# SPORTS CELEBRITY FACE RECOGNITION AND EMOTION DETECTION

## A PROJECT REPORT

In partial fulfillment of the requirements for the award of the degree

in

**BACHELORS OF TECHNOLOGY IN COMPUTER SCIENCE ENGINEERING**

Under the guidance of

**Mr. Tanmoy Bera**



**MURSHIDABAD COLLEGE OF ENGINEERING AND TECHNOLOGY (MCET), MURSHIDABAD, WEST BENGAL.**

**MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY, WEST BENGAL.**

1. Title of the project: **SPORTS CELEBRITY FACE RECOGNITION AND EMOTION DETECTION**

2. Project Members: **PRATTAY SAHA**
**ABID HOSSAIN**
**USUF SK**
**VASHKAR MANDAL**

3. Name of the Guide: **MR. TANMOY BERA**

**Project Version Control History:**

| Version | Members | Description of Version | Date of Completion |
|---|---|---|---|
| Final | Prattay Saha<br>Abid Hossain<br>Usuf Sk<br>Vashkar Mandal | Project Report | 18th Nov, 2022 |

Signature Of Team Members:

Date:-

Signature of the Approver/ Proposal Evaluator:

Date:

# DECLARATION

We hereby declare that the project work being presented in the project proposal entitled "**SPORTS CELEBRITY FACE RECOGNITION**" in partial fulfillment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE ENGINEERING**, is an authentic work carried out under the guidance of ***MR. TANMOY BERA***. The matter embodied in this project work has not been submitted elsewhere for the award of any degree of our knowledge and belief.

Date:

Name of the Student: **Prattay Saha, Abid Hossain, Usuf Sk, Vashkar Mandal.**

Signature of the students:

# ACKNOWLEDGEMENT

Success of any project depends largely on the encouragement and guidelines of many others. I take this sincere opportunity to express my gratitude to the people who have been instrumental in the successful completion of this project work.

I would like to show our greatest appreciation to ***Mr. TANMOY BERA***, Professor at Murshidabad College of Engineering and Technology, Murshidabad, West Bengal. I always feel motivated and encouraged every time by his valuable advice and constant inspiration; without his encouragement and guidance this project would not have materialized.

Words are inadequate in offering our thanks to the other trainees, project assistants and other members at Murshidabad College of Engineering and Technology for their encouragement and cooperation in carrying out this project work. The guidance and support received from all the members and who are contributing to this project, was vital for the success of this project.

# ABSTRACT

Importance of face recognition systems has sped up in the last few decades. A face recognition system is one of biometric information processing. Applicability is easier and the working range is larger than other biometric information processing, i.e.; fingerprint, iris scanning, signature, etc. A face recognition system is designed, implemented and tested in this study. The system utilizes a combination of techniques in three topics; face detection, recognition and emotion detection.The face detection is performed on five sport celebrity.process utilized in the system are face detection, facial feature extraction, emotion detection. Then a face detection model Face Detection(FP16) which is a floating point 16 version of the original caffe implementation. The system is tested with 10 people. The tested system has acceptable performance to recognize faces. System also capable of detecting and recognizing multiple faces in images.

# CONTENT

**CHAPTER 1: Introduction**

1.1. Why Computer Vision is Hard
1.2. Face Recognition Process
1.3. Library Used
1.4. Face Collection
1.5. Face Detection
1.6. Facial Feature Extractions

**CHAPTER 2: Training and Testing**

2.1. Face Data Pre-Processing
2.2 Train Machine Learning Model for Face
2.3. Emotion Data preprocessing
2.4 Train Machine Learning Model for Emotion
2.5. Pipeline all Models

**CHAPTER 3: User Interface**

1.7. Used Languages
1.8. UI design

**CHAPTER 4: Codes**

**4.1.** Face Detection
4.2. Facial Landmark Detection
4.3. Face Data Preprocessing
4.4. Train Machine Learning Model of Faces
4.5. Emotion Data preprocessing
4.6. Train Machine Learning Model of Emotions
4.7. Pipeline all Models
4.8. UI

**CHAPTER 5: CONCLUSION**

**CHAPTER 6: CURRENT AND FUTURE WORK**

**CHAPTER 7: BIBLIOGRAPHY**

# INTRODUCTION

Face recognition has been an active area of research in the past several decades. Initially a branch of artificial intelligence to enable robots with visual perception, it is now part of a more general and larger discipline of computer vision. Computer vision applications can process images from a wide range of the electromagnetic spectrum. X- rays are used in medical technology to create images of the human body without surgery. Gamma rays and radio waves in magnetic resonance imaging (MRI) capture images of thin slices of the human body useful for diagnostic and treatment of diseases. X-rays in the automotive industry are used for inspection of material that is hard to detect by the naked eye, such as casting of wheel rims for fractures, cracks, bubble-shaped voids, and defects in lack of fusion. In the food industry, X-rays and gamma rays are used for inspection, safety and quality of their products. Examples include detection of foreign objects in packaged food like fish bone in fish, contaminants in food products such as insect infestation in citrus fruits, and quality inspection for split-pits or water content distribution . Figure 1 shows the electromagnetic spectrum.

