# CSC 420 Project Report

**Chen Wang and Jiaguan Tang**

**Utorid: wangc425 & tangjiag**

**Student #: 1006058926 & 1006488793**

# Table of Contents

# 1. Introduction:

In this project, we are going to analyze movie trailers. Video analysis in computer vision focuses on object detection, which involves identifying and locating objects within a video frame. It has applications in surveillance, autonomous driving, and robotics. Various techniques, including deep learning-based methods, are used to improve the accuracy and effectiveness of object detection algorithms. In this scenario, we will be using shot detection and face detection techniques to analyze three different movie trailers.

The main logic of this project is divided into two parts. The first part is to go through the video and then get video clips. The second part is to track actor faces from the video shots and box the faces in the corresponding video clip.

# 2. Shot Detection

A  shot refers to a series of consecutive video frames that are captured with smooth and continuous camera movement, and together form an uninterrupted sequence of visual content in terms of space, time, and graphical arrangement. The point at which one shot ends and another begins is known as a shot boundary. Shot detection is to detect shot boundaries within a video. It can help face detection by segmenting the video into different shots, allowing for the efficient application of face detection algorithms optimized for specific shot types and lighting conditions. [1] [2]

## 2.1 Method

There are several common shot detection methods.
Threshold-based methods: These methods analyze changes in visual content such as camera angle, lighting, and motion, and set a threshold for the change required to signal a new shot.

Histogram-based methods: These methods analyze the distribution of pixel values within the video frames and use changes in the distribution to identify shots.

Edge-based methods: These methods analyze the edges and contours of objects within the video frames and use changes in the edges to detect shots. [5]

## 2.2 Implementation

I choose Threshold-based methods. Choose a feature to use for shot detection. An example of a feature is frame differences.

### 2.2.1 absolute difference

One such method involves using the absolute difference between two consecutive frames as a feature to determine the shots by comparing them with a threshold value. However, determining a suitable threshold

value for this feature can be challenging. The difference between frames and different videos can vary significantly, which means that an independent threshold value must be set for each video. For instance, if video A is 1080p, and video B is 480p, the absolute difference in video A is much higher than that in video B. Despite using independent threshold values, the output can still be inaccurate since the changing pattern in one video can vary substantially. As such, it is essential to explore other features or methods that can provide more accurate shot detection.

### 2.2.2 average absolute difference

Another way is to use the average absolute difference between two consecutive frames as a feature. This approach is advantageous because it is independent of the video's resolution and can be used with various types of videos. However, this method is still prone to inaccuracies since the average difference can be high even when there is no shot boundary. For instance, in consecutive frames, the average absolute difference can be all high, like 16, or it can be very low like 2, which can result in false positives or negatives in shot detection. To overcome this limitation, we need to explore other features.

### 2.2.3 difference of average absolute difference

This feature calculates the difference of the average absolute difference between frames and uses the changing difference of frames to detect shot boundaries, aiming to avoid any biases from previous methods. A shot is defined as a continuous sequence of video frames captured by a single camera without any cuts or edits. Therefore, in a shot, the differences between all consecutive frames tend to be similar. For example, if the consecutive difference of four frames is 16, 15, and 16, then no shot boundary is detected. However, if the consecutive difference of four frames is 16, 70, and 70, a shot boundary is detected, and the previous shot ends in frame one, and the next shot starts in frame two. This feature has demonstrated a higher accuracy and outperforms previous methods.

## 2.3 evaluation

We will be using Recall, Precision, and F1 Score to determine the metrics of our shot detection. To calculate R, P, and F1 for shot detection, we need to define what constitutes a "positive" and "negative" case. In the context of shot detection, we can consider a positive case as a frame is a shot boundary and a negative case as a frame or time interval is not a shot boundary.

Manually compute a ground truth dataset that contains the actual shot boundaries, and a predicted dataset that contains the shot boundaries detected by our model

True Positives(Corrects): Frames that are correctly identified as containing a shot boundary.
False Positives (Falses): Frames that are incorrectly identified as containing a shot boundary.
False Negatives (Misses): Frames that are missed by the model and do contain a shot boundary.
True Negatives (TN): Frames that are correctly identified as not containing a shot boundary.

The mathematical expression of Recall, Precision, and F1 Score is:

$$Recall = \frac{Correct}{Correct + Miss}$$

$$Precision = \frac{Correct}{Correct + False}$$

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

**Table1:**

| Trailer | C | M | F | R (%) | P (%) | F1 (%) |
|---|---|---|---|---|---|---|
| Trailer1 | 29 | 1 | 2 | 96.67 | 93.55 | 95.08 |
| Trailer2 | 22 | 1 | 2 | 95.65 | 91.67 | 93.62 |
| Trailer3 | 11 | 3 | 0 | 78.57 | 100 | 88.00 |

