Real Time Face Detection and Recognition Using Haar Cascade and Support Vector Machines

Chenxing Ouyang
University of California, San Diego
COGS 109 Final Project
c2ouyang@ucsd.edu

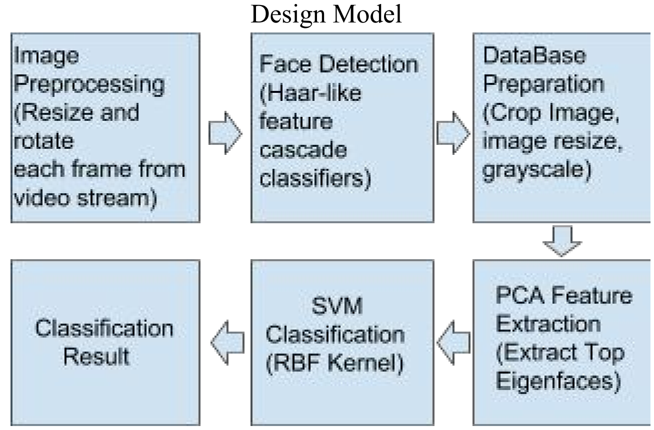
***Abstract*—Face detection and recognition have both been active research areas over the past few decades and have been proven effective in many applications such as computer security and artificial intelligence. This paper introduces a practical system for tracking and recognizing faces in real time using a webcam. The first part of the system is facial detection, which is achieved using Haar feature-based cascade classifiers, a novel way proposed by Paul Viola and Michael Jones in their 2001 paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" [1]. To further improve the method, geometric transformations are applied to each frame for face detection, allowing detection up to 45 degrees of head tilting. The second part of the system, face recognition, is achieved through a hybrid model consisting of feature extraction and classification trained on the cropped Extended Yale Face Database B [2]. To build the model, 2452 samples from 38 people in the database are splitted into training and testing sets by a ratio of 3:1. The top 150 eigenfaces are extracted from 1839 training faces in the database using Principal Component Analysis (PCA). The principal components are then feeded into the C-SVM Classification model and trained with various kernel tricks. At the end of the recognition task, an accuracy of 93.3% is obtained with the Radial Basis Function (RBF) kernel on the testing set of 613 samples. Used in the real time application via webcam, the proposed system runs at 10 frames per second with high recognition accuracy relative to the number of training images of real time testers and how representative those training images are.**

I. INTRODUCTION

The social informational form of nonverbal communication such as facial expression have become a primary focus of attention within our society nowadays in many different areas including security system, law and enforcement, criminal identification, realistic authentication as well as credit card verification. Therefore, many researchers have proposed several different algorithms for solving these issues that are considered as vital to human liability, interpersonal relationship, and commercial applications. In reality, even though one could recognize thousand of different faces throughout their lifetime, the process of human recognition eventually becomes difficult in case of aging, size, illumination, rotation and time apart. Unlike other systems of identification, the significance of facial recognition algorithm does not require the collaboration of individual after the first detection and by taking into account these side factors, the facial recognition algorithm would able to produce a very accurate results compared to other verifications as well as human recognition skills.

The goal of this paper is to propose a practical system for tracking and recognizing faces in real time using a webcam. The design is shown as follows in Figure 1.

Fig. 1. Real time Face Detection and Recognition System Design Model



II. FACE DETECTION

In the face detection part of the system, Haar feature based cascade classifiers are used to detect faces in frames obtained from the webcam video stream. Haar cascade is a machine learning approach for visual detection because it is trained from a great amount of positive and negative images [4]. Once the cascade of classifiers are trained, they are capable of processing images effectively and rapidly

[3]. The face detection part of the proposed system is implemented in python using OpenCV. Haar feature based detection algorithm is used with the trained face cascade classifiers that came with OpenCV.

A. METHOD DESCRIPTION

Haar feature based classifiers are appearance based. When using this algorithm to detect a face, a combinations of features such as head shape, motion and skin color tones are compared and evaluated. This combination of features are detected and extracted through a cascade of weak classifiers, or stages that Haar cascade is consisted of. During detection, those classifiers are applied to a region of interest subsequently until at some stages, the region fails in a classifier and is rejected. Those classifiers at every stage are built out of basic decision tree classifiers using different Adaboosting techniques [4]. The fundamental concept for detecting objects is the utilization of Haar-like features since they are the basic inputs of those classifiers. Haar-like features exploits the contrasting values of adjacent grouping pixels, which are then used to detect lighting differences from the images. For example, Haar-like features can be selected rely the property that the eyes are darker than the nose bridge as shown in Figure 2. Those haar features usually consists of two or three of those grouping pixels and can be scaled to fit the region of interest being examined [3]. The cascade classifiers in OpenCV use the following Haar features in Figure 3.

Fig. 2. Use Haar Features To Finding Lighting Differences on Human Faces [4]

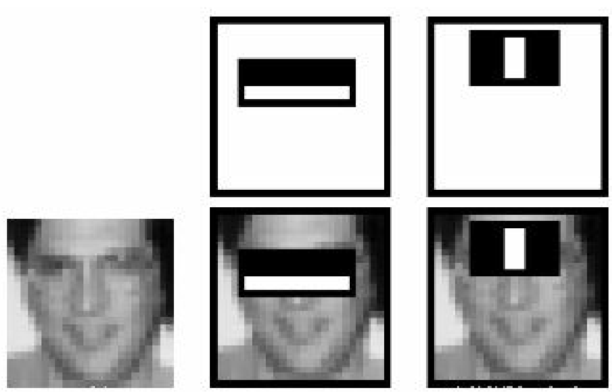
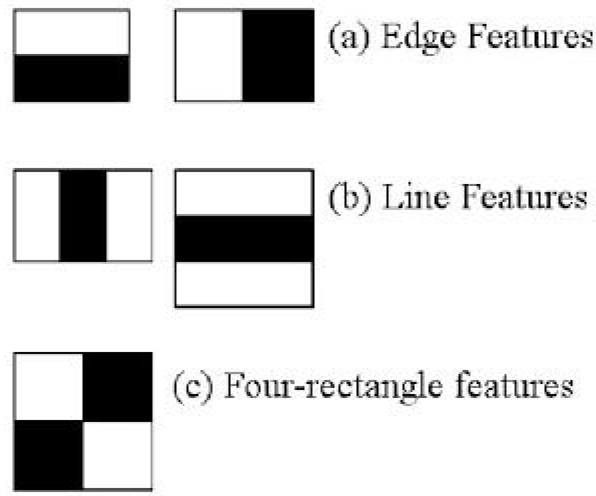


