[y: y + h, x: x + w])
```

```
# i=i+1
labels.append(nbr)
cv2.imshow("Adding faces to traning set", image[y: y + h, x: x + w])
cv2.imshow('win',image[y: y + h, x: x + w])
cv2.waitKey(50)
return images, labels
# provide path to training images as argument from command line
path = sys.argv[1]
# getting labels and image array from training images
images, labels = get_images_and_labels(path)
cv2.destroyAllWindows()
#now training the face reconizer with training images
recognizer.train(images, np.array(labels))
#the following function will save a yaml file that will contain trained data
#this file can be reused for recognizing faces
#To load the file we can use the function recognizer.load(path/to/yaml/file)
recognizer.save("model.yaml")
#reading the test image
im = cv2.imread("test.jpg")
#now predicting the test image label with predict function of
#LBPHfacerecognizer class
#function outputs is a tuple containing predicted label and confidence value for
#given image
#confidence value provides measurement of matching of given label
#if it low then given test image is matched with good number of geometric features
#of that face
predict_tuple = recognizer.predict(im)
predicted_label,predict_conf = predict_tuple
print(predicted_label)
print(predict_conf)
```

7.7 Face detection using haar-cascades

This code was used by us to learn more about haar-features and about cascade classifiers. A haar-like feature sums up the pixel intensities in adjacent rectangular regions and calculates the difference between these sums. By running these features on a lot of positive and negative samples we compute the best threshold value which signifies the presence or the absence of an object. In order to detect the faces, a large number of these features are required. And since major part of the image is a

non-face region and a lot of computation is wasted. For this, the concept of cascade of classifiers was introduced. So now, if a particular region clears the first stage of classifiers only then will the region go through the second stage and so on. So the region which passes through all the stages contains a face.

```
import numpy as np
import cv2

#importing the cascade classifiers for detecting face and eyes

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')

#Taking the input as a image and converting the image to grayscale

img = cv2.imread('sachin.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#Storing the co-ordinates where faces are detected
#Drawing a rectangle around these faces
#Trying to detect eyes in these faces

faces = face_cascade.detectMultiScale(gray, 1.3, 5)
for (x,y,w,h) in faces:
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]
    eyes = eye_cascade.detectMultiScale(roi_gray)

#Drawing a rectangle around these eyes when detected

for (ex,ey,ew,eh) in eyes:
    cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)
cv2.imshow('img',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

7.8 face recognition using scikit-learn

The following code does face recognition by using Eigen faces and Support Vector Machines (SVM). We used this code to try and learn how face detection is done and the major complications faced while doing face recognition. The code downloads a database of images and then it stores all the images in the form of matrix .And then divides the data using 25 percentage of the data as testing samples and the remaining 75 percentage data as the training set .After that unsupervised feature extraction is performed on the training set. After that, an effort is made to train an SVM classification model. Then the classifier is used on the test data set to predict the names of the people in the test data set.

```
from __future__ import print_function

from time import time
import logging
import matplotlib.pyplot as plt

from sklearn.cross_validation import train_test_split
from sklearn.datasets import fetch_lfw_people
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import RandomizedPCA
from sklearn.svm import SVC


print(__doc__)

# Display progress logs on stdout
logging.basicConfig(level=logging.INFO, format='%(asctime)s %(message)s')
# Download the data, if not already on disk and load it as numpy arrays
lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.4)
# introspect the images arrays to find the shapes (for plotting)
n_samples, h, w = lfw_people.images.shape
# for machine learning we use the 2 data directly (as relative pixel
# positions info is ignored by this model)
X = lfw_people.data
```

```
n_features = X.shape[1]
# the label to predict is the id of the person

y = lfw_people.target
target_names = lfw_people.target_names
n_classes = target_names.shape[0]

print("Total dataset size:")
print("n_samples: %d" % n_samples)
print("n_features: %d" % n_features)
print("n_classes: %d" % n_classes)

# Split into a training set and a test set using a stratified k fold
# split into a training and testing set

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42)

# Compute a PCA (eigenfaces) on the face dataset (treated as unlabeled
# dataset): unsupervised feature extraction / dimensionality reduction

n_components = 150
print("Extracting the top %d eigenfaces from %d faces"
      % (n_components, X_train.shape[0]))
t0 = time()
pca = RandomizedPCA(n_components=n_components, whiten=True).fit(X_train)
print("done in %0.3fs" % (time() - t0))
eigenfaces = pca.components_.reshape((n_components, h, w))
print("Projecting the input data on the eigenfaces orthonormal basis")
t0 = time()
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
print("done in %0.3fs" % (time() - t0))

# Train a SVM classification model

print("Fitting the classifier to the training set")
t0 = time()
```

```
param_grid = {'C': [1e3, 5e3, 1e4, 5e4, 1e5],
              'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1], }
clf = GridSearchCV(SVC(kernel='rbf', class_weight='balanced'), param_grid)
clf = clf.fit(X_train_pca, y_train)
print("done in %0.3fs" % (time() - t0))
print("Best estimator found by grid search:")
print(clf.best_estimator_)

# Quantitative evaluation of the model quality on the test set

print("Predicting people's names on the test set")
t0 = time()
y_pred = clf.predict(X_test_pca)
print("done in %0.3fs" % (time() - t0))

print(classification_report(y_test, y_pred, target_names=target_names))
print(confusion_matrix(y_test, y_pred, labels=range(n_classes)))

# Qualitative evaluation of the predictions using matplotlib

def plot_gallery(images, titles, h, w, n_row=3, n_col=4):
    """Helper function to plot a gallery of portraits"""
    plt.figure(figsize=(1.8 * n_col, 2.4 * n_row))
    plt.subplots_adjust(bottom=0, left=.01, right=.99, top=.90, hspace=.35)
    for i in range(n_row * n_col):
        plt.subplot(n_row, n_col, i + 1)
        plt.imshow(images[i].reshape((h, w)), cmap=plt.cm.gray)
        plt.title(titles[i], size=12)
        plt.xticks(())
        plt.yticks(())

# plot the result of the prediction on a portion of the test set

def title(y_pred, y_test, target_names, i):
    pred_name = target_names[y_pred[i]].rsplit(' ', 1)[-1]
    true_name = target_names[y_test[i]].rsplit(' ', 1)[-1]
    return 'predicted: %s\ntrue:      %s' % (pred_name, true_name)
```

```
prediction_titles = [title(y_pred, y_test, target_names, i)
                     for i in range(y_pred.shape[0])]
plot_gallery(X_test, prediction_titles, h, w)

# plot the gallery of the most significative eigenfaces

eigenface_titles = ["eigenface %d" % i for i in range(eigenfaces.shape[0])]
plot_gallery(eigenfaces, eigenface_titles, h, w)
plt.show()
```