**Figure 1**

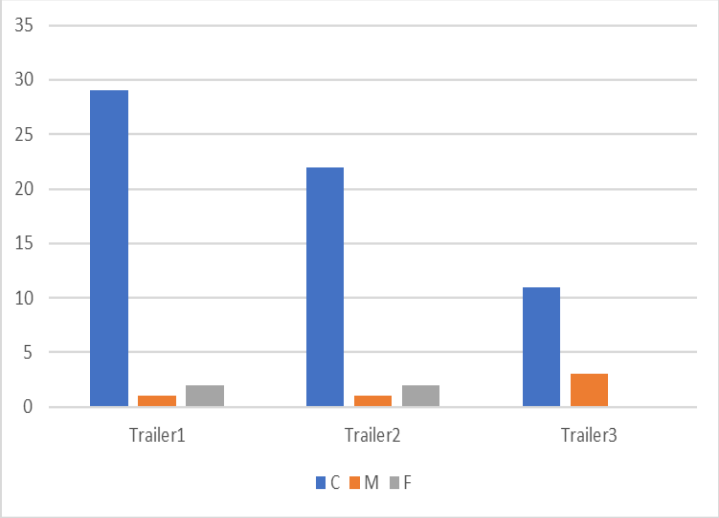
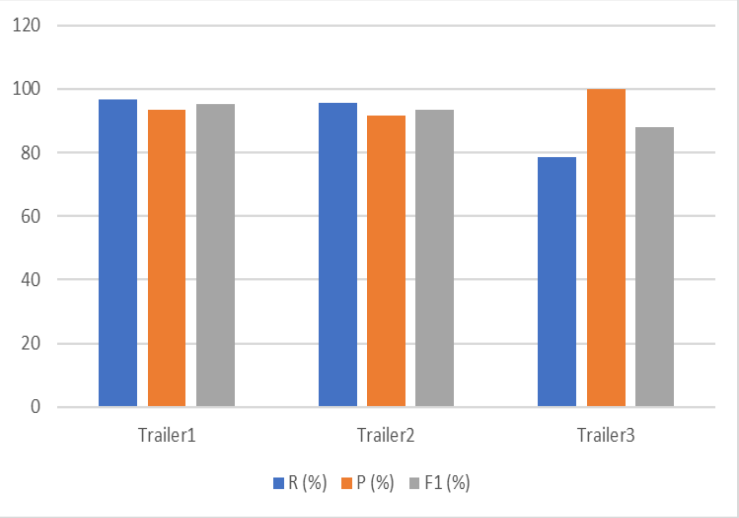
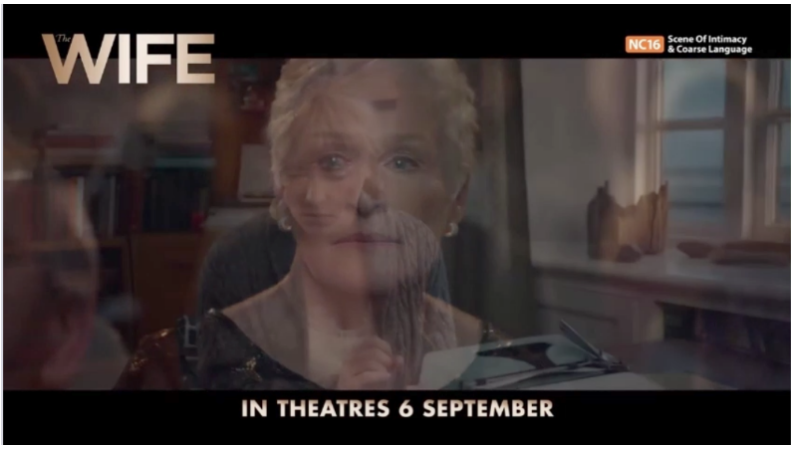



**Figure 2**




The table (**Table 1**) and graphs (**Figure1, Figure2**) show the metrics of our algorithm. As we can see the algorithm has a good metric overall. Most of the boundary can be detected if the camera break is cut. However, for the miss and false cases, I find that it is because some camera breaks of video are gradual transitions. During a gradual transition, the frame will not change significantly but gradually. The detector can't detect the boundary if the transition is smooth. As we can see from **Figure 3**.

**Figure 3**



# 3. Face detection

## 3.1 Model choice

There have been so many methods born for face detection. Some methods using deep learning concepts may have extraordinary performances compared to some older ideas. To compare the difference between them, I designed two models. The first one is called haar cascades which is first raised in 2001 and the second one is called dlib which is first raised in 2002.

### 3.1.1 Haar Cascades Classifier

The first model we are going to use is harr cascades. We know that some of the most common features on human faces are as follows:

**Haar feature:**

· *Eyes are darker than cheeks*

· *The bridge of the nose area is brighter compared to the eyes*

· *The positions of the eyes, mouth, and nose are relatively fixed*

For example, the first feature we will measure is the difference in intensity between the eye and the upper cheek. The method of eigenvalue calculation is very simple, just sum the pixels in the black area and subtract the pixels in the white area. And then, we apply this rectangle as a convolution kernel to the entire image. In order not to miss, we need to use all dimensions and positions of each convolution kernel.
To speed up this process, we use  Adaboost. Once the features are extracted, apply them to the training set using an Adaboost classifier. And then, we use the haar cascade classifier as our face detector and it detects faces by the method “detectMultiScale”.

**The following is a list of common parameters for the detectMultiScale function:**

*·scaleFactor: Determines the scale size of each image.*

*·minNeighbors: Determines how many neighbouring boxes should be kept for each candidate rectangle.*

*·minSize: The size of the smallest object. Targets smaller than this value will be ignored.*

*·maxSize: The size of the largest object. Targets larger than this value will be ignored.*

**Figure 4**


### 3.1.2 Dlip Recognizer

The second model we used is Dlib. Dlib has two methods: hog and Cnn.

Hog vs Cnn: hog is faster, CNN is more accurate

We pass two parameters to the face_recognition.face_locations method:

RGB: our RGB image.

model: CNN or hog (the value is included in the command line arguments dictionary associated with the "detection_method" key). The CNN approach is more accurate but slower. HOG is faster but less accurate.

No need to tune and face_recognition.face_location() can directly locate the faces. [4]

**Figure 5**

```
boxes = face_recognition.face_locations(rgb)
encodings = face_recognition.face_encodings(rgb, boxes)
names = []
```

## 3.2 Capture Result and Compare:

In terms of speed, HOG is the fastest algorithm, followed by the Haar cascade classifier and CNN. However, CNN in Dlib is the most accurate algorithm. HOG also performs well but has some issues recognizing smaller faces. The overall performance of Haar cascade classifiers is similar to HOG.

Below is the compared result: left is detected by Haar cascade and right is detected by Dlib recognizer

**Figure 6**

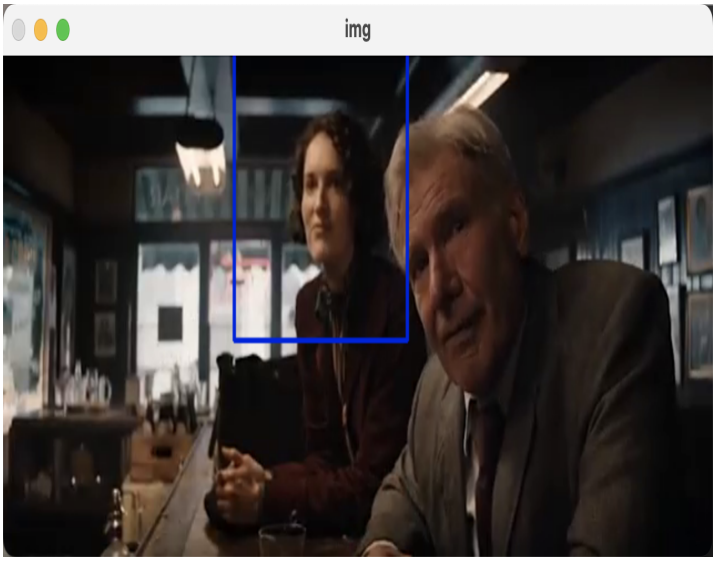



**Figure 7**

# 4. Face matching

## 4.1 Methods choose

### 4.1.1 Harr Cascade Classifier

For the Harr Cascade Classifier, we choose three recognizers to predict the actor faces in video shots: LBPHFaceRecognizer, EigenFaceRecognizer, and FisherFaceRecognizer.

