This project is about smart selfie using python programming. It detects a smile on the face and accordingly takes a selfie-and saves it into your drive. The full procedure is been described step by step.

**Libraries used in this code are :**

1. OpenCV
2. Numpy
3. DLIB ( Toolkit containing Machine Learning algorithms )

**The following pipeline would be followed :**

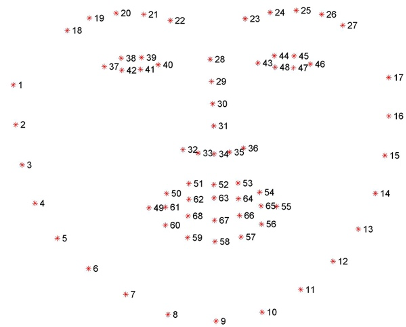
1. Recognize the face in the video
2. Recognize facial landmarks
3. Calculate the smile parameter
4. If sp > threshold, Take a selfie

Firstly, Let's define a function which finds the human face

detector = dlib.get\_frontal\_face\_detector()

As we know dlib package contains facial recognition features, we%u2019ll use them in our code.

Facial landmark detection: **68 point facial location**%u2013 jawline, eyebrow, eyes, nose, mouth



After finding human face, shape predictor will identify object to point different location i.e., eye, lips, outer face

predictor = dlib.shape\_predictor('%u2026/shape\_predictor.dat')

Secondly, Define a variable which detects human faces,

*Read the footage in grayscale*

rects = detector(gray,1)

As detector will predict the face using dlib package, we%u2019ll have to convert the acquired location into bounding box as understood by OpenCV with format (x, y , w, h)

def rect\_to\_bb(rect) :

    x = rect.left()

    y = rect.top()

    w = rect.right()-x

    h = rect.bottom()-y

    return (x,y,w,h)

*As this function will return facial co-ordinates, its better to convert them into numpy array format for better understanding and for further ease of access.*

def shape\_to\_np(shape,dtype='int') :

# Initialize the list of (x,y) co-ordinates

    coords = np.zeros((68,2),dtype=dtype)

# Run a for loop over 68 facial features

    for i in range(0,68) :

        coords[i] = (shape.part(i).x , shape.part(i).y)

    return coords

Now lets loop over face , to draw rectangle around it by using the coordinates acquired:

for i in range(0,len(rects)) :

        (x,y,w,h) = rect\_to\_bb(rects[i])

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

So till now, code is somewhat like this :

rects = detector(gray,1)

for i in range(0,len(rects)) :

        (x,y,w,h) = rect\_to\_bb(rects[i])

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

# Determine the facial landmarks for the face region

        shape = predictor(gray,rects[i])

# Convert the facial landamark into numpy array

        shape = shape\_to\_np(shape)

As of now, we have only detected the face. But we also want to show the facial features. So let's call a variable that will predict the features required.

# Determine the facial landmarks for the face region

        shape = predictor(gray,rects[i])

# Convert the facial landamark into numpy array

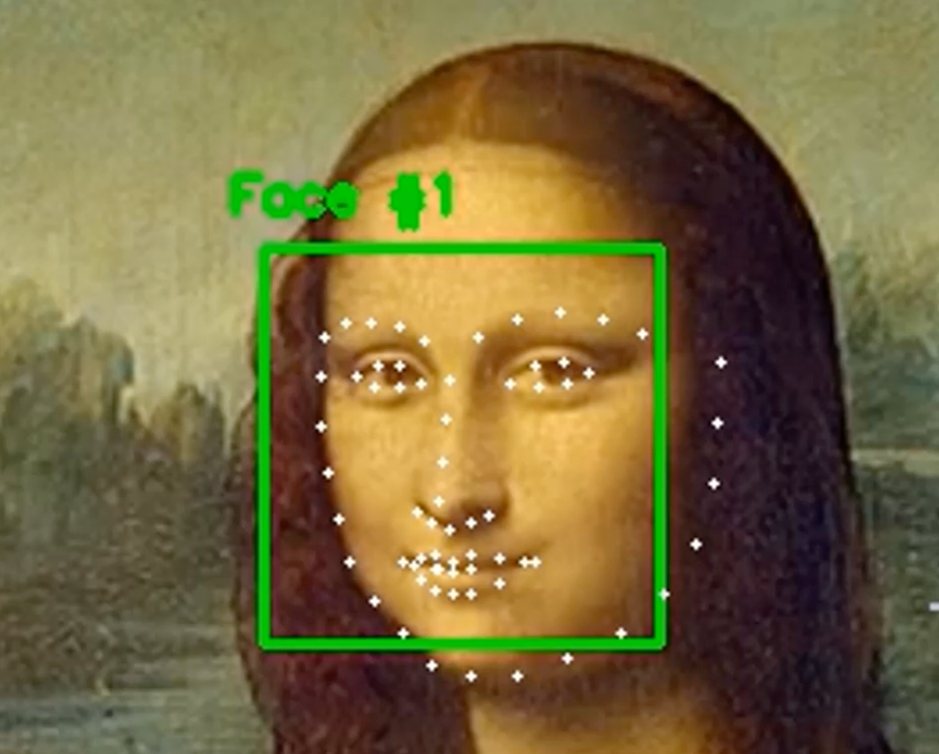
        shape = shape\_to\_np(shape)