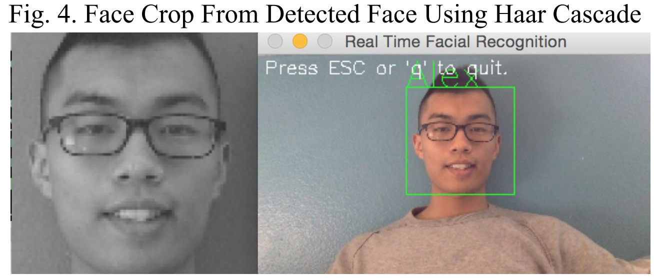
Fig. 3. Haar Features Used By OpenCV Cascade Classifiers [4]



However, there are tens of thousands of Haar-like features on human faces, how would one select the best features? Feature selection is achieved through a recent advancement called Adaboost, which is a machine learning technique that finds the best threshold amount all training images that will classify the face to positive and negative based on lighting differences. During which, the features with minimum error rate, or the features that best classifies the face and non face region are selected. According to the original paper by Viola and Jones, with only 200 selected features face detection can reach up to 95% accuracy.

To implement this method in real time, the detection algorithm is executed each frame for every frame in the video stream. In OpenCV, Haar cascade comes with parameters for optimizations, such as the minimum neighborhood distances between faces for multiple face detection and min face size to be detected. To further increase the frame rate and refreshing speed, the images are rescaled to be ¼ of its original pixel size and is converted to grayscale before feeding into the trained classifier for face detection. Once faces are found, they are then cropped out, processed and fed into face recognition part of the system as shown in Figure 4.

Fig. 4. Face Crop From Detected Face Using Haar Cascade



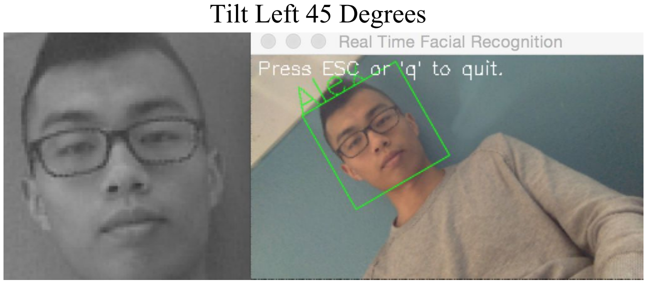
B. METHOD IMPROVEMENTS

Though Haar cascade seems to be efficient and fast, one of the biggest drawbacks of the original algorithm proposed by Viola and Jones is that it does not support rotated faces. The reason for such rotation invariance is that the algorithm uses Haar matrices which are rotation invariant in nature [3]. To overcome this limitation, several algorithms have been suggested, one of which is the Kanade-Lucas-tomasi algorithm (KLT), which finds the best match of the spatial rotation using the spatial intensity information from the image. Though KLT is computationally efficient, but for simplicity and time efficiency, I have implemented my own algorithm for detecting rotational faces.

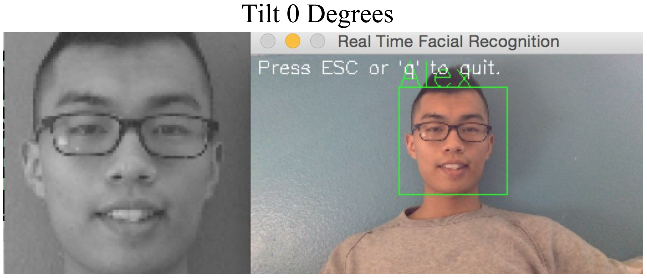
My method makes the assumption that people normally tilt their heads left and right more or less about -45 to 45 degrees. Since the given cascade classifiers can detect rotated faces up to approximately 15 degrees of rotation left and right, I used a dictionary mapping in python to keep track of three ndarray head rotation maps, namely: "left": np.array([-30, 0, 30]), "right": np.array([30, 0, -30]), and "middle": np.array([0, -30, 30])(See code in Appendix A). In each array arrangement that contains three degrees, a frame from the webcam stream is examined in real time at those three rotations through a geometric transformation, allowing detection up to 45 degrees of head tilting as shown in Figure 5.

