**Sai Deepak Reddy Konreddy**

**WSBY161652658**

**Complete real time Face Detection using Open CV and Haar Cascade Classifiers.**

**Abstract:**

This project utilizes the OpenCV library, haar cascade classifier for face detection, eye detection, nose detection and smile detection in a live web camera. Haar Cascade Classifier picks a location from the image, crops a sub-image with the selected pixel as the centre and runs different filters over that space trying to localize a face. For training purpose for the feature "face", we used the “haarcascade\_frontalface\_default” XML file, for feature "eye" we used “haarcascade\_eye” and respectively for "nose " and "smile" we used “haar\_mcs\_nose” and "haarcascade\_smile" XML files. The agenda of this model is to successfully recongnize the face of a person along with eyes, nose and smile with maximum efficiency and accuracy.

**Objective:**

The main objective of this project is by using Haar Feature-based Cascade Classifiers and open CV to develop a framework for face detection.

Firstly, face is detected and later we will be detecting specific features such as eyes, nose, and smile for a real time web cam.

Outline

•Fundamentals

•Cascades of classifiers

•Face detection in an image

•Eyes detection in a face

•Nose detection in a face

•Results

•Face detection

•Eyes detection

•Nose detection

•Conclusion

•Fundamentals

•Cascades of classifiers

•Face detection in an image

•Eyes detection in a face

•Nose detection in a face

•Results

•Face detection

•Eyes detection

•Nose detection

•Conclusion

•Fundamentals

•Cascades of classifiers

•Face detection in an image

•Eyes detection in a face

•Nose detection in a face

•Results

•Face detection

•Eyes detection

•Nose detection

•Conclusion

•Fundamentals

•Cascades of classifiers

•Face detection in an image

•Eyes detection in a face

•Nose detection in a face

•Results

•Face detection

•Eyes detection

•Nose detection

•Conclusion

•Fundamentals

•Cascades of classifiers

•Face detection in an image

•Eyes detection in a face

•Nose detection in a face

•Results

•Face detection

•Eyes detection

•Nose detection

•Conclusion

* Fundamentals
* Cascades of classifiers
* Face Detection in an image
* Eyes Detection in face
* Nose Detection in face
* Smile Detection in face
* Results:

Face Detection

Eyes Detection

Nose Detection

Smile Detection

* Conclusion

# Fundamentals:

* Object detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their seminal paper titled “Rapid Object Detection using a Boosted Cascade of Simple Features” in 2001.
* It is a machine learning-based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.
* The detection algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it. For this, Haar features shown in the images are used.
* They are just like our convolutional kernel. Each feature is a single value obtained by subtracting the sum of pixels under the white rectangle from the sum of pixels under the black rectangle



# **Fundamentals:**

* Now all possible sizes and locations of each kernel are used to calculate plenty of features. Even a 24\*24 window results in over 160000 features.
* For each feature calculation, we need to find the sum of pixels under white and black rectangles. To solve this problem, Viola and Jones introduced the concept of “integral images.” It simplifies the calculation of the sum of pixels, however large may be the number of pixels, to an operation involving just four pixels. This makes the computation fast.
* Consider the following image. If we want to compute the sum of the rectangle ABCD, we do not need to go through each pixel in that rectangular area.
* Let us say OC indicates the area of the rectangle formed by the top left corner O and the point C on the diagonally opposite corners of the rectangle. To calculate the area of the rectangle ABCD, we can use the following:
* Area of the rectangle ABCD = OC – (OB + OD –OA)
* We do not have to iterate through anything or re-compute any rectangle areas. All the values on the right-hand side of the equation are already available as they were computed during the earlier cycles.

# **Fundamentals:**

Most of the features extracted, however, are irrelevant.

•In the image below, the top row shows two good features. The first feature

selected seems to focus on the property that the region of the eyes is often

darker than the region of the nose and cheeks. The second feature selected

relies on the property that the eyes are darker than the bridge of the nose.

However, the same windows applying on cheeks or any other place is

irrelevant.

* Most of the features extracted, however, are irrelevant.
* In the image below, the top row shows two good features. The first feature selected seems to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks.
* The second feature selected relies on the property that the eyes are darker than the bridge of the nose. However, the same windows applying on cheeks or any other place is irrelevant.