1. In contrast to computer vision, face recognition applications are confined to the narrow band of visible light where surveillance and biometrics authentication can be performed. Biometrics is the term used to describe human characteristics metrics such as iris, fingerprint or hand geometry. These metrics are used for identification and access control of individuals that are under surveillance . Face is becoming the preferred metric over current biometrics simply because it is a natural assertion of identity, and its non-intrusive nature provides more convenience and ease of verification. For example, in a fingerprinting system, the subject

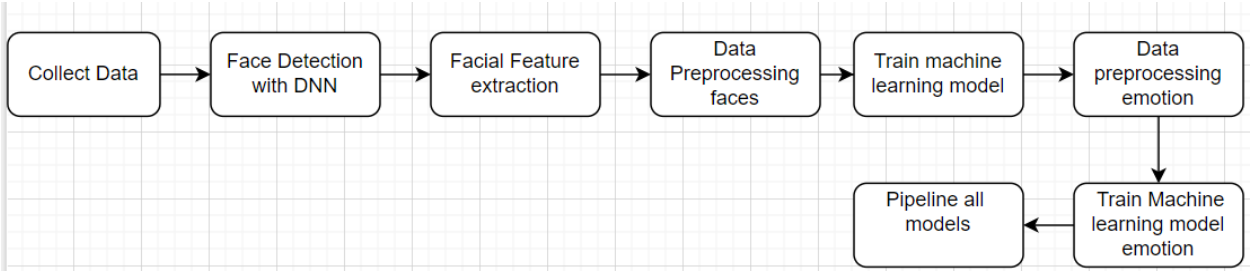
is required to interact with the system by placing a finger under a fingerprint reader, and the results must be verified by an expert. In contrast, using the subject's face as a metric requires no intervention, and the results can be verified by a non-expert.

## Why is Computer Vision Hard?

All images must be first captured by a camera and then be given to a computer vision application for further processing. Compared to the \human visual system, the camera is the eye, and the processing software is the brain of the application. To acquire the image, the camera uses light reflecting off an object and transmits the light intensity to its built-in sensors. The sensors then convert each of their cell intensities to a value in the range of 0-255, where a grid of numbers in this range becomes the final representation of the captured image. Note that light is a form of electromagnetic energy spanning a frequency range known as the visual spectrum. Also, sensors are unique to digital cameras as older analog cameras captured images on film.

# Face Recognition Process

Face recognition is the process of labeling a face as recognized or unrecognized. The process has a life cycle based on a pipeline that goes through collection, detection, pre-processing, and a recognition stage. In the collection step, images are captured and stored for training and recognition. In the detection phase, regions of a face within an image are identified and their location is recorded. The pre-processing stage modifies the image by removing unwanted features such as shadow or excessive illumination. Recognition, the final stage of the pipeline, identifies the face as recognized or not recognized.



# Library Used

**Library used:**

1. numpy
2. pandas
3. matplotlib
4. jupyter
5. pillow
6. opencv-python
7. Sklearn
8. Dlib

# Face Collection

## Procedure for Data collection:

Before a recognition system can identify a face, it must first be trained on a collection of images, known as the training set. The set enables comparison of its contents with a new image to determine if the difference is small enough for a positive identification. For a successful recognition, the set must be robust, meaning it must contain a variety of images such as facial images (positive samples) as well as non-facial images (negative samples) such as cars, trees, etc. Furthermore, the set must contain a variation of facial images, where the subject is looking up or down, with different facial 6 expressions and lighting conditions. It is important to have variety in the set rather than just a large number of images with little or no variation in them

Steps:

1. Download
2. Converting

- **Downloads :**
    - fatkun extension
        - Open web store ; add to the extension
        - Search for the desire subject in google search engine ;
        - Open the extension ; select fatkun ; Download [current tab]
        - Automatically it download all the images in the current tab
- **Converting :**
    - command prompt
        - Run cmd
        - Select the disk where the downloaded files are save
        - Run the code “ ren *png *jpg”
    - AVS image convertor
        - Automated software to convert all the image in desire format
        - Run the software ;select the images ; then select “convert it change all the images in same format

**Procedure for manual data cleaning** :

1. Manually data cleaning
2. Folder

- **Manually detecting**

    - Delete
        - Check all the images manually and delete all the multi count person images
    - Cropping
        - High quality images can be considers to crop and remove the unwanted person or background
    - Folder
        - Creating folder for training the data

# Face Detection

Face detection is the process of locating a face in an image without identification.

- Single Shot MultiBox Detector(SSD) framework using a ResNet-10 network
- openCV DNN module:
    1. Caffe
    2. Tensorflow
    3. Torch
    4. Darknet

Here we use Caffe model:

1. res10_300x300_ssd_iter_140000_fp16.caffemodel
2. Deploy.prototxt.txt

Now we have to pass the picture to the model but it is not  a very straight forward method. At first we need to extract the blob from the image -

1. **extract blob**
   blob=cv2.dnn.blobFromImage(img,1,(300,300),(104,177,123),swapRB=False)
   {
   **img** = image we want to blob
   **1** = scale factor
   **(300,300)** = resize the image by 300X300 because the model only support this size
   **(104,177,123)** = mean of a BGR that is mean for B mean for G and mean for R. and blob will basically subtracted mean images.
   **swapRB=False** : openCV assumes that you have read an image in BGR format. If a image read in RGB format then set  swapRB = True otherwise swapRB = False
   }

2. **Set the blob as input** : net.setInput(blob)

3. **run the model :** detections = net.forward()

Basically ae are doing forward pass operation.i means we are not changing any weights to it what are the weights are there is directly use it.