Fig. 5. Face Rotation Variant of Face Recognition System

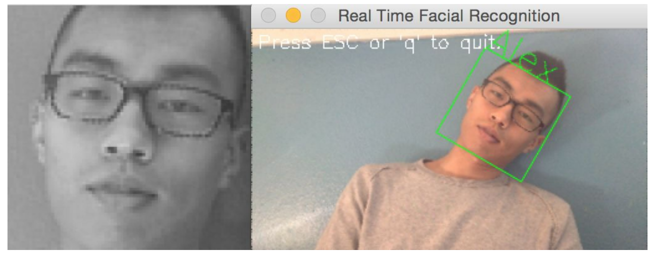
Tilt Left 45 Degrees



Tilt 0 Degrees



Tilt Right 45 Degrees

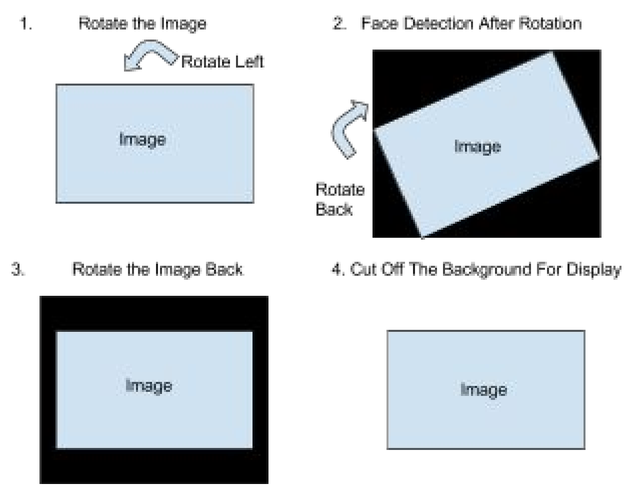


The geometric transformation of the frame is achieved through the procedure below using a rotational matrix:

$$\begin{bmatrix} \alpha & \beta & (1-\alpha)\cdot \texttt{center.x} - \beta \cdot \texttt{center.y} \\ -\beta & \alpha & \beta \cdot \texttt{center.x} + (1-\alpha)\cdot \texttt{center.y} \end{bmatrix}$$

[5] (See rotate_image() in Appendix B). After rotation, the face detection algorithm is used. The algorithm ends until either a face is found in one of the specified rotation angle in the rotation map or no face is found. Then the frame is rotated back with the offset cut off, then displayed in OpenCV as shown in Figure 6.

Fig. 6. Geometric Rotation of the Frames



The arrangement of three degrees are chosen after trying multiple of angle combinations. The choice of rotation degrees is shown to be most effective. Since people normally tilt their heads either from left to middle to right, or from right to middle to left for about 45 degrees. By having the same arrangement in different orders decreases detection iterations, thus increasing computationally efficiency. However, the overall algorithm for face detection still proves to be computationally inefficient and somewhat inaccuracy even after optimization. First of all it is still rather inaccurate after optimization because due to lighting effect, Haar features are not detected consistently after head tilting though my algorithm have provided higher detection rate than the original algorithm. Second of all, the algorithm is slow. Since the same face detection procedure is iterated multiple times until a face is found. For frames that have no face detected, detection algorithms would iterate through all degrees in the rotation maps, then the frame is rendered for display. To avoid this pitfall, I have decrease the detection rate to be once every two frames instead of once per frame if no faces are found in frames consecutively. Moreover, multiple trained face classifiers such as frontal face and side face classifiers are used to obtain more accurate detection result in addition to face rotation detection.

IV. FACIAL RECOGNITION

The facial recognition part of the system is a hybrid model mainly consisted of three modules: data preparation, feature extraction and classification.

A. DATABASE PREPARATION

The database used to train the C-SVM Classification model is prepared using the cropped faces from the extended Yale Face Database B, which contains 2452 images from 37 human subjects under 9 poses and 64 illumination conditions [2]. The initial conditions for all training and testing image are that all images must be taken in similar illuminations and without body obstruction. Images must also be in grayscale and contains only facial features with height ranges from the hairline above forehead to the chain. The originally cropped images are 192 by 168 pixels, and they are resized to be 50 by 50 pixels to achieve higher processing speed during training and detection.

To test the integrated system in real time, the images of user's face must be scanned, cropped out, resized and grayscaled until they satisfy the initial conditions sued for database preparation discussed above, and then they can be stored in the database for feature extraction and SVM training. I have build a automatic training system in OpenCV that allows users to take pictures of their faces (The code is in Appendix C). The training system would crop out the user's faces following the initial conditions for data preparation using the face detection system discussed above and specify a customized face profile directory in a default database to save all the training images in real time.

B.FEATURE EXTRACTION

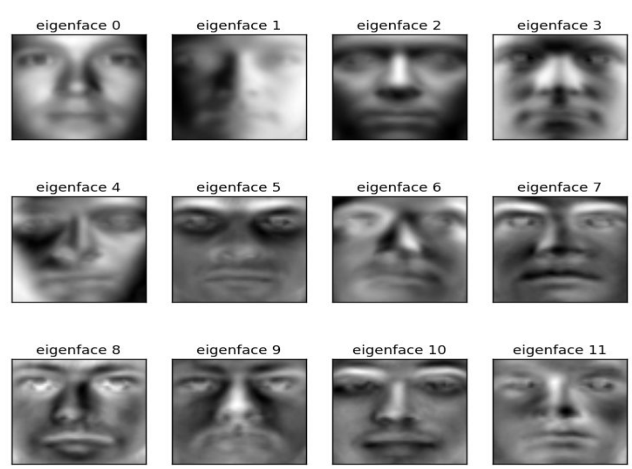
To build the model from the initially prepared database from the Extended Yale Face Database, 2452 samples from 38 people in the database are splitted into training and testing sets by a ratio of 3:1 randomly. The next step is to extract the most representative features from the training set of biometric images in the form of numeric values that can be compared and evaluated. The top 150 eigenfaces are extracted from 1839 training faces in the database using Principal Component Analysis (PCA). PCA is a unsupervised dimensionality reduction method that uses an orthogonal transformation to extract the linearly independent variables from linearly dependent variables. Those linearly independent variables are called the principal components, the top principal components contains the largest variance, which means they account for the largest variability of the data. PCA allows for geometric features extraction on lips and eyes in biometric images and those feature vectors are called eigenfaces. Another advantage of feature extraction before training is to reduce the dimensionality of the data for best computational efficiency optimization.