We use lbphFaceRecognizer.train(), EigenFaceRecognizer.train(), and FisherFaceRecognizer.train() respectively to train the model to store the trained file as “model.yml” inside the trainer folder. [5]

**Figure 8**

```
lbph_recognizer = cv2.face.LBPHFaceRecognizer_create(1, 8, 8, 8, 123)
eigen_recognizer = cv2.face.EigenFaceRecognizer_create(10)
fisher_recognizer = cv2.face.FisherFaceRecognizer_create(10)
```

### 4.1.2 EigenFaceRecognizer:

First convert a batch of face images into a set of feature vectors, called "Eigenfaces", or "Eigenfaces", which contains the basic components of the initial training image set. The process of recognition is to project a new image into the eigenface subspace and pass its projection point in the subspace and the projection length of the line is used for judgment and identification.

After the image is transformed into another space, the images of the same category converge together, and the convergence of images of different categories is far away. In the original pixel space, it is difficult to divide the distribution of images of different categories with simple lines or planes. The specific implementation is to perform eigenvalue decomposition on the covariance matrix of all face images in the training set to obtain the corresponding eigenvectors, these eigenvectors are the "eigenfaces". Each eigenvector or eigenface is equivalent to capturing or describing a change or characteristic between human faces. This means that each face can be represented as a linear combination of these eigenfaces.

### 4.1.3 LBP Recognizer

The core idea of LBP is: to use the gray value of the central pixel as the threshold and compare it with its domain to obtain the corresponding binary code to represent the local texture features.

LBP extracts local features as the basis for discrimination. The significant advantage of the LBP method is that it is not sensitive to illumination, but it still does not solve the problem of pose and expression. However, compared with the eigenface method, the recognition rate of LBP has been greatly improved.

### 4.1.4 Fisher Recognizer

Because the set of feature vectors extracted by the fisher recognizer emphasizes the differences between different faces rather than changes in lighting conditions, facial expressions, and orientations, it is less sensitive to changes in illumination and facial posture for face recognition, which helps to improve the recognition effect.

### 4.1.5 Practice

After training each of the recognizers, we use them to predict a label based on the training set. Besides, setting a threshold to judge if the prediction is convincing or not. Like below:

**Figure 9**


### 4.1.6 Compare

Based on the training data set, LBP and Eigen have a similar result while Fisher cannot recognize the actor/actress faces correctly.

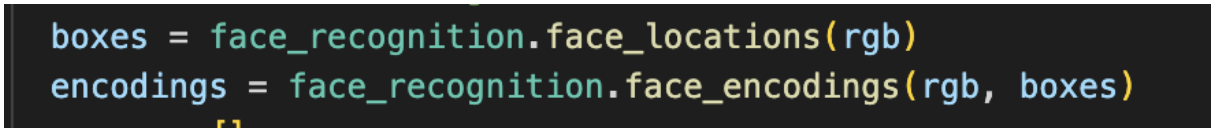
Eigenface is not computationally efficient for large datasets because of SVD decomposition, the faces in the training pictures need to be aligned, and the pictures have the same size.

LBP cannot solve the problem of pose and expression.

## 4.2 Dlib Recognizer and face_recognition

We use a loop through the video and use the shot_detect function to detect if there is a face or not. If there is not, the loop will continue to the next frame. If there is a face detected, we use face_recognition.face_locations(image) to locate the face location and use encodings = face_recognition.face_encodings(RGB, boxes)to find similar faces from our dataset.

**Figure 10**




Here, we will use a dictionary to store the score and decide which actor/actress the face belongs to based on the score. After recognizing the face, we will use cv2.rectangle to draw a square outside the face and add its corresponding name to it. [3] [6]

**Figure 11**

```
matchedIdxs = [i for (i, b) in enumerate(matches) if b]
counts = {}
# loop over the matched indexes and maintain a count for
# each recognized face face
for i in matchedIdxs:
    name = data["names"][i]
    counts[name] = counts.get(name, 0) + 1
# determine the recognized face with the largest number
# of votes (note: in the event of an unlikely tie Python
# will select first entry in the dictionary)
name = max(counts, key=counts.get)
```

## 4.3 Result

Use 45 photos per actor/actress to train the Haar Cascade classifier(9 actors/actresses in total) and the test accuracy is around 30% using LBP and Eigen recognizer. The accuracy is much lower(around 15%) using the fisher recognizer.

Use 14 photos per actor/actress to train the Dlib classifier(9 actors/actresses in total) and the test accuracy is around 90%.

This result agrees with the hypothesis we raised above:  CNN in Dlib is the most accurate algorithm while Haar Cascade has a faster speed.

Below is the different result generated by Haar Cascade(left) and Face_rocognition(right).