4. **detections.shape**

Output: (1,1,200,7)

Here:

1,1 = for number of pictures

200 = model has detected the 200 faces. It does not mean that in that picture there are 200 faces.

7 = it gives a probability that the object that is detected is a face or not. In 7 we have seven numbers :

0: image number

1: Binary(0 or 1) where 0 indicate the box where the whatever it is detected in the box is not a face and 1 indicate whatever is detected in the box is a face

2: Confidence Score(0 to 1)

3: StartX

4: StartY

5: EndX

6: EndY

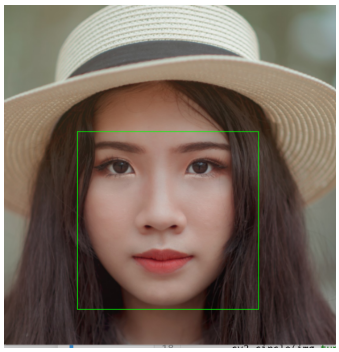
5. **Draw the rectangle:**

```
h,w = img.shape[:2]                    #height and width of the image
for i in range(0,detections.shape[2]):
   confidence = detections[0,0,i,2]
   if confidence >= 0.5
   box = detections[0,0,i,3:7]     # normalized bounding box values
```

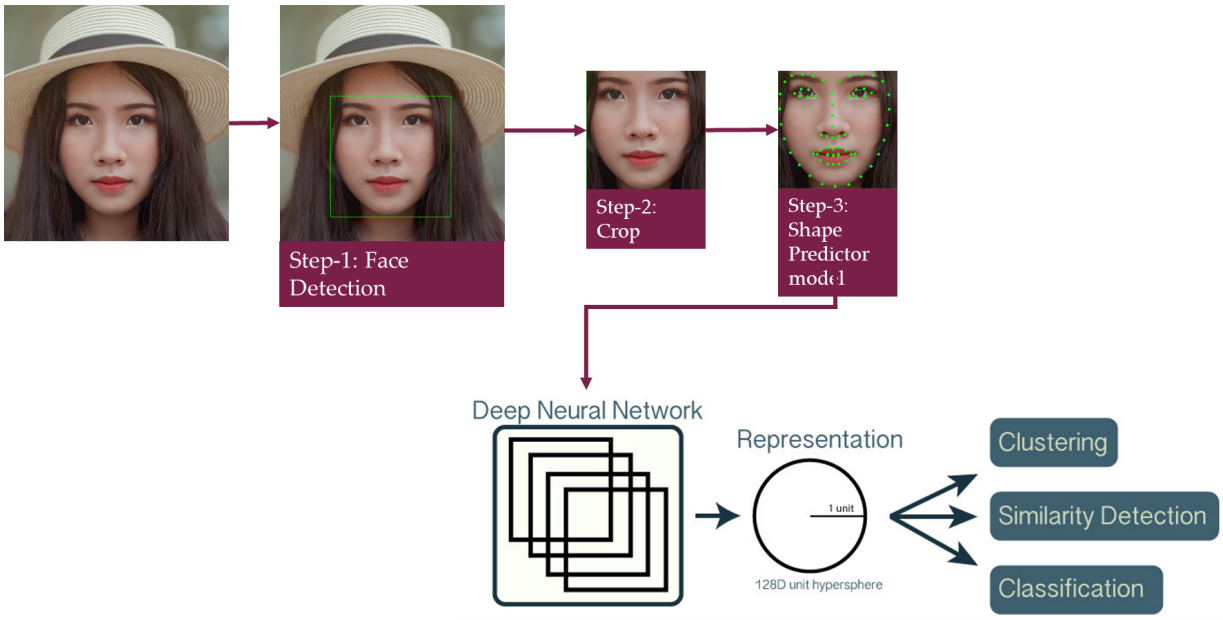
means you will Get the values which is normalize to width and height of an image

```
#to denormalize the value we need to multiply the normalized value
with width and height of the image respectively.

    box = box*np.array([w,h,w,h])
#opencv does not allow float values because image is an unsigned 8
bit integer that's why we change it to integer type
    box = box.astype(int)
    startx, starty , endx, endy = box
    # draw the rectangle
    cv2.rectangle(img,(startx,starty),(endx,endy),(0,255,0))
```