Here we provide a detailed step by step explanation on how to use PCA to extract Eigenfaces feature vectors:

1. Zero mean the data, compute mean and subtract from all the data: z (i) = x (i) − m
2. Compute the covariance matrix, normalize it and its eigenvectors and their associated eigenvalues.: [Vp,Dp] = eig(1/N*Z*Z')
3. Sort the eigenvectors in order of decreasing eigenvalue: [V, D] = eigsort(Vp,Dp);
4. Compute principal components: C = V'*Z;
5. Project original data onto reduced component 'principal component' space, take the top k eigenvectors: C_hat = C(1:k,:); where k≤p, p = original data dimension. The top k eigen vectors are the eigenfaces we are looking for. [6]

A visual representation of the extracted feature vectors from the prepared database is shown in the plot gallery of the 12 most significative eigenfaces in figure 7.

Fig. 7. Top 12 Eigenfaces



C. CLASSIFICATION

C-SVM is used for classification part of the model. C-SVM, or type 1 SVM is a statistical model that separates data sets by having maximum distances between data classes. Through the search of an optimal hyperplane, data classes are distinguished and separated. The bounds between the data classes and OSH are the so called support vectors [2]. The training involves the minimization of the error function of the C-SVM classification model. The error function:

$$\frac{1}{2}w^T w + C\sum_{i=1}^{N}\xi_i$$

is subjected to the constraint function:

$$y_i\left(w^T\phi(x_i)+b\right)\geq 1-\xi_i \text{ and } \xi_i \geq 0, i=1,\ldots,N$$

In the error function, C is the capacity constant and w is the vector of coefficients. In the constraint function, b is a constant and ξ_i represents parameters for handling inputs. N training classes are labeled by index i. $y \in +-1$ represents the class labels and the kernel ϕ is used to transform data from the input to the feature space.

The code for C-SVM classification model is implemented in scikit learn as shown in Appendix D. Using scikit learn, the principal components are then feeded into the C-SVM Classification model and trained with various kernel tricks, such as linear function, polynomial function, sigmoid function and radial basis function kernels.

V. EXPERIMENTATION RESULT

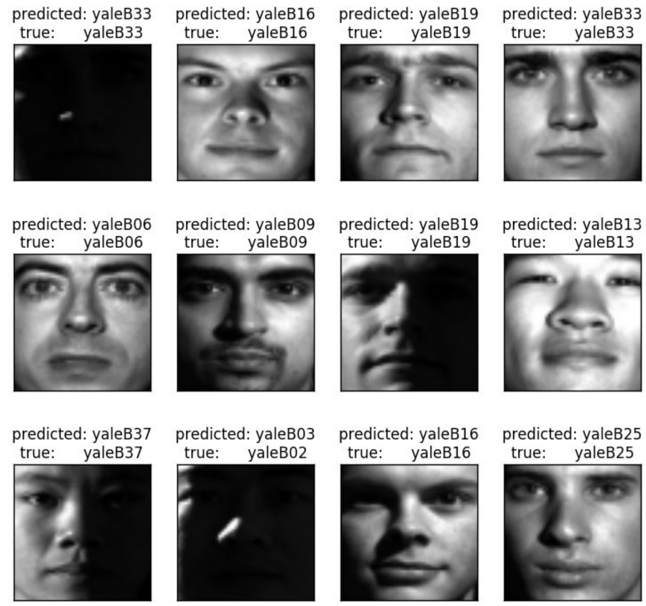
In the face recognition model, we have prepared the data and extracted the top 150 eigenfaces. The principal components are then feeded into the C-SVM Classification model and trained with various kernel tricks, the results are shown below in TABLE 1.

TABLA I
Comparisons Of Different Kernel Tricks For Facial Recognition

SVM Kernel Function	Recognition Rate
Linear	*90.9%*
Poly	*79.9%*
Sigmoid	*1.5%*
RBF	*93.8%*

At the end of the recognition task, an accuracy of 93.3% is obtained with the Radial Basis Function (RBF) kernel on the testing set of 613 samples. To show the quantitative evaluation of the model quality, result of the prediction on a portion of the test set are plotted as shown below in Figure 8.

Fig. 8. Sample Predictions From Extended Yale Face Database B



This system has been proven effective if all initial data preparations are satisfied for the training data. Used in the real time application via webcam, the proposed system runs at 10 frames per second with high recognition accuracy relative to the number of training images of real time testers and the saliency of the training images.

REFERENCIAS

[1] "Eigenfaces and Support Vector Machine Approaches for Hybrid Face Recognition." The Online Journal on Electronics and Electrical Engineering (OJEEE). December 1, 2010. Accessed December 4, 2015. http://www.infomesr.org/attachments/09-005.pdf.

[2] "The Extended Yale Face Database B." May 1, 2005. Accessed December 11, 2015. http://vision.ucsd.edu/~leekc/ExtYaleDatabase/ExtYaleB.html

[4]"Face Detection Using Haar Cascades¶." Face Detection Using Haar Cascades — OpenCV-Python Tutorials 1 Documentation. Accessed December 11, 2015. http://opencv-python tutroals.readthedocs.org/en/latest/py_tutorials/py_objdetect/py_face _detection/py_face_detection.html.

[5] "Geometric Image Transformations¶." Geometric Image Transformations — OpenCV 2.4.12.0 Documentation. 2011. Accessed December 11, 2015.

http://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transfor mations.html.

[6]"Dimensionality Reduction and PCA." COGS 109, Fall 2015 Modeling and Data Analysis. Accessed October 11, 2015. https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbnx1Y3NkY29nczEwOWZhbGwyMDE1fGd4OjhlNzY5OGNlOTMwYjEzOA

7] "Support Vector Machines (SVM)." Support Vector Machines (SVM). 2015. Accessed December 11, 2015. http://www.statsoft.com/textbook/support-vector-machines.