Left is wrong and right is correct.  [7]

**Figure 10**

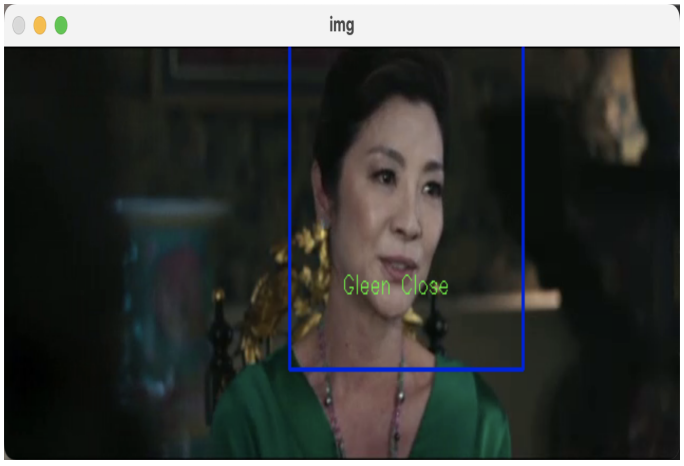



**Figure 11**

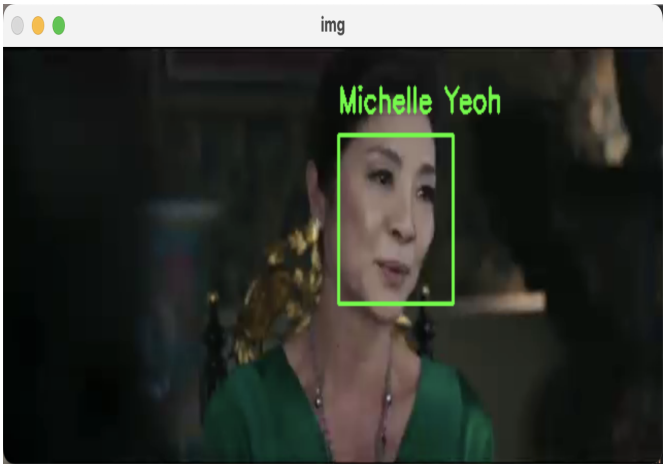
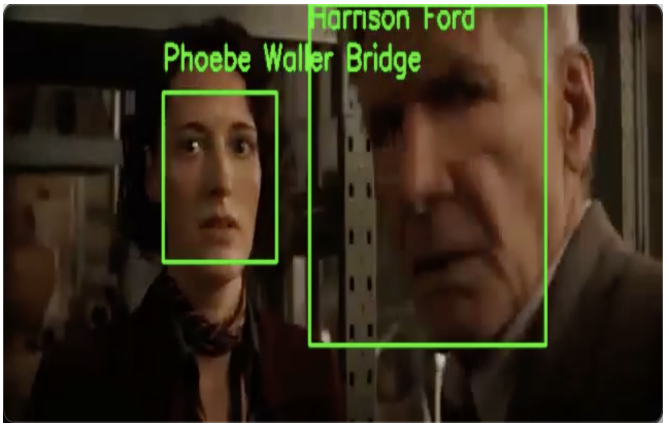



**Figure 12**


**Figure 13**



# Contribution

**Tang, Jiaguan:**

- Shot Detection
- Data Collection

**Wang, Chen:**

- Face Detection
- Face Maching

# Reference


1.

Zheng, G., & Xu, Y. (2021b). Efficient face detection and tracking in video sequences based on deep learning. *Information Sciences*, *568*, 265–285. https://doi.org/10.1016/j.ins.2021.03.027

2.

Ngo, C. (2009). Video Shot Detection. *Springer eBooks*, 3316–3320.

https://doi.org/10.1007/978-0-387-39940-9_1021

3.

Davisking. (n.d.). *GitHub - davisking/dlib-models: Trained model files for dlib example programs.* GitHub.

https://github.com/davisking/dlib-models

4.

Rosebrock, A. (2023, March 23). *Face detection with OpenCV and deep learning - PyImageSearch*.

PyImageSearch. https://pyimagesearch.com/2018/02/26/face-detection-with-opencv-and-deep-learning/

5.

*Video Shot Detection Techniques Brief Overview*

http://www.ijergs.org/files/documents/VIDEO-108.pdf

6.

Rosebrock, A. (2021, April 17). *Faster video file FPS with cv2.VideoCapture and OpenCV - PyImageSearch*.

PyImageSearch.

https://pyimagesearch.com/2017/02/06/faster-video-file-fps-with-cv2-videocapture-and-opencv/

7.

Suárez, I. (2021, December 27). Improving your image matching results by 14% with one line of code.

*Medium*.

https://towardsdatascience.com/improving-your-image-matching-results-by-14-with-one-line-of-code-b72ae9ca2b73

# Appendix

## Shot Detection Python code:

```
import cv2
from google.colab.patches import cv2_imshow
import matplotlib.pyplot as plt
import numpy as np


movie1 = cv2.VideoCapture("Movie_1_new.mp4")
movie2 = cv2.VideoCapture("Movie_2_new.mp4")
movie3 = cv2.VideoCapture("Movie_3_new.mp4")
```

```
def shot_detector(video):
    # Set the threshold for shot detection
    threshold = 7
    # Initialize variables
    frame_count = 0
    prev_frame = None
    shots = []
    dif = []
    start = True
    skip = True

    while True:
            # Read the next frame
            ret, frame = video.read()

            # If there are no more frames, break out of the loop
            if not ret:
                if shots != []:
                    shots[-1][-1] = frame_count
                break

            # Convert the frame to grayscale
            gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

            # If this is not the first frame, compare it to the previous frame
            if prev_frame is not None:

                diff = cv2.absdiff(gray_frame, prev_frame)

                if np.average(diff) > 0.2 or not skip:
                    if len(dif) > 0:
                        if np.average(diff) - dif[-1] > threshold:
                            shots[-1][-1]=frame_count - 1
                            shots.append([frame_count, frame_count])
                    dif.append(np.average(diff))
                    skip = True
                elif skip:
                    shots[-1][-1] = frame_count
                    skip = False

            else:
                shots.append([frame_count, frame_count])
                start = False

            # Update variables
            frame_count += 1
            prev_frame = gray_frame

    return shots
```

```
def accu(output, truth):
    out_put = []
    truth_ = []
    correct = 0
    miss = 0
    false = 0
    for i in range(1, len(output)):
        out_put.append(output[i][0])
    for i in range(1, len(truth)):
        truth_.append(truth[i][0])
    for i in truth_:
        if i in out_put:
            correct += 1
        else:
            miss += 1
    for i in out_put:
        if i not in truth_:
            false += 1
    return correct, miss, false
```

```
o1 = shot_detector(movie1)
o2 = shot_detector(movie2)
o3 = shot_detector(movie3)
```

```
print("Video1_output: \n", o1)
print("Video2_output: \n", o2)
print("Video2_output: \n", o3)
```

```
Video1_output:
 [[0, 50], [51, 92], [93, 123], [124, 192], [193, 222], [223, 261], [262, 300], [301, 316], [317, 339], [340, 391], [392, 412], [413, 429], [430
Video2_output:
 [[0, 51], [52, 77], [78, 109], [110, 149], [150, 211], [212, 250], [251, 278], [279, 296], [297, 328], [329, 360], [361, 391], [392, 418], [419
Video2_output:
 [[0, 4], [5, 45], [46, 80], [81, 156], [157, 191], [192, 233], [234, 274], [275, 315], [316, 345], [346, 492], [493, 522], [523, 586]]
```

```
t1 = [[0, 50], [51, 92], [93, 123], [124, 192], [193, 222], [223,261], [262, 300], [301, 316], [317, 339],
      [340, 391], [392, 412], [413, 429], [430, 454], [455, 456], [457, 514], [515, 552], [553, 596],
      [597, 611], [612, 635], [636,659], [660, 686], [687, 711], [712, 735], [736, 763], [764,791],
      [792, 805], [806, 825], [826, 845], [846, 870], [871, 886], [887, 912]]
t2 = [[0, 51], [52, 77], [78, 109], [110, 149], [150, 211], [212, 250], [251, 278], [279, 296],
      [297, 328], [329, 360], [361, 391], [392, 418], [419, 482], [483, 513], [514, 525],
      [526, 541], [542, 566], [567, 585], [586, 615], [616, 641], [642, 693], [694, 728],
      [729, 757], [578, 776]]
t3 = [[0, 4], [5, 45], [46, 80], [81, 156], [157, 191], [192, 233], [234, 274], [275, 315],
      [316, 345], [346, 381], [382, 424], [425, 469], [470, 492], [493, 522], [523, 586]]
```

```
Movie1_accuracy = accu(o1, t1)
Movie2_accuracy = accu(o2, t2)
Movie3_accuracy = accu(o3, t3)
```

```
print("The num of correct, miss, false of Movie1: \n", Movie1_accuracy)
print("The num of correct, miss, false of Movie2: \n", Movie2_accuracy)
print("The num of correct, miss, false of Movie3: \n", Movie3_accuracy)
```

```
The num of correct, miss, false of Movie1:
 (29, 1, 2)
The num of correct, miss, false of Movie2:
 (22, 1, 2)
The num of correct, miss, false of Movie3:
 (11, 3, 0)
```