## Output:



# Facial features Extractions

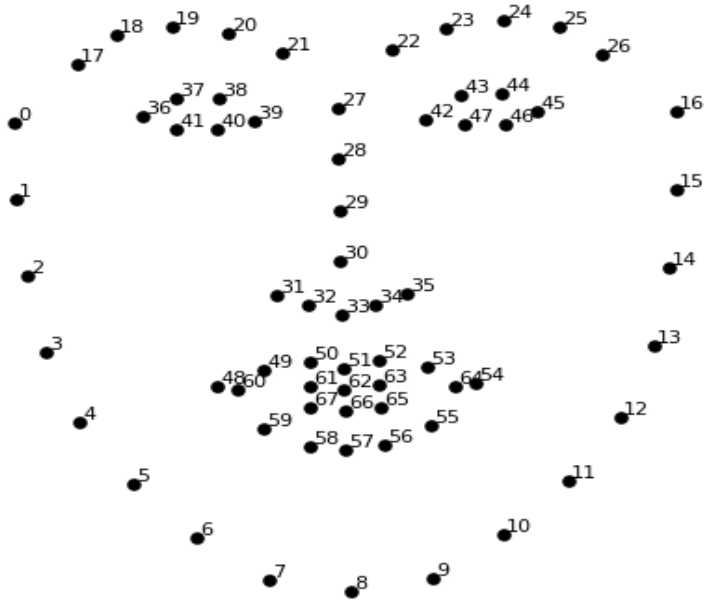



In the crop face we apply shape predictor model that will actually identify the landmarks in the face basically the keypoint like positions of the face like:

1. Eyes
2. Nose
3. Chin part
4. Mouth
5. Eyebrows

The shape predictor is passed to the Deep Neural Network and that is going to return you the 128 dimensional unit hypersphere which is the basic features of the face.

The model shape_predictor_68_face_landmarks.dat is trained on the iBUG-300 W dataset, where it contains images and their corresponding 68(x,y)-coordinates that map the face landmark points. In general, those landmark points belong to the nose, the eyes, the mouth, and the edge of a face.
Here is the visualization of the face landmark locations below:




- Load shape predictor model:

shape_predictor=dlib.shape_predictor('./models/shape_predictor_68_face_landmarks.dat')

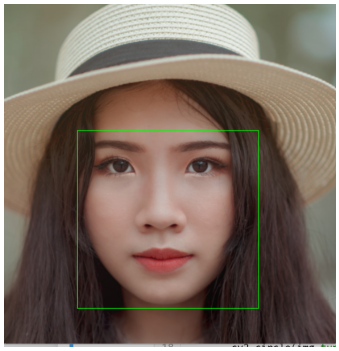
- Load shape descriptor model:

shape_descriptor=
dlib.face_recognition_model_v1('./models/dlib_face_recognition_resnet_model_v1.dat')

- to detect the face shape:

```
face_detector = dlib.get_frontal_face_detector()
faces = face_detector(img)
for box in faces:
    pt1 = box.left(), box.top()
    pt2 = box.right(),box.bottom()
    cv2.rectangle(img,pt1,pt2,(0,255,0))
```

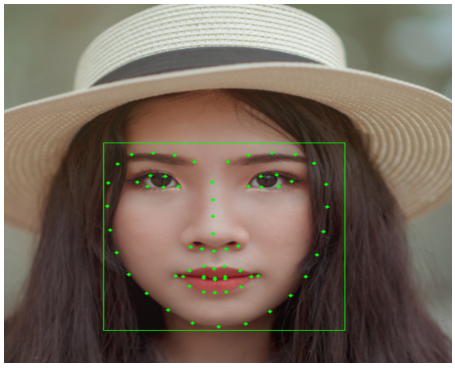
**Output:**



- Now use shape predictor model which will basically indicates the landmark of the face

```
face_shape = shape_predictor(img,box)
 face_shape_array = face_utils.shape_to_np(face_shape)
for points in face_shape_array:
      cv2.circle(img,tuple(points),2,(0,255,0),-2)
```

**Output:**



- Now use face descriptor which will basically calculate the face descriptor face_descriptors= shape_descriptor.compute_face_descriptor(img,face_shape)

The descriptor is used for the classification model or in order to train a machine learning model the input for this model is the face descriptor.

# Face Data Pre-Processing

- **Load the models using cv2 dnn:**
  face_detection_model='./models/res10_300x300_ssd_iter_140000.caffemodel'
  face_detection_proto = './models/deploy.prototxt.txt'
  face_descriptor = './models/openface.nn4.small2.v1.t7'
  # load models using cv2 dnn
  detector_model=cv2.dnn.readNetFromCaffe(face_detection_proto,face_detection_model)

- **Apply the images to the model and get the face descriptor from all the faces**

  Step1 :1. face detection
  2. Set the input
  {
  detector_model.setInput(img_blob)
  detections = detector_model.forward()
  }
  Step2 : 1. Feature extractions or Embeddings
  2. get the face descriptors
  To get the face descriptor we use our torch model which is descriptor_model
  {
  faceblob=
  cv2.dnn.blobFromImage(roi,1/255,(96,96),(0,0,0),swapRB=True,crop=True)
  descriptor_model.setInput(faceblob)
  vectors = descriptor_model.forward()
  }
  - ☐ roi = image
  - ☐ 1/255 = scale factor
  - ☐ (96,96) = size
  - ☐ (0,0,0) = rgb mean subtraction value

