1) Get media stream instance to capture images from video:

var stream, imageCapture;

function getMediaStream()

{

window.navigator.mediaDevices.getUserMedia({video: true})

.then(function(mediaStream)

{

stream = mediaStream;

let mediaStreamTrack = mediaStream.getVideoTracks()[0];

imageCapture = new ImageCapture(mediaStreamTrack);

console.log(imageCapture);

})

.catch(error => {

console.error('error:', error);

});

}

2) Capture image after 1 second time interval in loop

setInterval(function(){

*/\* and when you want to capture an image \*/*

onTakePhoto();

}, 2000)

function onTakePhoto() {

imageCapture.takePhoto()

.then(blob => createImageBitmap(blob))

.then(imageBitmap => {

const canvas = document.querySelector('#takePhotoCanvas');

drawCanvas(canvas, imageBitmap);

drawImage(canvas);

})

.catch(error => {

console.error('error:', error);

});

}

Utility method to create image from canvas:

function drawCanvas(canvas, img) {

canvas.width = getComputedStyle(canvas).width.split('px')[0];

canvas.height = getComputedStyle(canvas).height.split('px')[0];

let ratio = Math.min(canvas.width / img.width, canvas.height / img.height);

let x = (canvas.width - img.width \* ratio) / 2;

let y = (canvas.height - img.height \* ratio) / 2;

canvas.getContext('2d').clearRect(0, 0, canvas.width, canvas.height);

canvas.getContext('2d').drawImage(img, 0, 0, img.width, img.height,

x, y, img.width \* ratio, img.height \* ratio);

}

function drawImage(canvas) {

const photo = document.querySelector('#photo');

var data = canvas.toDataURL('image/png');

photo.setAttribute('src', data);

*// upload file*

upload\_photo\_2\_server(data)

}

3) Upload image to server:

function upload\_photo\_2\_server(photo)

{

var fd = new FormData();

var file\_name = 'temp.png'

fd.append('photo', photo);

$.ajax({

url : '/capture/save\_photo',

data : fd,

contentType : false,

processData : false,

type : 'POST',

success : function(data, textStatus, jqXHR){

*// Callback code*

console.log("photo saved Successfully.")

}

});

}

Python code:

4) Save image on server:

def save\_photo(request):

""" save captured photo"""

user\_id = request.POST.get('user\_id')

photo\_data = request.POST.get('photo')

format, img\_str = photo\_data.split(';base64,')

ext = format.split('/')[-1]

data = ContentFile(base64.b64decode(img\_str), name='img.' + ext)

try:

# save info in database

Photo.objects.create(

user\_id=user\_id,

photo=data

)

return JsonResponse({'success':1})

except Exception as e:

print(e)

return JsonResponse({'success': 0})

5) method to Train model: (Called from training view)

def train\_model(

    embeddings\_pickle\_file="output/embeddings.pickle",

    recognizer\_file="output/recognizer.pickle",

    label\_encoder="output/le.pickle"

    ):

*# load the face embeddings*

     print("[INFO] loading face embeddings...")

     data = pickle.loads(open(embeddings\_pickle\_file, "rb").read())

*# encode the labels*

     print("[INFO] encoding labels...")

     le = LabelEncoder()

     labels = le.fit\_transform(data["names"])

*# train the model used to accept the 128-d embeddings of the face and*

*# then produce the actual face recognition*

     print("[INFO] training model...")

     recognizer = SVC(C=1.0, kernel="linear", probability=True)

     recognizer.fit(data["embeddings"], labels)

*# write the actual face recognition model to disk*

     f = open(recognizer\_file, "wb")

     f.write(pickle.dumps(recognizer))

     f.close()

*# write the label encoder to disk*

     f = open(label\_encoder, "wb")

     f.write(pickle.dumps(le))

    f.close()