## Face Detection Python code:

```python
import cv2
import numpy as np
from imutils.video import VideoStream
import face_recognition
import argparse
import imutils
import pickle
from imutils import paths
import argparse
import os
from PIL import Image

class dlib:
    def __init__(self):
        self.video_frames = []

    def create(self):
        ap = argparse.ArgumentParser()
        ap.add_argument("-i", "--dataset", required=True,
        help="path to input directory of faces + images")
        ap.add_argument("-e", "--encodings", required=True,
        help="path to serialized db of facial encodings")
        ap.add_argument("-d", "--detection-method", type=str, default="cnn",
        help="face detection model to use: either `hog` or `cnn`")
        args = vars(ap.parse_args())

        # grab the paths to the input images in our dataset
        print("[INFO] quantifying faces...")
        imagePaths = list(paths.list_images(args["dataset"]))
        # initialize the list of known encodings and known names
```

```python
        knownEncodings = []
        knownNames = []

        # loop over the image paths
        for (i, imagePath) in enumerate(imagePaths):
            # extract the person name from the image path
            print("[INFO] processing image {}/{}".format(i + 1,
                len(imagePaths)))
            name = 'unknown'
            tag = int(imagePath.split(os.path.sep)[-1].split('.')[1])
            if tag == 1:
                name = "Thomas Kretschman"
            elif tag == 2:
                name = "Harrison Ford"
            elif tag == 3:
                name = "Phoebe Waller Bridge"
            elif tag == 4:
                name = "Michelle Yeoh"
            elif tag == 5:
                name = "Henry Golding"
            elif tag == 6:
                name = "Constance Wu"
            elif tag == 7:
                name = "Jonathan Pryce"
            elif tag == 8:
                name = "Max Irons"
            elif tag == 9:
                name = "Gleen Close"
            # name = imagePath.split(os.path.sep)[-2]
            # load the input image and convert it from BGR (OpenCV ordering)
```

```
61              # to dlib ordering (RGB)
62              image = cv2.imread(imagePath)
63              rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
64
65              # detect the (x, y)-coordinates of the bounding boxes
66              # corresponding to each face in the input image
67              boxes = face_recognition.face_locations(rgb,
68                  model=args["detection_method"])
69              # compute the facial embedding for the face
70              encodings = face_recognition.face_encodings(rgb, boxes)
71              # loop over the encodings
72              for encoding in encodings:
73                  # add each encoding + name to our set of known names and
74                  # encodings
75                  knownEncodings.append(encoding)
76                  knownNames.append(name)
77
78          # dump the facial encodings + names to disk
79          print("[INFO] serializing encodings...")
80          data = {"encodings": knownEncodings, "names": knownNames}
81          f = open(args["encodings"], "wb")
82          f.write(pickle.dumps(data))
83          f.close()
84
85      def phototest(self):
86          ap = argparse.ArgumentParser()
87          ap.add_argument("-e", "--encodings", required=True,
88          help="path to serialized db of facial encodings")
89          ap.add_argument("-i", "--image", required=True,
90          help="path to input image")
```

```
        ap.add_argument("-d", "--detection-method", type=str, default="cnn",
        help="face detection model to use: either `hog` or `cnn`")
        args = vars(ap.parse_args())
        # load the known faces and embeddings
        print("[INFO] loading encodings...")
        data = pickle.loads(open(args["encodings"], "rb").read())
        # load the input image and convert it from BGR to RGB
        image = cv2.imread(args["image"])
        rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        # detect the (x, y)-coordinates of the bounding boxes corresponding
        # to each face in the input image, then compute the facial embeddings
        # for each face
        boxes = face_recognition.face_locations(rgb,
        model=args["detection_method"])
        encodings = face_recognition.face_encodings(rgb, boxes)
        # initialize the list of names for each face detected
        names = []
        # loop over the facial embeddings
        for encoding in encodings:
            # attempt to match each face in the input image to our known
            # encodings
            matches = face_recognition.compare_faces(data["encodings"],
                encoding)
            name = "Unknown"
            # check to see if we have found a match
            if True in matches:
                # find the indexes of all matched faces then initialize a
                # dictionary to count the total number of times each face
                # was matched
                matchedIdxs = [i for (i, b) in enumerate(matches) if b]
```

```
                counts = {}
                # loop over the matched indexes and maintain a count for
                # each recognized face face
                for i in matchedIdxs:
                    name = data["names"][i]
                    counts[name] = counts.get(name, 0) + 1
                # determine the recognized face with the largest number of
                # votes (note: in the event of an unlikely tie Python will
                # select first entry in the dictionary)
                name = max(counts, key=counts.get)

            # update the list of names
            names.append(name)
        # loop over the recognized faces
        for ((top, right, bottom, left), name) in zip(boxes, names):
        # draw the predicted face name on the image
            cv2.rectangle(image, (left, top), (right, bottom), (0, 255, 0), 2)
            y = top - 15 if top - 15 > 15 else top + 15
            cv2.putText(image, name, (left, y), cv2.FONT_HERSHEY_SIMPLEX,
                0.75, (0, 255, 0), 2)
        # show the output image
        cv2.imshow("Image", image)
        cv2.waitKey(0)

    def shot_detector(self, video_name):
        video = cv2.VideoCapture(video_name)

        # Set the threshold for shot detection
        threshold = 6