- **Save the data in a pickle format**
  pickle.dump(data,open('data_face_features.pickle',mode='wb'))

# Train Machine Learning Model for Face

## A. Data

- **Load data from pickle file**

  data = pickle.load(open('data_face_features.pickle',mode='rb'))

- **split the data into independent and dependent**

  X = np.array(data['data']) # independent variable

  y = np.array(data['label']) # dependent variable

- **split to train and test set**

  x_train,x_test,y_train,y_test =
  train_test_split(X,y,train_size=0.8,random_state=0)

## B. Train Machine Learning Model

- **Logistic Regression**

  model_logistic = LogisticRegression()

  model_logistic.fit(x_train,y_train)

- **Support Vector Machines**

  model_svc = SVC(probability=True)

  model_svc.fit(x_train,y_train)

- **Random Forest**

```
model_rf = RandomForestClassifier(n_estimators=10,)
model_rf.fit(x_train,y_train)
```

# Results:

| Classifier | Accuracy Train | Accuracy Test | F1 Score Train | F1 Score Test |
|---|---|---|---|---|
| **Logistic Regression** | 0.83 | 0.80 | 0.77 | 0.70 |
| **Support Vector Machines** | 0.90 | 0.83 | 0.86 | 0.73 |
| **Random Forest** | 0.99 | 0.76 | 0.99 | 0.61 |

- **Voting Classifier**

A voting Classifier is a machine learning model that trains on an ensemble or numerous models and predicts an output (class) based on their highest probability of chosen class as the output.

```
model_voting = VotingClassifier(estimators=[
   ('logistic',LogisticRegression()),
   ('svm',SVC(probability=True)),
   ('rf',RandomForestClassifier())
], voting='soft',weights=[2,3,1])
```

## C. Parameter Tuning

GridSearchCV is the process of performing hyperparameter tuning in order to determine the optimal values for a given model.

```
model_grid = GridSearchCV(model_voting,
                          param_grid={
                             'svm__C':[3,5,7,10],
                             'svm__gamma':[0.1,0.3,0.5],
                             'rf__n_estimators':[5,10,20],
                             'rf__max_depth':[3,5,7],
                             'voting':['soft','hard']
                          },scoring='accuracy',cv=3,n_jobs=1,verbose=2)
```

**D. Fit in the model**

```
model_grid.fit(x_train,y_train)
```

**E. Save Model**

```
pickle.dump(model_best_estimator,open('./models/machinelearning_face_
person_identity.pkl',mode='wb'))
```

# Emotion Data Preprocessing

**A. Face Detection Model**

- **Load the models**

```
face_detection_model                                                    =
'./models/res10_300x300_ssd_iter_140000.caffemodel'
face_detection_proto = './models/deploy.prototxt.txt'
face_descriptor = './models/openface.nn4.small2.v1.t7'
# load models using cv2 dnn
```

detector_model=
cv2.dnn.readNetFromCaffe(face_detection_proto,face_detection_model)
descriptor_model = cv2.dnn.readNetFromTorch(face_descriptor)

- **Create Helper Function**
- **apply helper function to all images and get face descriptors**
- **Save the data**
  pickle.dump(data,open('data_face_features_emotion.pickle',mode='wb'))

# Train Machine learning model for emotion

**A. Data**

- **Load data from pickle file**
  data=pickle.load(open('data_face_features_emotion.pickle',mode='rb'))

- **split the data into independent variable and dependent variable**
  X = np.array(data['data']) # indendepent variable
  y = np.array(data['label']) # dependent variable

- **split the data into train and test**
  from sklearn.model_selection import train_test_split
  x_train,x_test,y_train,y_test = train_test_split(X,y,train_size=0.8,random_state=0

**B. Train Machine Learning**

- **Logistic Regression**

  model_logistic = LogisticRegression()

  model_logistic.fit(x_train,y_train)

- **Support Vector Machines**

```
model_svc = SVC(probability=True)
model_svc.fit(x_train,y_train)
```

- **Random Forest**

```
model_rf = RandomForestClassifier(n_estimators=10,)
model_rf.fit(x_train,y_train)
```

## Comparative Outcome:

| Classifier | Accuracy Train | Accuracy Test | F1 Score Train | F1 Score Test |
|---|---|---|---|---|
| Logistic Regression | 0.31 | 0.23 | 0.23 | 0.18 |
| Support Vector Machines | 0.42 | 0.25 | 0.32 | 0.17 |
| Random Forest | 0.99 | 0.47 | 0.99 | 0.48 |

- **Voting Classifier**

```
model_voting = VotingClassifier(estimators=[
    ('logistic',LogisticRegression()),
    ('svm',SVC(probability=True)),
    ('rf',RandomForestClassifier())
], voting='soft',weights=[2,3,1])
```

**C. Parameter tuning**

```
from sklearn.model_selection import GridSearchCV
model_grid = GridSearchCV(model_voting,
                    param_grid={
                       'svm__C':[3,5,7,10],
                       'svm__gamma':[0.1,0.3,0.5],
                       'rf__n_estimators':[5,10,20],
                       'rf__max_depth':[3,5,7],
                       'voting':['soft','hard']
                    },scoring='accuracy',cv=3,n_jobs=1,verbose=2)
```