# **Fundamentals:**

* We apply each feature on all the training images. For each feature, it finds the best threshold which will classify the faces to positive and negative. However, there will be misclassifications too.
* We select the features with a minimum error rate, which means they are the features that best classifies the face and non-face images.
* The above process is carried out as follows. Each image is given an equal weight at the start. After each classification step, weights of misclassified images are increased.
* Then again, the same process is repeated, new error rates and new weights are computed. The process is continued until the required accuracy or error rate is achieved or the required number of features is found.
* The final classifier is a weighted sum of these weak classifiers. The classifiers are called weak because they individually cannot classify the image but together can form a very powerful classifier.
* The original work of Viola and Jones involved 200 features and that yielded 95% accuracy. The final setup had around 6000 features ( a reduction from 160000+ features)

**Fundamentals:**

* So now we take an image. Take each 24\*24 window. Apply 6000 features to it. Then, check if it is a face or not. Of course, a very inefficient and time-consuming proposition. However, Viola and Jones had a smarter proposition.
* In an image, most of the image region is a non-face region. Hence, it is a better idea to have a simple method to check if a window is not a face region.
* If it is not, we discard it straight away.
* To implement the above concept, Viola and Jones introduced the concept of Cascade of Classifiers.
* Instead of applying all the 6000 features on a window, we group the features into different stages of classifiers and apply one-by-one. Usually, the first few stages will contain very few numbers of features.
* If a window fails in the first stage, we discard it and don’t consider remaining features on it.
* If a window passes the first stage, we apply the second stage of features and continue with the process. A window that passes all stages is a face region.
* The detector of Viola and Jones has 6000+ features with 38 stages with 1, 10, 25, 15, and 50 features in the first five stages.

**Cascade of Classifiers:**

* OpenCV comes with a lot of pre-trained classifiers. For instance, there are classifiers for smile, eyes, face, nose, etc.
* These come in the form of XML files and are located in the location: opencv/data/haarcascades/ folder.
* We have downloaded the XML files from this link and placed them in the “data” folder in the same working directory as the “jupyter” notebook.
* For “Face detection” we have used the “detectMultiScale” module of the classifier. The function returns a rectangle with coordinates (x, y, w, h) around the detected face.
* The function has two important parameters that have to be tuned according to the data –“scaleFactor” and “minNeighbors”. We will discuss their respective roles when we discuss our implementation.

**Face Detection in Image:**

* We first loaded the Haar cascade file of the frontal face using the CascadeClassifier method of cv2 module using the following command:

face\_cas = cv2.CascadeClassifier('C:\\Users\\saide\\Documents\\Haar Cascade\\haarcascade\_frontalface\_default.xml')

* Next, we read the RGB image from webcam in which faces are to be detected using the imread function of the cv2 module.

vcap = cv2.VideoCapture(0)

vcap.set(3,1280)

vcap.set(4,7200)

while(vcap.isOpened()):

# Captures video\_capture frame by frame

ret, image = vcap.read()

* Next, we converted the RGB image into a grayscale image using the function cvtColor defined in the module cv2. The function cvtColor needs two mandatory parameters “src” –the image whose color space is to be changed, and “code” –the color space conversion code. We have used cv2.COLOR\_BGR2GRAY, as we need to convert the source BGR image into a grayscale

bw = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

* Next, we run the face detector on the grayscale image using the following line with “scaleFactor” value 1.3 (default value) and the minNeighbors value 5

face = face\_cas.detectMultiScale(bw, 1.3, 5)

**Face Detection in Image:**

* The first parameter “scaleFactor” of the detectMultiScale function determines a trade-off between detection accuracy and speed of detection. The detection window starts at size “minSize”, and after testing all windows of that size, the window is scaled up the “scaleFactor”, and re-tested, and so on until the window reaches or exceeds the “maxSize”. If the “scaleFactor” is large, (e.g., 2.0), there will be fewer steps, so detection will be faster, but we may miss objects whose size is between two tested scales. We usually don’t change the default value, which is 1.3.
* The second parameter known as the “minNeighbors” is changed based on the requirements. The higher the values of the second parameter(minNeighbors), less will be the number of false positives and less error will be in terms of false detection of faces. However, there is a chance of missing some unclear face traces as well.
* Next, we iterate through the detected faces and draw rectangles around them, so that the faces are clearly identified and the detected faces are displayed.