```

```
    # Initialize variables
    frame_count = 0
    prev_frame = None
    shots = []
    dif = []
    start = True

    while True:
        # Read the next frame
        ret, frame = video.read()

        # If there are no more frames, break out of the loop
        if not ret:
            break

        # Convert the frame to grayscale
        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # If this is not the first frame, compare it to the previous frame
        # print(prev_frame)
        if prev_frame is not None:

            diff = cv2.absdiff(gray_frame, prev_frame)

            if len(dif) > 0:
                # print(np.average(diff) - dif[-1] > threshold,start)
                if dif[-1] - np.average(diff) > threshold and start:

                    shots.append([frame_count, frame_count])

                    # plt.imshow(prev_frame, cmap='gray')
                    # plt.axis("off")
                    # plt.title("start at " + str(frame_count))
                    # plt.show()
                    start = False
                elif np.average(diff) - dif[-1] > threshold and not start:

                    # plt.imshow(prev_frame, cmap='gray')
                    # plt.axis("off")
                    # plt.title("end at " + str(frame_count))
                    # plt.show()


                    shots[-1][-1] = frame_count
                    start = True
                dif.append(np.average(diff))

            else:
                shots.append([frame_count, frame_count])

                # plt.imshow(gray_frame, cmap='gray')
                # plt.axis("off")
                # plt.title("start at " + str(frame_count))
                # plt.show()

                start = False
                dif.append(0)

            # Update variables
            frame_count += 1
```

```
            frame_count += 1
            prev_frame = gray_frame

        # Print the shot boundaries
        return shots

    def has_face(self, img):
        rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        rgb = imutils.resize(img, width=750)
        boxes = face_recognition.face_locations(rgb)
        if boxes:
            return True
        return False

    def inShot(self, index, l):
        for item in l:
            if item[0] <= index <= item[1] and (self.has_face(self.video_frames[item[0]]) or
                                                self.has_face(self.video_frames[item[1]])):
                return True
        return False

    def videotest(self):
        ap = argparse.ArgumentParser()
        ap.add_argument("-e", "--encodings", required=True,
        help="path to serialized db of facial encodings")
        ap.add_argument("-v", "--video", required=True,
            help="path to input video file")
        ap.add_argument("-o", "--output", type=str,
        help="path to output video")
        # ap.add_argument("-y", "--display", type=int, default=1,
        #   help="whether or not to display output frame to screen")
```

```
        # ap.add_argument("-d", "--detection-method", type=str, default="cnn",
        #   help="face detection model to use: either `hog` or `cnn`")
        args = vars(ap.parse_args())
        # load the known faces and embeddings
        data = pickle.loads(open(args["encodings"], "rb").read())
        # initialize the video stream and pointer to output video file, then
        # allow the camera sensor to warm up
        vs = cv2.VideoCapture(args["video"])
        # vs = VideoStream(src=0).start()
        writer = None
        video = cv2.VideoCapture(args["video"])
        while True:
            # Read the next frame
            ret, frame = video.read()
            if frame is not None:
                self.video_frames.append(frame)
            # If there are no more frames, break out of the loop
            if not ret:
                break
        video.release()
        shotBoundaryIndex = self.shot_detector(args["video"])
        i = 0
        # loop over frames from the video file stream
        while True:
        # grab the frame from the threaded video stream
            ret, frame = vs.read()
            if not ret:
                break
        # convert the input frame from BGR to RGB then resize it to have
        # a width of 750px (to speedup processing)
```

```
271                 if self.inShot(i, shotBoundaryIndex):
272                     rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
273                     rgb = imutils.resize(frame, width=750)
274                     r = frame.shape[1] / float(rgb.shape[1])
275                     # detect the (x, y)-coordinates of the bounding boxes
276                     # corresponding to each face in the input frame, then compute
277                     # the facial embeddings for each face
278                     boxes = face_recognition.face_locations(rgb)
279                     encodings = face_recognition.face_encodings(rgb, boxes)
280                     names = []
281                 # loop over the facial embeddings
282                     for encoding in encodings:
283                     # attempt to match each face in the input image to our known
284                     # encodings
285                         matches = face_recognition.compare_faces(data["encodings"],
286                             encoding)
287                         name = "Unknown"
288                     # check to see if we have found a match
289                         if True in matches:
290                             # find the indexes of all matched faces then initialize a
291                             # dictionary to count the total number of times each face
292                             # was matched
293                             matchedIdxs = [i for (i, b) in enumerate(matches) if b]
294                             counts = {}
295                             # loop over the matched indexes and maintain a count for
296                             # each recognized face face
297                             for i in matchedIdxs:
298                                 name = data["names"][i]
299                                 counts[name] = counts.get(name, 0) + 1
300                             # determine the recognized face with the largest number
```

```
301                             # of votes (note: in the event of an unlikely tie Python
302                             # will select first entry in the dictionary)
303                             name = max(counts, key=counts.get)
304                             if counts[name] <= 3 :
305                                 name = 'Unknown'
306 
307                         # update the list of names
308                         names.append(name)
309                 # loop over the recognized faces
310                     for ((top, right, bottom, left), name) in zip(boxes, names):
311                         # rescale the face coordinates
312                         top = int(top * r)
313                         right = int(right * r)
314                         bottom = int(bottom * r)
315                         left = int(left * r)
316                         # draw the predicted face name on the image
317                         cv2.rectangle(frame, (left, top), (right, bottom),
318                             (0, 255, 0), 2)
319                         y = top - 15 if top - 15 > 15 else top + 15
320                         cv2.putText(frame, name, (left, y), cv2.FONT_HERSHEY_SIMPLEX,
321                             0.75, (0, 255, 0), 2)
322                 i += 1
323                 cv2.imshow('img',frame)
324                 key = cv2.waitKey(1)
325                 if key == ord('q'):
326                     break
327                 # if the video writer is None *AND* we are supposed to write
328                 # the output video to disk initialize the writer
329                 if writer is None and args["output"] is not None:
330                     fourcc = cv2.VideoWriter_fourcc(*"MJPG")