**D. Fit in the model**

```
model_grid.fit(x_train,y_train)
```

**E. Save the model**

```
pickle.dump(model_best_estimator,open('./models/machinelearning_face_
emotion.pkl',mode='wb'))
```

# Pipeline all models

**A. Load the models**

```
# face detection
face_detector_model                                                    =
cv2.dnn.readNetFromCaffe('./models/deploy.prototxt.txt',

'./models/res10_300x300_ssd_iter_140000.caffemodel')
# feature extraction
face_feature_model                                                     =
cv2.dnn.readNetFromTorch('./models/openface.nn4.small2.v1.t7')

# face recognition
face_recognition_model=
pickle.load(open('./models/machinelearning_face_person_identity.pkl',
                                  mode='rb'))
```

```
# emotion recognition model
emotion_recognition_model=
pickle.load(open('./models/machinelearning_face_emotion.pkl',mode='rb'))
```

**B. Face detection**

```
img_blob=cv2.dnn.blobFromImage(img,1,(300,300),(104,177,123),swapR
B=False,crop=False)
face_detector_model.setInput(img_blob)
detections = face_detector_model.forward()
 if len(detections) > 0:
        for i , confidence in enumerate(detections[0,0,:,2]):
            if confidence > 0.5:
                box = detections[0,0,i,3:7]*np.array([w,h,w,h])
                startx,starty,endx,endy = box.astype(int)

                cv2.rectangle(image,(startx,starty),(endx,endy),(0,255,0))
```

**C. Feature extraction**

```
face_roi = img[starty:endy,startx:endx]
face_blob=cv2.dnn.blobFromImage(face_roi,1/255,(96,96),(0,0,0),swapRB
=True,crop=True)
face_feature_model.setInput(face_blob)
vectors = face_feature_model.forward()
```

**D. Predict name and emotion with machine learning**

```
face_name = face_recognition_model.predict(vectors)[0]
face_score = face_recognition_model.predict_proba(vectors).max()

 emotion_name = emotion_recognition_model.predict(vectors)[0]
                                        emotion_score                                    =
emotion_recognition_model.predict_proba(vectors).max()
```

**E. Put all these in a function**

# User Interface

## Used languages:

1. Python
2. Django
3. Html

## UI Design:

1. **Create virtual environment**
   To create a virtual environment:
   Code:
   Python -m venv (environment_name)

2. **To open virtual environment**
   .\(environment_name)\Scripts\activate

3. **Install packages:**
   a. numpy
   b. sklearn
   c. pillow
   d. django

4. **Start a project:**
   django-admin start project <project name>

5. **To create app**
   Cd <project name>
   django-admin startapp <app name>

6. **To run the app:**
   Python manage.py runserver

7. In the project file we have a file name **settings.py** which has all the settings. It is used to add templates, static files etc. and it is also used for deployment purposes.
8. In **settings.py add the app**
9. Here we use **sqlite3 database** which is django default database
10. **urls.py** is mainly used for url routine basically in order to add or any kind of extension of url we use.
11. In the app folder **views.py** is basically a bridge between the controllers and the templates.
It is actually the bridge between the html and the database.
12. **models.py** is basically used for create database, table, structure and all kind of authentication is used in models.py
13. In the project file create a new folder name **templates**. Here we add all our html.
14. Next add the path of the folder templates in settings.py
**15.** In templates folder create a file name **index.html**
**16.** In project dir create a new folder name **static**
**17.** Add static dir in setting.py
18. Static file is used to deal with photos.
19. In models.py create a database
Code:

```
from django.db import models
class FaceRecognition(models.Model):
    id = models.AutoField(primary_key=True)
    record_date = models.DateTimeField(auto_now_add=True)
    image = models.ImageField(upload_to = 'images/')

    def __str__(self):
        return str(self.record_date)
```

20. In admin.py register the model
Code:

```
from django.contrib import admin
from app.models import <project name>
# Register your models here.
```

admin.site.register(project name)

21. Now in the project folder create a new folder name media and in media create a new folder image.
22. Now add the path of the folder in settings.py
23. In settings.py add media root and media url
Code:
MEDIA_ROOT = MEDIA_DIR
MEDIA_URL = '/media/'
24. Migrate all the files
Code:
python manage.py migrate
25. Now run python manage.py makemigrations
27. Create a superuser
Code:
Python manage.py createsuperuser
28. In app folder create a new file name forms.py
29. Create forms
Code:

```
from django import forms
from app.models import FaceRecognition

class FaceRecognitionform(forms.ModelForm):

    class Meta:
        model = FaceRecognition
        fields =['image']
```

30. Now load the forms in views.py
31. Connect forms with index.html file through views.py
32. In the views we are validating the request that it is a post or get. Once it is a post means either an image or a file is uploaded. Once it is uploaded we valid the form. If the form is valid then we are saving this in the database.
33. Import machine learning model in django app
34.save the models folder in the static folder
35. Add the machinelearning.py file in the app folder