for (x, y, w, h) in face:

cv2.rectangle(image, (x, y), ((x + w), (y + h)), (255, 0, 0), 2)

cv2.putText(image, "Face", (x, y), cv2.FONT\_HERSHEY\_DUPLEX, 1.0, (255,0,0))

cv2.imshow('Video', real)

**Eye Detection in Face:**

* We first loaded the Haar cascade files of “frontal face” and “eye” using the Cascade Classifier method of the cv2 module using the following command:

face\_cas = cv2.CascadeClassifier('C:\\Users\\saide\\Documents\\Haar Cascade\\haarcascade\_frontalface\_default.xml')

eye\_cas = cv2.CascadeClassifier('C:\\Users\\saide\\Documents\\Haar Cascade\\haarcascade\_eye.xml')

* Next, we read the RGB image from webcam in which faces are to be detected using the imread function of the cv2 module.

vcap = cv2.VideoCapture(0)

vcap.set(3,1280)

vcap.set(4,720)

while(vcap.isOpened())

# Captures video\_capture frame by frame

ret, image = vcap.read()

* Next, we converted the RGB image into a grayscale image using the function cvtColor defined in the module cv2. The function cvtColor needs two mandatory parameters “src” –the image whose color space is to be changed, and “code” –the color space conversion code. We have used cv2.COLOR\_BGR2GRAY, as we need to convert the source BGR image into a grayscale

bw = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

* Next, we run the face detector on the grayscale image using the following line with “scaleFactor” value 1.3 (default value) and the minNeighbors value 5 in an outer “for loop”.

face = face\_cas.detectMultiScale(bw, 1.3, 5)

* In an inner “for” loop we detected the eyes for each “face” detected in the outer “for” loop. cv2.rectangle(roi\_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),4)
* The code of the two for loops together is as follows:

face = face\_cas.detectMultiScale(bw, 1.3, 5) #where 1.3 is the scaling factor, and 5 is the number of nearest neighbors

for (x, y, w, h) in face:

cv2.rectangle(image, (x, y), ((x + w), (y + h)), (255, 0, 0), 2)

cv2.putText(image, "Face", (x, y), cv2.FONT\_HERSHEY\_DUPLEX, 1.0, (255,0,0))

roi\_bw = bw[y:y + h, x:x + w]

roi\_col = image[y:y + h, x:x + w]

eyes = eye\_cas.detectMultiScale(roi\_bw)

for (ex, ey, ew, eh) in eyes:

cv2.rectangle(roi\_col,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)

cv2.putText(image, "Eyes", (ex+450, ey+250), cv2.FONT\_HERSHEY\_COMPLEX\_SMALL, 1.0, (0,255,0)) cv2.imshow('Video', real)

**Nose Detection in Face:**

* We first loaded the Haar cascade files of “frontal face” and “nose” using the Cascade Classifier method of the cv2 module using the following command:

face\_cas = cv2.CascadeClassifier('C:\\Users\\saide\\Documents\\Haar Cascade\\haarcascade\_frontalface\_default.xml')

nose\_cas=cv2.CascadeClassifier('C:\\Users\\saide\\Documents\\Haar Cascade\\haarcascade\_mcs\_nose.xml')

* Next, we read the RGB image from webcam in which faces are to be detected using the imread function of the cv2 module.

vcap = cv2.VideoCapture(0)

vcap.set(3,1280)

vcap.set(4,720)

while(vcap.isOpened())

# Captures video\_capture frame by frame

ret, image = vcap.read()

* Next, we converted the RGB image into a grayscale image using the function cvtColor defined in the module cv2. The function cvtColor needs two mandatory parameters “src” –the image whose color space is to be changed, and “code” –the color space conversion code. We have used cv2.COLOR\_BGR2GRAY, as we need to convert the source BGR image into a grayscale

bw = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

* Next, we run the face detector on the grayscale image using the following line with “scaleFactor” value 1.3 (default value) and the minNeighbors value 5 in an outer “for loop”.