```

```
                writer = cv2.VideoWriter(args["output"], fourcc, 20,
                (frame.shape[1], frame.shape[0]), True)
            # if the writer is not None, write the frame with recognized
            # faces to disk
            if writer is not None:
                writer.write(frame)
        # check to see if we are supposed to display the output frame to
        # the screen
        #   if args["display"] > 0:
        #     cv2.imshow("Frame", frame)
        #     key = cv2.waitKey(0) & 0xFF
        #     # if the `q` key was pressed, break from the loop
        #     if key == ord("q"):
        #       break
        # do a bit of cleanup
        cv2.destroyAllWindows()
        vs.release()
        # check to see if the video writer point needs to be released
        if writer is not None:
            writer.release()


class haarcascade:
    def __init__(self):
        self.detector = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_alt.xml')
        self.lbph_recognizer = cv2.face.LBPHFaceRecognizer_create(1, 8, 8, 8, 123)
        self.eigen_recognizer = cv2.face.EigenFaceRecognizer_create(10)
        self.fisher_recognizer = cv2.face.FisherFaceRecognizer_create(10)
        self.videoName = 'video/Movie_2.mp4'
        self.cascade_path = cv2.data.haarcascades + "haarcascade_frontalface_alt.xml"
```

```
        self.face_cascade = cv2.CascadeClassifier(self.cascade_path)
        self.video_frames = []

    def procress(self):
        sampleNum = 1
        Id = 0
        pic = 1

        while True:
            if sampleNum > 14:
                sampleNum -= 14
            if pic <= 14:
                Id = 1
            elif 14 < pic <= 28:
                Id = 2
            elif 28 < pic <= 42:
                Id = 3
            elif 42 < pic <= 56:
                Id = 4
            elif 56 < pic <= 70:
                Id = 5
            elif 70 < pic <= 84:
                Id = 6
            elif 84 < pic <= 98:
                Id = 7
            elif 98 < pic <= 112:
                Id = 8
            elif 112 < pic <= 126:
                Id = 9
            if Id == 1:
```

```
            filename = './actor/thomas kretschmann.' + str(Id) + '.' + str(sampleNum) + '.jpeg'
        elif Id == 2:
            filename = './actor/harrison ford.' + str(Id) + '.' + str(sampleNum) + '.jpeg'
        elif Id == 3:
            filename = './actor/phoebe waller bridge.' + str(Id) + '.' + str(sampleNum) + '.jpeg'
        elif Id == 4:
            filename = './actor/Michelle Yeoh.' + str(Id) + '.' + str(sampleNum) + '.jpeg'
        elif Id == 5:
            filename = './actor/Henry Golding.' + str(Id) + '.' + str(sampleNum) + '.jpeg'
        elif Id == 6:
            filename = './actor/COnstance Wu.' + str(Id) + '.' + str(sampleNum) + '.jpeg'
        elif Id == 7:
            filename = './actor/Jonathan Pryce.' + str(Id) + '.' + str(sampleNum) + '.jpeg'
        elif Id == 8:
            filename = './actor/Max Irons.' + str(Id) + '.' + str(sampleNum) + '.jpeg'
        elif Id == 9:
            filename = './actor/Glenn Close.' + str(Id) + '.' + str(sampleNum) + '.jpeg'
        img = cv2.imread(filename)
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        faces = self.detector.detectMultiScale(gray, 1.3, 5, minSize=(50,50))
        if type(faces) == np.ndarray:
            for (x, y, w, h) in faces:
                cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)
                f = cv2.resize(gray[y:y+h, x:x+w], (200,200))
                # saving the captured face in the dataset folder
                cv2.imwrite("dataSet/Actor." + str(Id) + '.' + str(sampleNum) + ".jpg", f)

            cv2.imshow('frame', img)
            # wait for 100 miliseconds
        sampleNum += 1
```

```
            pic += 1
            if cv2.waitKey(50) & 0xFF == ord('q'):
                break
            if pic > 135:
                break

        cv2.destroyAllWindows()

    def shot_detector(self, video_name):
        video = cv2.VideoCapture(video_name)

        # Set the threshold for shot detection
        threshold = 6

        # Initialize variables
        frame_count = 0
        prev_frame = None
        shots = []
        dif = []
        start = True

        while True:
            # Read the next frame
            ret, frame = video.read()

            # If there are no more frames, break out of the loop
            if not ret:
                break

            # Convert the frame to grayscale
```

```
            gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

            # If this is not the first frame, compare it to the previous frame
            # print(prev_frame)
            if prev_frame is not None:

                diff = cv2.absdiff(gray_frame, prev_frame)

                if len(dif) > 0:
                    # print(np.average(diff) - dif[-1] > threshold,start)
                    if dif[-1] - np.average(diff) > threshold and start:

                        shots.append([frame_count, frame_count])

                        # plt.imshow(prev_frame, cmap='gray')
                        # plt.axis("off")
                        # plt.title("start at " + str(frame_count))
                        # plt.show()
                        start = False
                    elif np.average(diff) - dif[-1] > threshold and not start:

                        # plt.imshow(prev_frame, cmap='gray')
                        # plt.axis("off")
                        # plt.title("end at " + str(frame_count))
                        # plt.show()


                        shots[-1][-1] = frame_count
                        start = True
                    dif.append(np.average(diff))
```

```
481
482             else:
483                 shots.append([frame_count, frame_count])
484
485                 # plt.imshow(gray_frame, cmap='gray')
486                 # plt.axis("off")
487                 # plt.title("start at " + str(frame_count))
488                 # plt.show()
489
490                 start = False
491                 dif.append(0)
492
493             # Update variables
494             frame_count += 1
495             prev_frame = gray_frame
496
497         # Print the shot boundaries
498         return shots
499
500     def train(self):
501         faces, Ids = self.get_images_and_labels('dataSet')
502         self.lbph_recognizer.train(faces, np.array(Ids))
503         self.eigen_recognizer.train(faces, np.array(Ids))
504         self.fisher_recognizer.train(faces, np.array(Ids))
505         self.lbph_recognizer.save('haarcascades/trainner/lbph_trainner.yml')
506         self.eigen_recognizer.save('haarcascades/trainner/eigen_trainner.yml')
507         self.fisher_recognizer.save('haarcascades/trainner/fisher_trainner.yml')
508
509     def get_images_and_labels(self,path):
510         image_paths = [os.path.join(path, f) for f in os.listdir(path)]
```

```
        face_samples = []
        ids = []

        for image_path in image_paths:
            if image_path == 'dataSet/' + '.DS_Store':
                continue
            image = Image.open(image_path).convert('L')
            image_np = np.array(image, 'uint8')
            if os.path.split(image_path)[-1].split(".")[-1] != 'jpg':
                continue
            image_id = int(os.path.split(image_path)[-1].split(".")[1])
            faces = self.detector.detectMultiScale(image_np, 1.1, 5, minSize=(50,50))
            for (x, y, w, h) in faces:
                face_samples.append(cv2.resize(image_np[y:y + h, x:x + w],(200,200)))
                ids.append(image_id)

        return face_samples, ids

    def createframe(self):
        video = cv2.VideoCapture(self.videoName)
        while True:
            # Read the next frame
            ret, frame = video.read()
            if frame is not None:
                self.video_frames.append(frame)
            # If there are no more frames, break out of the loop
            if not ret:
                break
        video.release()