36. In views.py file import the pipeline_model function
37. In views.py extract the image object from database
Code:

```
primary_key = save.pk
        imgobj = FaceRecognition.objects.get(pk=primary_key)
        fileroot = str(imgobj.image)
        filepath = os.path.join(settings.MEDIA_ROOT,fileroot)
        results = pipeline_model(filepath)
        print(results)
```

38. Now display the result when form is valid
39. In media folder create a new folder ml_output to save the images
40. Display the images of ml_output in index.html
41. Finally styling the webapp with bootstrap

## Comparative study

Here we took 95 samples to check the accuracy of the model. We checked it manually and got the result:

Total no. of 'YES': **86**
Total no. of 'NO': **9**
Total no. of photos: **95**
Accuracy: **90.52%**

**TEST RESULT OF 95 PHOTOS**

| Name | file name | Face Detection score | Face Score | Emotion | Emotion Score | Matching |
|---|---|---|---|---|---|---|
| ABID | abid | 0.9966966 | 0.5732444439 | neutral | 0.2118983152 | Yes |

| | | | | | | |
|---|---|---|---|---|---|---|
| | abid 1 | 0.99991417 | 0.8180170362 | happy | 0.2107843745 | Yes |
| | abid 2 | 0.99943036 | 0.7402016069 | neutral | 0.2760899742 | Yes |
| | abid 3 | 0.999851 | 0.8947569697 | neutral | 0.2699298354 | Yes |
| | abid 4 | 0.9995351 | 0.5732444439 | neutral | 0.2536698338 | Yes |
| | abid 5 | 0.9966966 | 0.7777935877 | neutral | 0.2509350355 | Yes |
| | abid 6 | 0.99957544 | 0.924570844 | sad | 0.2118983152 | Yes |
| | abid 7 | 0.99954456 | 0.8593116804 | neutral | 0.2523951973 | Yes |
| | abid 8 | 0.99991417 | 0.7777935877 | neutral | 0.2699298354 | No |
| | abid 9 | 0.99943036 | 0.924570844 | happy | 0.2536698338 | No |
| | abid 10 | 0.999851 | 0.8593116804 | neutral | 0.2509350355 | Yes |
| | abid 11 | 0.9903352 | 0.9384130639 | neutral | 0.2474332997 | Yes |
| | abid 12 | 0.9903352 | 0.9384130639 | neutral | 0.2474332997 | Yes |
| | abid 13 | 0.99943036 | 0.8180170362 | happy | 0.2699298354 | Yes |
| | abid 14 | 0.999851 | 0.7402016069 | neutral | 0.2536698338 | Yes |
| | abid 15 | 0.9995351 | 0.8947569697 | neutral | 0.2509350355 | Yes |
| | abid 16 | 0.9966966 | 0.5732444439 | neutral | 0.2118983152 | Yes |
| | abid 17 | 0.99957544 | 0.7777935877 | happy | 0.2523951973 | Yes |
| | abid 18 | 0.99954456 | 0.924570844 | happy | 0.2699298354 | Yes |

| | | | | | | |
|---|---|---|---|---|---|---|
| | abid 19 | 0.9998362 | 0.8593116804 | neutral | 0.2536698338 | Yes |
| | abid 20 | 0.999851 | 0.5732444439 | neutral | 0.2509350355 | Yes |
| | abid 21 | 0.9995351 | 0.7777935877 | neutral | 0.2536698338 | Yes |
| | abid 22 | 0.99943036 | 0.924570844 | neutral | 0.2509350355 | Yes |
| PRATTAY | prattay 1 | 0.99959415 | 0.7634740955 | neutral | 0.2828151599 | Yes |
| | prattay 2 | 0.9916825 | 0.6745780736 | neutral | 0.337855998 | Yes |
| | prattay 3 | 0.99886477 | 0.5539139129 | neutral | 0.3912469666 | Yes |
| | prattay 4 | 0.9974955 | 0.455306955 | neutral | 0.4454087807 | Yes |
| | prattay 5 | 0.9974336 | 0.578677661 | neutral | 0.4051456793 | Yes |
| | prattay 6 | 0.9903352 | 0.9384130639 | neutral | 0.2474332997 | Yes |
| | prattay 7 | 0.9903352 | 0.9384130639 | neutral | 0.2474332997 | Yes |
| | prattay 8 | 0.99943036 | 0.8180170362 | happy | 0.2699298354 | Yes |
| | prattay 9 | 0.999851 | 0.7402016069 | neutral | 0.2536698338 | Yes |
| | prattay 10 | 0.9995351 | 0.8947569697 | neutral | 0.2509350355 | Yes |
| | prattay 11 | 0.99886477 | 0.5539139129 | neutral | 0.3912469666 | Yes |
| | prattay 12 | 0.9974955 | 0.455306955 | happy | 0.4454087807 | No |
| | prattay 13 | 0.9974336 | 0.578677661 | happy | 0.4051456793 | Yes |
| | prattay 14 | 0.9903352 | 0.9384130639 | neutral | 0.2474332997 | Yes |