6) Real time Recognize :

def face\_recognize(

    face\_detection\_model\_folder="face\_detection\_model",

    embedding\_model="openface\_nn4.small2.v1.t7",

    recognizer\_file="output/recognizer.pickle",

    label\_encoder="output/le.pickle",

    image\_file="images/n.jpg",

    min\_confidence=0.5,

    res\_file\_name\_with\_path="result.png"

):

*# load our serialized face detector from disk*

    print("[INFO] loading face detector...")

    protoPath = os.path.join(face\_detection\_model\_folder, "deploy.prototxt")

    modelPath = os.path.join(

        face\_detection\_model\_folder,

        "res10\_300x300\_ssd\_iter\_140000.caffemodel"

    )

    detector = cv2.dnn.readNetFromCaffe(protoPath, modelPath)

*# load our serialized face embedding model from disk*

    print("[INFO] loading face recognizer...")

    embedder = cv2.dnn.readNetFromTorch(embedding\_model)

*# load the actual face recognition model along with the label encoder*

    recognizer = pickle.loads(open(recognizer\_file, "rb").read())

    le = pickle.loads(open(label\_encoder, "rb").read())

*# load the image, resize it to have a width of 600 pixels (while*

*# maintaining the aspect ratio), and then grab the image dimensions*

    image = cv2.imread(image\_file)

    image = imutils.resize(image, width=600)

    (h, w) = image.shape[:2]

*# construct a blob from the image*

    imageBlob = cv2.dnn.blobFromImage(

        cv2.resize(image, (300, 300)), 1.0, (300, 300),

        (104.0, 177.0, 123.0), swapRB=False, crop=False)

*# apply OpenCV's deep learning-based face detector to localize*

*# faces in the input image*

    detector.setInput(imageBlob)

    detections = detector.forward()

*# loop over the detections*

    for i in range(0, detections.shape[2]):

*# extract the confidence (i.e., probability) associated with the*

*# prediction*

        confidence = detections[0, 0, i, 2]

*# filter out weak detections*

        if confidence > min\_confidence:

*# compute the (x, y)-coordinates of the bounding box for the*

*# face*

            box = detections[0, 0, i, 3:7] \* np.array([w, h, w, h])

            (startX, startY, endX, endY) = box.astype("int")

*# extract the face ROI*

            face = image[startY:endY, startX:endX]

            (fH, fW) = face.shape[:2]

*# ensure the face width and height are sufficiently large*

            if fW < 20 or fH < 20:

                continue

*# construct a blob for the face ROI, then pass the blob*

*# through our face embedding model to obtain the 128-d*

*# quantification of the face*

            faceBlob = cv2.dnn.blobFromImage(face, 1.0 / 255, (96, 96),

                (0, 0, 0), swapRB=True, crop=False)

            embedder.setInput(faceBlob)

            vec = embedder.forward()

*# perform classification to recognize the face*

            preds = recognizer.predict\_proba(vec)[0]

            j = np.argmax(preds)

            proba = preds[j]

            name = le.classes\_[j]

*# draw the bounding box of the face along with the associated*

*# probability*

            text = "{}: {:.2f}%".format(name, proba \* 100)

            y = startY - 10 if startY - 10 > 10 else startY + 10

            cv2.rectangle(image, (startX, startY), (endX, endY),

                (0, 0, 255), 2)

            cv2.putText(image, text, (startX, y),

                cv2.FONT\_HERSHEY\_SIMPLEX, 0.45, (0, 0, 255), 2)

return image

6) Display recognized image:

function recognize\_photo(data)

{

var fd = new FormData();

*// Attach file*

var file\_name = 'tmp.png'

fd.append('photo', data);

$.ajax({

url : '/train\_recognize/recognize\_photo',

data : fd,

contentType : false,

processData : false,

type : 'POST',

success : function(data, textStatus, jqXHR){

*// Callback code*

console.log("Photo recognized .....")

*// set recognized result*

const photo = document.querySelector('#result\_photo');

photo.setAttribute('src', data.img);

}

});

}