*This 'shape' will return coordinates*

Once the (x,y) coordinates has been acquired, lets draw them on the image

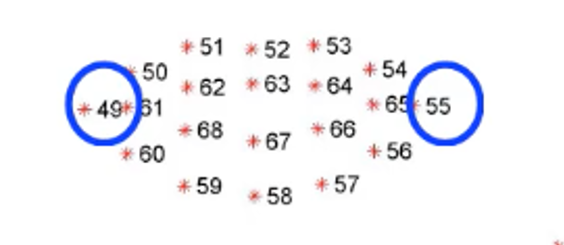
for (x,y) in shape:

            cv2.circle(frame , (x,y), 1 , (255,0,0),-1)



As we want only the mouth features detected, let's define the mouth region.

Indices for coordinates corresponding to mouth: 48 to 67



Now let's define the start and end of mouth by NumPy array

(mstart,mend) = (48,67)

Define a subset NumPy array to only identify only mouth features

mouth = shape[mstart:]

Now instead of showing all the facial features, lets only show mouth features :

for (x,y) in shape:

            cv2.circle(frame , (x,y), 1 , (255,0,0),-1)

Replace shape with mouth :

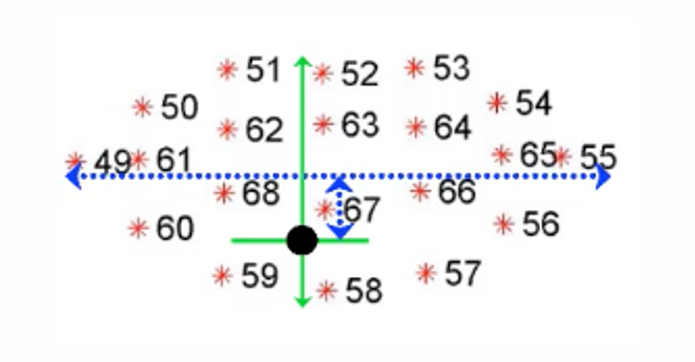
for (x,y) in shape:

            cv2.circle(frame , (x,y), 1 , (255,0,0),-1)

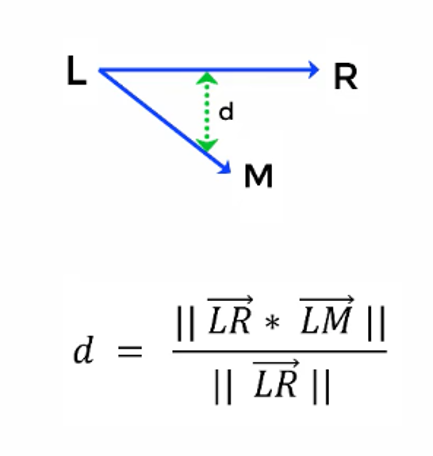
We have defined the mouth%u2019s start and endpoint as mentioned earlier,

**Let's define a parameter which calculates the average of points in the center of the mouth.**

mid = (shape[51]+shape[62]+shape[66]+shape[57])/4



Let us calculate the perpendicular distance between midpoint and line joining the ends by simple vector formula :



We have inbuilt NumPy functions already available for the above formula

dist = np.abs(np.cross(right-left,left-mid)) / np.linalg.norm(right-left)

So our function looks something like this, which returns the distance

def smile(shape) :

    left = shape[48]

    right = shape[54]

    mid = (shape[51]+shape[62]+shape[66]+shape[57])/4

    dist = np.abs(np.cross(right-left,left-mid)) / np.linalg.norm(right-left)

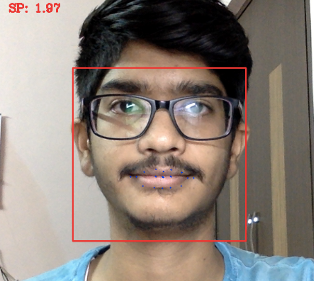
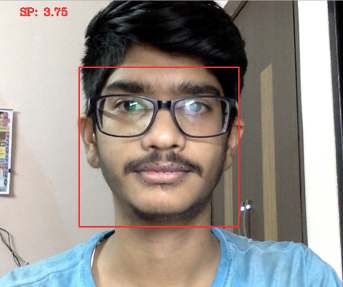
    return dist

Define a smile parameter :

smile\_param = smile(shape)

cv2.putText(frame,"SP: {:.2f}".format(smile\_param),(300,30),cv2.FONT\_HERSHEY\_COMPLEX,0.7,(0,0,255),2)