face = face\_cas.detectMultiScale(bw, 1.3, 5)

•In an inner “for” loop we detected the nose for each “face” detected in the outer “for” loop.

cv2.rectangle(roi\_col,(nx,ny),(nx+nw,ny+nh),(255,255,0),2)

•The code of the two “for loops” together is as follows:

for (x, y, w, h) in face:

cv2.rectangle(image, (x, y), ((x + w), (y + h)), (255, 0, 0), 2)

cv2.putText(image, "Face", (x, y), cv2.FONT\_HERSHEY\_DUPLEX, 1.0, (255,0,0))

roi\_bw = bw[y:y + h, x:x + w]

roi\_col = image[y:y + h, x:x + w]

for (nx, ny, nw, nh) in nose:

cv2.rectangle(roi\_col,(nx,ny),(nx+nw,ny+nh),(255,255,0),2)

cv2.putText(image, "Nose", (nx+430, ny+150), cv2.FONT\_ITALIC, 1.0, (255,255,0))

cv2.imshow('Video', real)

**Smile Detection in Face:**

* We first loaded the Haar cascade files of “frontal face” and “smile” using the Cascade Classifier method of the cv2 module using the following command:

face\_cas = cv2.CascadeClassifier('C:\\Users\\saide\\Documents\\Haar Cascade\\haarcascade\_frontalface\_default.xml')

smile\_cas=cv2.CascadeClassifier('C:\\Users\\saide\\Documents\\Haar Cascade\\haarcascade\_smile.xml')

* Next, we read the RGB image from webcam in which faces are to be detected using the imread function of the cv2 module.

vcap = cv2.VideoCapture(0)

vcap.set(3,1280)

vcap.set(4,720)

while(vcap.isOpened())

# Captures video\_capture frame by frame

ret, image = vcap.read()

* Next, we converted the RGB image into a grayscale image using the function cvtColor defined in the module cv2. The function cvtColor needs two mandatory parameters “src” –the image whose color space is to be changed, and “code” –the color space conversion code. We have used cv2.COLOR\_BGR2GRAY, as we need to convert the source BGR image into a grayscale

bw = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

* Next, we run the face detector on the grayscale image using the following line with “scaleFactor” value 1.3 (default value) and the minNeighbors value 5 in an outer “for loop”.

face = face\_cas.detectMultiScale(bw, 1.3, 5)

•In an inner “for” loop we detected the nose for each “face” detected in the outer “for” loop.

cv2.rectangle(roi\_col, (sx, sy), ((sx + sw), (sy + sh)), (0, 0, 255), 2)

•The code of the two “for loops” together is as follows:

for (x, y, w, h) in face:

cv2.rectangle(image, (x, y), ((x + w), (y + h)), (255, 0, 0), 2)

cv2.putText(image, "Face", (x, y), cv2.FONT\_HERSHEY\_DUPLEX, 1.0, (255,0,0))

roi\_bw = bw[y:y + h, x:x + w]

roi\_col = image[y:y + h, x:x + w]