[8] "Sklearn.svm.SVC¶." Sklearn.svm.SVC — Scikit-learn 0.17 Documentation. Accessed December 11, 2015. http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC.

Appendix A (main.py)

```
"""
=========================================================
   Faces recognition and detection using OpenCV
=========================================================

The dataset used is the Extended Yale Database B Cropped

  http://vision.ucsd.edu/~leekc/ExtYaleDatabase/ExtYaleB.html


Summary:
        Real time facial tracking and recognition using harrcascade
        and SVM

To Ignore Warnings:
        python -W ignore main.py

        Created by:  Chenxing Ouyang

"""

import cv2
import os
import numpy as np
from scipy import ndimage
from time import time
import logging
import matplotlib.pyplot as plt


import utils as ut
import svm

import warnings


print(__doc__)

###############################################################################
# Building SVC from database

# Used to load data bases
def load_Yale_Exteded_Database(number_of_Faces):
    for i in range (1, number_of_Faces):
        missing_database = [14]
        name_prefix = "yaleB"
        if i < 10:
            name_index = "0" + str(i)
        else:
            name_index = str(i)
```

```
        name = name_prefix + name_index
        if i in missing_database:
            print "Missing Database: ", name
        else:
            target_names.append(name)


FACE_DIM = (50,50) # h = 50, w = 50

# target_names = ["Alex02", "Kristine"]
target_names = []

# load YaleDatabaseB
load_Yale_Exteded_Database(40)

# print target_names

# Build the classifier
face_data, face_target = ut.load_data(target_names, data_directory = "../face_data/")

print face_target.shape[0], " samples from ", len(target_names), " people are loaded"
for i in range(1,2): print ("\n")

# clf = svm.build_SVC(face_data, face_target, FACE_DIM)
clf, pca = svm.test_SVM(face_data, face_target, FACE_DIM, target_names)


################################################################################
# Facial Recognition and Tracking Live


DISPLAY_FACE_DIM = (200, 200)
SKIP_FRAME = 2      # the fixed skip frame
frame_skip_rate = 0 # skip SKIP_FRAME frames every other frame
SCALE_FACTOR = 2 # used to resize the captured frame for face detection for faster processing speed
face_cascade = cv2.CascadeClassifier("../data/haarcascade_frontalface_default.xml") #create a cascade classifier
sideFace_cascade = cv2.CascadeClassifier('../data/haarcascade_profileface.xml')

# dictionary mapping used to keep track of head rotation maps
rotation_maps = {
    "left": np.array([-30, 0, 30]),
    "right": np.array([30, 0, -30]),
    "middle": np.array([0, -30, 30]),
}

def get_rotation_map(rotation):
    """ Takes in an angle rotation, and returns an optimized rotation map """
    if rotation > 0: return rotation_maps.get("right", None)
    if rotation < 0: return rotation_maps.get("left", None)
    if rotation == 0: return rotation_maps.get("middle", None)

current_rotation_map = get_rotation_map(0)


webcam = cv2.VideoCapture(0)
```

```
ret, frame = webcam.read() # get first frame
frame_scale = (frame.shape[1]/SCALE_FACTOR,frame.shape[0]/SCALE_FACTOR)  # (y, x)

crop_face = []
num_of_face_saved = 0


while ret:
    key = cv2.waitKey(1)
    # exit on 'q' 'esc' 'Q'
    if key in [27, ord('Q'), ord('q')]:
        break
    # resize the captured frame for face detection to increase processing speed
    resized_frame = cv2.resize(frame, frame_scale)

    processed_frame = resized_frame
    # Skip a frame if the no face was found last frame

    t0 = time()

    if frame_skip_rate == 0:
        faceFound = False
        for rotation in current_rotation_map:

            rotated_frame = ndimage.rotate(resized_frame, rotation)

            gray = cv2.cvtColor(rotated_frame, cv2.COLOR_BGR2GRAY)

            # return tuple is empty, ndarray if detected face
            faces = face_cascade.detectMultiScale(
                gray,
                scaleFactor=1.3,
                minNeighbors=5,
                minSize=(30, 30),
                flags=cv2.cv.CV_HAAR_SCALE_IMAGE
            )

            # If frontal face detector failed, use profileface detector
            faces = faces if len(faces) else sideFace_cascade.detectMultiScale(
                gray,
                scaleFactor=1.3,
                minNeighbors=5,
                minSize=(30, 30),
                flags=cv2.cv.CV_HAAR_SCALE_IMAGE
            )

            # for f in faces:
            #     x, y, w, h = [ v*SCALE_FACTOR for v in f ] # scale the bounding box back to original frame size
            #     cv2.rectangle(frame, (x,y), (x+w,y+h), (0,255,0))
            #     cv2.putText(frame, "DumbAss", (x,y), cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0,255,0))

            if len(faces):
                for f in faces:
                    # Crop out the face
                    x, y, w, h = [ v for v in f ] # scale the bounding box back to original frame size
                    crop_face = rotated_frame[y: y + h, x: x + w]   # img[y: y + h, x: x + w]
```

```
                crop_face = cv2.resize(crop_face, DISPLAY_FACE_DIM, interpolation = cv2.INTER_AREA)

                # Name Prediction
                face_to_predict = cv2.resize(crop_face, FACE_DIM, interpolation = cv2.INTER_AREA)
                face_to_predict = cv2.cvtColor(face_to_predict, cv2.COLOR_BGR2GRAY)
                face_to_predict = face_to_predict.ravel()
                name_to_display = svm.predict(clf, pca, face_to_predict, target_names)

                # Display frame
                cv2.rectangle(rotated_frame, (x,y), (x+w,y+h), (0,255,0))
                cv2.putText(rotated_frame, name_to_display, (x,y), cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0,255,0))

            # rotate the frame back and trim the black paddings
            processed_frame = ut.trim(ut.rotate_image(rotated_frame, rotation * (-1)), frame_scale)

            # reset the optmized rotation map
            current_rotation_map = get_rotation_map(rotation)

            faceFound = True


            break

        if faceFound:
            frame_skip_rate = 0
            # print "Face Found"
        else:
            frame_skip_rate = SKIP_FRAME
            # print "Face Not Found"

    else:
        frame_skip_rate -= 1
        # print "Face Not Found"


    # print("\nDetection + Classification took %0.3fs" % (time() - t0))
    # print "Frame dimension: ", processed_frame.shape

    cv2.putText(processed_frame, "Press ESC or 'q' to quit.", (5, 15),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255,255,255))

    cv2.imshow("Real Time Facial Recognition", processed_frame)



    if len(crop_face):
        cv2.imshow("Cropped Face", cv2.cvtColor(crop_face, cv2.COLOR_BGR2GRAY))
        # face_to_predict = cv2.resize(crop_face, FACE_DIM, interpolation = cv2.INTER_AREA)
        # face_to_predict = cv2.cvtColor(face_to_predict, cv2.COLOR_BGR2GRAY)
        # name_to_display = svm.predict(clf, pca, face_to_predict, target_names)
    # get next frame
    ret, frame = webcam.read()


webcam.release()
cv2.destroyAllWindows()
```