*Then run the code till now to see output like below.*



                        \*\*\*\*\* As we smile the smile parameter changes accordingly \*\*\*\*\*

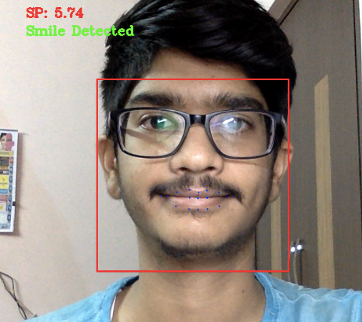
Lets set a constant value such that if that value is exceeded, we can understand that the person is smiling.

smile\_cont = 5

Now lets write a condition :

if smile\_param > smile\_cont :

            cv2.putText(frame,"Smile Detected",(300,60),cv2.FONT\_HERSHEY\_COMPLEX,0.7,(0,255,0),2)



                            ~ We are almost done, one last thing is to save the selfie

The question is how to save the image?

So what we'll do is introduce a constant and increment its value each time the condition (smile\_param > smile\_const ) is satisfied.

We%u2019ll do so by introducing a constant

counter = 0

Once smile\_parameter exceeds the smile\_constant, we increment counter

if smile\_param > smile\_cont :

            cv2.putText(frame,"Smile Detected",(300,60),cv2.FONT\_HERSHEY\_COMPLEX,0.7,(0,255,0),2)

            counter = counter +1

Now we condition such that, if counter exceeds 15 , save thae corresponding frame

if counter > 15 :

                ret,frame1 = cam.read()

                img\_name = "../smart\_selfie\_.png"

                cv2.imwrite(img\_name,frame1)

                print('taken !')

                counter = 0

The full code is here as follows...

import cv2

import numpy as np

import dlib

detector = dlib.get\_frontal\_face\_detector()

predictor = dlib.shape\_predictor('../shape\_predictor.dat')

(mstart,mend) = (48,67)

smile\_cont = 5

counter = 0

selfie\_no = 0

def rect\_to\_bb(rect) :

x = rect.left()

y = rect.top()

w = rect.right()-x

h = rect.bottom()-y

return (x,y,w,h)

def shape\_to\_np(shape,dtype='int') :

coords = np.zeros((68,2),dtype=dtype)

for i in range(0,68) :

coords[i] = (shape.part(i).x , shape.part(i).y)

return coords

def smile(shape) :

left = shape[48]

right = shape[54]

mid = (shape[51]+shape[62]+shape[66]+shape[57])/4

dist = np.abs(np.cross(right-left,left-mid)) / np.linalg.norm(right-left)

return dist

cam = cv2.VideoCapture(0)

while(cam.isOpened()) :

ret,frame = cam.read()

frame = cv2.flip(frame,1)

gray = cv2.cvtColor(frame,cv2.COLOR\_BGR2GRAY)

rects = detector(gray,1)

for i in range(0,len(rects)) :

(x,y,w,h) = rect\_to\_bb(rects[i])

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

shape = predictor(gray,rects[i])

shape = shape\_to\_np(shape)

mouth = shape[mstart:]

for (x,y) in mouth:

cv2.circle(frame , (x,y), 1 , (255,0,0),-1)

smile\_param = smile(shape)

cv2.putText(frame,"SP: {:.2f}".format(smile\_param),(300,30),cv2.FONT\_HERSHEY\_COMPLEX,0.7,(0,0,255),2)

if smile\_param > smile\_cont :

cv2.putText(frame,"Smile Detected",(300,60),cv2.FONT\_HERSHEY\_COMPLEX,0.7,(0,255,0),2)

counter = counter +1

if counter > 15 :

selfie\_no = selfie\_no+1

ret,frame1 = cam.read()

img\_name = "../smart\_selfie\_{}.png".format(selfie\_no)

cv2.imwrite(img\_name,frame1)

print('taken !')

counter = 0

else :

counter = 0

cv2.imshow('frames',frame)

key = cv2.waitKey(10)

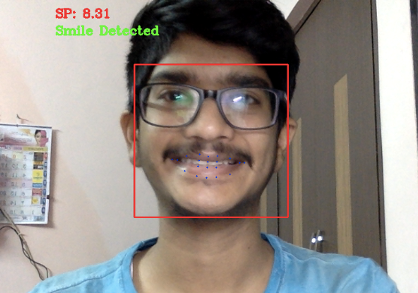
if key == 27 :

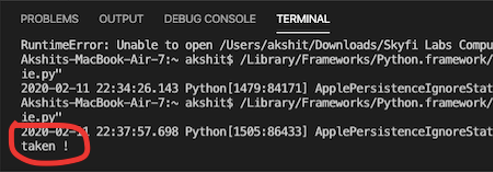
break

cam.release()

cv2.destroyAllWindows()

Few output samples are shown below :





Thanks for following through!

\*\*\*\*\* Written by Akshit Tayade \*\*\*\*\*\*