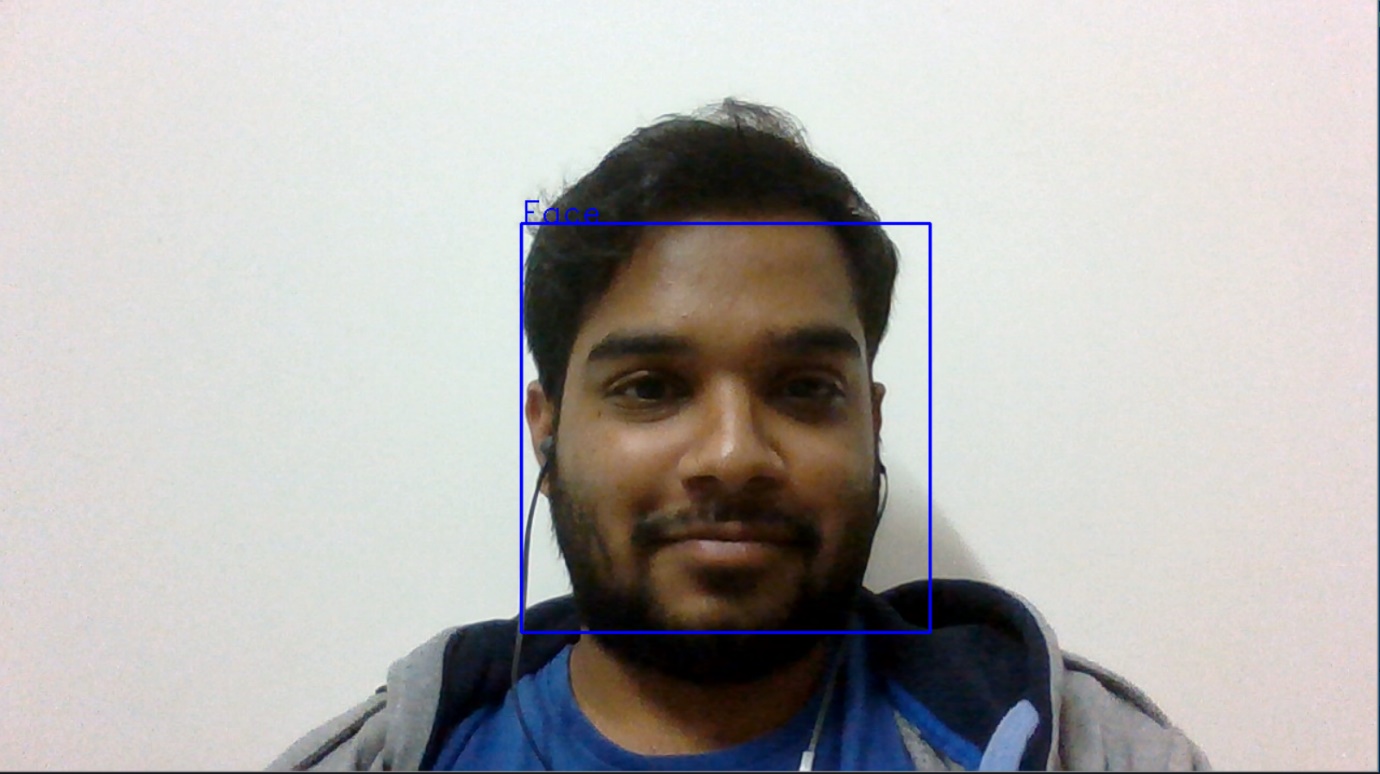
for (sx, sy, sw, sh) in smile:

cv2.rectangle(roi\_col, (sx, sy), ((sx + sw), (sy + sh)), (0, 0, 255), 2)

cv2.putText(image, "Smile", (sx+500,sy+350), cv2.FONT\_HERSHEY\_PLAIN, 1.0, (0,0,255))

cv2.imshow('Video', real)

**Results: Face Detection**

****

**Results: Eye Detection**

A picture containing wall, person, person, indoor

Description automatically generated

**Results: Nose Detection**

A picture containing person, wall, person, posing

Description automatically generated

**Results: Smile DetectionA picture containing person, person

Description automatically generated**

**Conclusion:**

In this work, we have discussed the concept of face

detection using OpenCV in Python using Haar Cascade.

•We used the rich library set of OpenCV for a robust face

detection from a sample image. For training the model with

the feature set of a face, we used the “Haar frontal face”

XML file.

•We later extended our model to detect eyes and nose in

the same input image. We used “haar\_eyes” and

“haar\_mcs\_nose” XML files for this purpose.

•Our model could successfully detect all faces, eyes, and

noses in the input image with 100% detection accuracy

and in real-time detection speed.

* In this work, we have discussed the concept of face detection using OpenCV in Python using Haar Cascade.
* We used the rich library set of OpenCV for a robust face detection from a sample image. For training the model with the feature set of a face, we used the “Haar frontal face” XML file.
* We later extended our model to detect eyes, nose and smile in the same input image. We used “haarcascade\_eye” “haar\_mcs\_nose” and haarcascade\_smil XML files for this purpose.
* Our model could successfully detect all faces, eyes, and noses in the input image with 100% detection accuracy and in real-time detection speed.