Appendix B (utils.py)

```
"""
Auther: Chenxing Ouyang <c2ouyang@ucsd.edu>

This file is part of Cogs 109 Project.

Summary: Utilties used for facial tracking in OpenCV and facial recognition in SVM

"""


import cv2
import numpy as np
from scipy import ndimage
import os
import errno


###############################################################################
# Used For Facial Tracking and Traning in OpenCV

def rotate_image(image, angle, scale = 1.0):
    """ returns an rotated image with the same dimensions """
    if angle == 0: return image
    h, w = image.shape[:2]
    rot_mat = cv2.getRotationMatrix2D((w/2, h/2), angle, scale)
    return cv2.warpAffine(image, rot_mat, (w, h), flags=cv2.INTER_LINEAR)

def trim(img, dim):
    """ dim = (y, x),  img.shape = (x, y) retruns a trimmed image with black paddings removed"""
    # if the img has the same dimension then do nothing
    if img.shape[0] == dim[1] and img.shape[1] == dim[0]: return img
    x = int((img.shape[0] - dim[1])/2) + 1
    y = int((img.shape[1] - dim[0])/2) + 1
    trimmed_img = img[x: x + dim[1], y: y + dim[0]]   # crop the image
    return trimmed_img

def clean_directory(directory = "../pics"):
    """ Deletes all files and folders contained in the directory """
    for the_file in os.listdir(directory):
        file_path = os.path.join(directory, the_file)
        try:
            if os.path.isfile(file_path):
                os.unlink(file_path)
            #elif os.path.isdir(file_path): shutil.rmtree(file_path)
        except Exception, e:
            print e


def create_directory(path):
    """ create directories for saving images"""
    try:
        print "Making directory"
```

```python
        os.makedirs(path)
    except OSError as exception:
        if exception.errno != errno.EEXIST:
            raise

def create_profile_in_database(profile_folder_name, database_path="../face_data/", clean_directory=False):
    """ Save to the default directory """
    profile_folder_path = database_path + profile_folder_name + "/"
    create_directory(profile_folder_path)
    # Delete all the pictures before recording new
    if clean_directory:
        clean_directory(profile_folder_path)
    return profile_folder_path




###############################################################################
# Used for Facial Recognition in SVM

def readImage(directory, y, dim = (50, 50)):
    """ Takes in a directory of images
      Returns X_data = (numOfFace X ImgPixelSize) face data array
              Y_data = (numOfFace X 1) target_name_index array
    """
    X_data = np.array([])
    index = 0
    for the_file in os.listdir(directory):
        file_path = os.path.join(directory, the_file)
        if file_path.endswith(".png") or file_path.endswith(".jpg") or file_path.endswith(".pgm"):
            img = cv2.imread(file_path, 0)
            img = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)
            img_data = img.ravel()
            X_data = img_data if not X_data.shape[0] else np.vstack((X_data,img_data))
            index += 1

    Y_data = np.empty(index, dtype = int)
    Y_data.fill(y)
    return X_data, Y_data


def errorRate(pred, actual):
    """ Returns the error rate """
    if pred.shape != actual.shape: return None
    error_rate = np.count_nonzero(pred - actual)/float(pred.shape[0])
    return error_rate

def recognitionRate(pred, actual):
    """ Returns the recognition rate and error rate """
    if pred.shape != actual.shape: return None
    error_rate = np.count_nonzero(pred - actual)/float(pred.shape[0])
    recognitionRate = 1.0 - error_rate
    return recognitionRate, error_rate


def load_data(target_names, data_directory):
```

```
""" Takes in a list of target_names (names of the directory that contains face pics)
  Retruns X_mat = (numbeOfFace X numberOfPixel) face data matrix
          Y_mat = (numbeOfFace X 1) target_name_index matrix
"""
if len(target_names) < 2: return None
first_data = str(target_names[0])
first_data_path = os.path.join(data_directory, first_data)
X1, y1 = readImage(first_data_path, 0)
X_mat = X1
Y_mat = y1
print "Loading Database: "
print 0,"    ", first_data_path
for i in range(1, len(target_names)):
    directory_name = str(target_names[i])
    directory_path = os.path.join(data_directory, directory_name)
    tempX, tempY = readImage(directory_path, i)
    X_mat = np.concatenate((X_mat, tempX), axis=0)
    Y_mat = np.append(Y_mat, tempY)
    print i, "    ", directory_path
return X_mat, Y_mat
```