```

```
    def has_face(self, img):
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        faces = self.face_cascade.detectMultiScale(gray, 1.1, 5, minSize=(15,15))
        if type(faces) == np.ndarray:
            return True
        return False

    def inShot(self, index, l):
        for item in l:
            if item[0] <= index <= item[1] and (self.has_face(self.video_frames[item[0]]) or
                                                self.has_face(self.video_frames[item[1]])):
                return True
        return False

    def testshot(self):
        self.lbph_recognizer.read('haarcascades/trainner/lbph_trainner.yml')
        self.eigen_recognizer.read('haarcascades/trainner/eigen_trainner.yml')
        self.fisher_recognizer.read('haarcascades/trainner/fisher_trainner.yml')

        shotBoundaryIndex = self.shot_detector(self.videoName)
        # font = cv2.cv.InitFont(cv2.cv.CV_FONT_HERSHEY_SIMPLEX, 1, 1, 0, 1, 1) # in OpenCV 2
        font = cv2.FONT_HERSHEY_SIMPLEX
        video = cv2.VideoCapture(self.videoName)
        v = cv2.VideoCapture(self.videoName)
        while True:
            # Read the next frame
            ret, frame = v.read()
            if frame is not None:
                self.video_frames.append(frame)
            # If there are no more frames, break out of the loop
```

```
            if not ret:
                break
        v.release()
        i = 0
        while True:
            ret, img = video.read()
            if not ret:
                break
            if self.inShot(i, shotBoundaryIndex):
                gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
                faces = self.face_cascade.detectMultiScale(gray, 1.2, 5, minSize=(15,15))
                for (x, y, w, h) in faces:
                    face = gray[y:y+h, x:x+w]
                    # k1 = np.array([[y, x],[y+h,x],[y,x+w],[y+h,x+w]])
                    # k2 = np.array([[0,0],[0,200],[200,0],[200,200]])
                    # h,mark = cv2.findHomography(k1,k2,cv2.RANSAC)
                    # face_test = cv2.warpPerspective(face, h, (200,200))
                    face_test = cv2.resize(face, (200, 200))
                    cv2.rectangle(img, (x - 50, y - 50), (x + w + 50, y + h + 50), (225, 0, 0), 2)
                    # img_id, conf = fisher_recognizer.predict(face_test)
                    # if conf > 80:
                    #     if img_id == 1:
                    #         img_id = 'Thomas Kretschman'
                    #     elif img_id == 2:
                    #         img_id = 'Harrison Ford'
                    #     elif img_id == 3:
                    #         img_id = 'Phoebe Waller Bridge'
                    #     # elif img_id == 4:
                    #     #     img_id = 'Shaunette Renée Wilson'
                    # else:
```

```
                    #     img_id = "Unknown"
                    predictPCA = 0
                    if not face_test is None:
                        predictPCA, conf = self.eigen_recognizer.predict(face_test)
                    if conf > 90:
                        if predictPCA == 1:
                            name = "Thomas Kretschman"
                        elif predictPCA == 2:
                            name = "Harrison Ford"
                        elif predictPCA == 3:
                            name = "Phoebe Waller Bridge"
                        elif predictPCA == 4:
                            name = "Michelle Yeoh"
                        elif predictPCA == 5:
                            name = "Henry Golding"
                        elif predictPCA == 6:
                            name = "Constance Wu"
                        elif predictPCA == 7:
                            name = "Jonathan Pryce"
                        elif predictPCA == 8:
                            name = "Max Irons"
                        elif predictPCA == 9:
                            name = "Gleen Close"
                    else:
                        name = 'unknown'
                    cv2.putText(img, str(name), (x, y + h), font, 0.55, (0, 255, 0), 1)
            i += 1
            cv2.imshow('img',img)
            key = cv2.waitKey(5)
            if key == ord('q'):
```

```
631                     break
632
633                 # if cv2.waitKey(100) & 0xFF == ord('q'):
634                 #     break
635             video.release()
636             cv2.destroyAllWindows()
637
638
639
640     if __name__ == '__main__':
641         h = haarcascade()
642         h.testshot()
643         # d = dlib()
644         # d.videotest()
```