Appendix C (train.py)

```
"""
Auther: Chenxing Ouyang <c2ouyang@ucsd.edu>

This file is part of Cogs 109 Project.

Summary: Used for data colelction and SVM training

"""

import cv2
import numpy as np
from scipy import ndimage
import sys
import os

import utils as ut


FACE_DIM = (200, 200)
SKIP_FRAME = 2      # the fixed skip frame
frame_skip_rate = 0 # skip SKIP_FRAME frames every other frame
SCALE_FACTOR = 4 # used to resize the captured frame for face detection for faster processing speed
face_cascade = cv2.CascadeClassifier("../data/haarcascade_frontalface_default.xml") #create a cascade classifier
sideFace_cascade = cv2.CascadeClassifier('../data/haarcascade_profileface.xml')

# dictionary mapping used to keep track of head rotation maps
rotation_maps = {
    "left": np.array([-30, 0, 30]),
    "right": np.array([30, 0, -30]),
    "middle": np.array([0, -30, 30]),
}

def get_rotation_map(rotation):
    """ Takes in an angle rotation, and returns an optimized rotation map """
    if rotation > 0: return rotation_maps.get("right", None)
    if rotation < 0: return rotation_maps.get("left", None)
    if rotation == 0: return rotation_maps.get("middle", None)

current_rotation_map = get_rotation_map(0)


webcam = cv2.VideoCapture(0)

ret, frame = webcam.read() # get first frame
frame_scale = (frame.shape[1]/SCALE_FACTOR,frame.shape[0]/SCALE_FACTOR)  # (y, x)


crop_face = []
num_of_face_to_collect = 150
num_of_face_saved = 0


#  For saving face data to directory
```

```
profile_folder_path = None

if len(sys.argv) == 1:
    print "\nError: No Saving Diectory Specified\n"
    exit()
elif len(sys.argv) > 2:
    print "\nError: More Than One Saving Directory Specified\n"
    exit()
else:
    profile_folder_path = ut.create_profile_in_database(sys.argv[1])


while ret:
    key = cv2.waitKey(1)
    # exit on 'q' 'esc' 'Q'
    if key in [27, ord('Q'), ord('q')]:
        break
    # resize the captured frame for face detection to increase processing speed
    resized_frame = cv2.resize(frame, frame_scale)

    processed_frame = resized_frame
    # Skip a frame if the no face was found last frame
    if frame_skip_rate == 0:
        faceFound = False
        for rotation in current_rotation_map:

            rotated_frame = ndimage.rotate(resized_frame, rotation)

            gray = cv2.cvtColor(rotated_frame, cv2.COLOR_BGR2GRAY)

            # return tuple is empty, ndarray if detected face
            faces = face_cascade.detectMultiScale(
                gray,
                scaleFactor=1.3,
                minNeighbors=5,
                minSize=(30, 30),
                flags=cv2.cv.CV_HAAR_SCALE_IMAGE
            )

            # If frontal face detector failed, use profileface detector
            faces = faces if len(faces) else sideFace_cascade.detectMultiScale(
                gray,
                scaleFactor=1.3,
                minNeighbors=5,
                minSize=(30, 30),
                flags=cv2.cv.CV_HAAR_SCALE_IMAGE
            )

            # for f in faces:
            #     x, y, w, h = [ v*SCALE_FACTOR for v in f ] # scale the bounding box back to original frame size
            #     cv2.rectangle(frame, (x,y), (x+w,y+h), (0,255,0))
            #     cv2.putText(frame, "DumbAss", (x,y), cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0,255,0))

            if len(faces):
                for f in faces:
                    x, y, w, h = [ v for v in f ] # scale the bounding box back to original frame size
```

```
                crop_face = rotated_frame[y: y + h, x: x + w]   # img[y: y + h, x: x + w]
                crop_face = cv2.resize(crop_face, FACE_DIM, interpolation = cv2.INTER_AREA)
                cv2.rectangle(rotated_frame, (x,y), (x+w,y+h), (0,255,0))
                cv2.putText(rotated_frame, "DumbAss", (x,y), cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0,255,0))

            # rotate the frame back and trim the black paddings
            processed_frame = ut.trim(ut.rotate_image(rotated_frame, rotation * (-1)), frame_scale)

            # reset the optmized rotation map
            current_rotation_map = get_rotation_map(rotation)

            faceFound = True


            break

        if faceFound:
            frame_skip_rate = 0
            # print "Face Found"
        else:
            frame_skip_rate = SKIP_FRAME
            # print "Face Not Found"

    else:
        frame_skip_rate -= 1
        # print "Face Not Found"



    cv2.putText(processed_frame, "Press ESC or 'q' to quit.", (5, 15),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255,255,255))

    cv2.imshow("Real Time Facial Recognition", processed_frame)



    if len(crop_face):
        cv2.imshow("Cropped Face", cv2.cvtColor(crop_face, cv2.COLOR_BGR2GRAY))
        if num_of_face_saved < num_of_face_to_collect and key == ord('p'):
            face_to_save = cv2.resize(crop_face, (50, 50), interpolation = cv2.INTER_AREA)
            face_name = profile_folder_path+str(num_of_face_saved)+".png"
            cv2.imwrite(face_name, face_to_save)
            print "Pic Saved: ", face_name
            num_of_face_saved += 1

    # get next frame
    ret, frame = webcam.read()


webcam.release()
cv2.destroyAllWindows()
```

Appendix D (svm.py)

```
"""
Auther: Chenxing Ouyang <c2ouyang@ucsd.edu>

This file is part of Cogs 109 Project.

"""

import cv2
import os
import numpy as np
from scipy import ndimage
from time import time
import warnings

with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    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

import utils as ut


def test_SVM(face_data, face_target, face_dim, target_names):
    """ Testing SVM

        Build SVM classification modle using the face_data matrix (numOfFace X numOfPixel)
        and face_target array, face_dim is a tuple of the dimension of each image(h,w)
        Returns the SVM classification modle
    """
    X = face_data
    y = face_target

    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 # maximum number of components to keep

    print("\nExtracting the top %d eigenfaces from %d faces" % (n_components, X_train.shape[0]))

    pca = RandomizedPCA(n_components=n_components, whiten=True).fit(X_train)
    eigenfaces = pca.components_.reshape((n_components, face_dim[0], face_dim[1]))

    # This portion of the code is used if the data is scarce, it uses the number
    # of imputs as the number of features
    # pca = RandomizedPCA(n_components=None, whiten=True).fit(X_train)
```

```
    # eigenfaces = pca.components_.reshape((pca.components_.shape[0], face_dim[0], face_dim[1]))

    print("\nProjecting the input data on the eigenfaces orthonormal basis")
    t0 = time()
    X_train_pca = pca.transform(X_train)
    X_test_pca = pca.transform(X_test)

    # Train a SVM classification model

    print("\nFitting the classifier to the training set")
    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)
    # Train_pca Test Error Rate:  0.0670016750419
    # Train_pca Test Recognition Rate:  0.932998324958



    # clf = SVC(kernel='linear', C=1)
    # 2452  samples from  38  people are loaded
    # Extracting the top 150 eigenfaces from 1839 faces
    # Extracting the top 150 eigenfaces from 1790 faces
    # Train_pca Test Error Rate:  0.0904522613065
    # Train_pca Test Recognition Rate:  0.909547738693

    # clf = SVC(kernel='poly')
    # Train_pca Test Error Rate:  0.201005025126
    # Train_pca Test Recognition Rate:  0.798994974874

    # clf = SVC(kernel='sigmoid')
    # Train_pca Test Error Rate:  0.985318107667
    # Train_pca Test Recognition Rate:  0.0146818923328


    # clf = SVC(kernel='rbf').fit(X_train, y_train)
    # Train_pca Test Error Rate:  0.0619765494137
    # Train_pca Test Recognition Rate:  0.938023450586



    # Best Estimator found using Radial Basis Function Kernal:
    clf = SVC(C=1000.0, cache_size=200, class_weight='balanced', coef0=0.0,
  decision_function_shape=None, degree=3, gamma=0.0001, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
    # Train_pca with Alex Test Error Rate:  0.088424437299
    # Train_pca with Alex Test Recognition Rate:  0.911575562701

    clf = clf.fit(X_train_pca, y_train)
    # print("\nBest estimator found by grid search:")
    # print(clf.best_estimator_)

    ###############################################################################
    # Quantitative evaluation of the model quality on the test set
    print("\nPredicting people's names on the test set")
    y_pred = clf.predict(X_test_pca)
```

```
# print "predicated names: ", y_pred
# print "actual names: ", y_test
print "Test Error Rate: ", ut.errorRate(y_pred, y_test)
print "Test Recognition Rate: ", 1.0-ut.errorRate(y_pred, y_test)

###############################################################################
# Testing

X_test_pic1 = X_test[0]
X_test_pic1_for_display = np.reshape(X_test_pic1, face_dim)

t0 = time()
pic1_pred_name = predict(clf, pca, X_test_pic1, target_names)
print("\nPrediction took %0.3fs" % (time() - t0))
print "\nPredicated result for picture_1 name: ", pic1_pred_name
for i in range(1,3): print ("\n")

# Display the picture
# plt.figure(1)
# plt.title(pic1_pred_name)
# plt.subplot(111)
# plt.imshow(X_test_pic1_for_display)
# plt.show()


###############################################################################
# Qualitative evaluation of the predictions using matplotlib
# import matplotlib.pyplot as plt

# def plot_gallery(images, titles, face_dim, 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(face_dim), 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, face_dim)

# # 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, face_dim)

    # plt.show()


    return clf, pca


def build_SVC(face_data, face_target, face_dim):
    """ Build SVM classification modle using the face_data matrix (numOfFace X numOfPixel)
        and face_target array, face_dim is a tuple of the dimension of each image(h,w)
        Returns the SVM classification modle
    """
    X = face_data
    y = face_target

    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 # maximum number of components to keep

    print("\nExtracting the top %d eigenfaces from %d faces" % (n_components, X_train.shape[0]))

    pca = RandomizedPCA(n_components=n_components, whiten=True).fit(X_train)
    eigenfaces = pca.components_.reshape((n_components, face_dim[0], face_dim[1]))

    # This portion of the code is used if the data is scarce, it uses the number
    # of imputs as the number of features
    # pca = RandomizedPCA(n_components=None, whiten=True).fit(X_train)
    # eigenfaces = pca.components_.reshape((pca.components_.shape[0], face_dim[0], face_dim[1]))

    print("\nProjecting the input data on the eigenfaces orthonormal basis")
    t0 = time()
    X_train_pca = pca.transform(X_train)
    X_test_pca = pca.transform(X_test)

    # Train a SVM classification model

    print("\nFitting the classifier to the training set")
    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)
    # Best Estimator found:
    clf = SVC(C=1000.0, cache_size=200, class_weight='balanced', coef0=0.0,
    decision_function_shape=None, degree=3, gamma=0.0001, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False).fit(X_train_pca, y_train)


    # Quantitative evaluation of the model quality on the test set
    print("\nPredicting people's names on the test set")
    y_pred = clf.predict(X_test_pca)
    # print "predicated names: ", y_pred
    # print "actual names: ", y_test
    print "Test Error Rate: ", ut.errorRate(y_pred, y_test)
```

```
    return clf, pca


def predict(clf, pca, img, target_names):
    """ Takes in a classifier, img (1 X w*h) and target_names
        Returns the predicated name
    """
    principle_component = pca.transform(img)
    pred = clf.predict(principle_component)
    name = target_names[pred]
    return name
```