| | | | | | | |
|---|---|---|---|---|---|---|
| | prattay 15 | 0.9903352 | 0.9384130639 | sad | 0.2474332997 | Yes |
| | prattay 16 | 0.99943036 | 0.8180170362 | sad | 0.2699298354 | Yes |
| | prattay 17 | 0.999851 | 0.7402016069 | neutral | 0.2536698338 | Yes |
| | prattay 18 | 0.9995351 | 0.8947569697 | sad | 0.2509350355 | Yes |
| | prattay 19 | 0.99886477 | 0.5539139129 | neutral | 0.3912469666 | No |
| | prattay 20 | 0.9974955 | 0.455306955 | happy | 0.4454087807 | No |
| | prattay 21 | 0.9995351 | 0.8947569697 | neutral | 0.2509350355 | Yes |
| | prattay 22 | 0.9966966 | 0.5732444439 | neutral | 0.2118983152 | Yes |
| | prattay 23 | 0.99957544 | 0.7777935877 | happy | 0.2523951973 | Yes |
| | prattay 24 | 0.9999826 | 0.6995131156 | neutral | 0.2000359719 | Yes |
| | prattay 25 | 0.9995468 | 0.4591035547 | neutral | 0.2434479096 | Yes |
| | prattay 26 | 0.99008536 | 0.605208237 | neutral | 0.2522344062 | Yes |
| | prattay 27 | 0.984244 | 0.5019538203 | neutral | 0.2929320884 | No |
| USUF | usuf 1 | 0.9982607 | 0.4056565009 | neutral | 0.2350173364 | Yes |
| | usuf 2 | 0.99571675 | 0.4687481952 | happy | 0.2524072447 | Yes |
| | usuf 3 | 0.99969065 | 0.7906173659 | neutral | 0.3403016753 | Yes |
| | usuf 4 | 0.9903352 | 0.9384130639 | neutral | 0.2474332997 | Yes |
| | usuf 5 | 0.9903352 | 0.9384130639 | neutral | 0.2474332997 | Yes |

| | | | | | | |
|---|---|---|---|---|---|---|
| | usuf 6 | 0.99943036 | 0.8180170362 | happy | 0.2699298354 | Yes |
| | usuf 7 | 0.999851 | 0.7402016069 | neutral | 0.2536698338 | Yes |
| | usuf 8 | 0.9995351 | 0.8947569697 | neutral | 0.2509350355 | Yes |
| | usuf 9 | 0.99886477 | 0.5539139129 | neutral | 0.3912469666 | Yes |
| | usuf 10 | 0.9974955 | 0.455306955 | happy | 0.4454087807 | Yes |
| | usuf 11 | 0.9974336 | 0.578677661 | happy | 0.4051456793 | Yes |
| | usuf 12 | 0.9903352 | 0.9384130639 | neutral | 0.2474332997 | Yes |
| | usuf 13 | 0.9903352 | 0.9384130639 | sad | 0.2474332997 | Yes |
| | usuf 14 | 0.99943036 | 0.8180170362 | sad | 0.2699298354 | Yes |
| | usuf 15 | 0.999851 | 0.7402016069 | neutral | 0.2536698338 | Yes |
| | usuf 16 | 0.9995351 | 0.8947569697 | sad | 0.2509350355 | Yes |
| | usuf 17 | 0.99886477 | 0.5539139129 | neutral | 0.3912469666 | No |
| | usuf 18 | 0.999851 | 0.8947569697 | neutral | 0.2699298354 | Yes |
| | usuf 19 | 0.9995351 | 0.5732444439 | neutral | 0.2536698338 | Yes |
| | usuf 20 | 0.9966966 | 0.7777935877 | neutral | 0.2509350355 | Yes |
| | usuf 21 | 0.99957544 | 0.924570844 | sad | 0.2118983152 | Yes |
| | usuf 22 | 0.99954456 | 0.8593116804 | neutral | 0.2523951973 | Yes |
| | usuf 23 | 0.99991417 | 0.7777935877 | neutral | 0.2699298354 | Yes |

| | | | | | | |
|---|---|---|---|---|---|---|
| | usuf 24 | 0.99943036 | 0.924570844 | happy | 0.2536698338 | Yes |
| | usuf 25 | 0.999851 | 0.859311680 4 | neutral | 0.2509350355 | Yes |
| VASHKAR | vashkar | 0.99943036 | 0.924570844 | happy | 0.2536698338 | Yes |
| | vashkar 1 | 0.9903352 | 0.938413063 9 | neutral | 0.2474332997 | Yes |
| | vashkar 2 | 0.9903352 | 0.938413063 9 | neutral | 0.2474332997 | Yes |
| | vashkar 3 | 0.99943036 | 0.818017036 2 | happy | 0.2699298354 | No |
| | vashkar 4 | 0.999851 | 0.740201606 9 | neutral | 0.2536698338 | Yes |
| | vashkar 5 | 0.9995351 | 0.894756969 7 | neutral | 0.2509350355 | Yes |
| | vashkar 6 | 0.9966966 | 0.573244443 9 | neutral | 0.2118983152 | Yes |
| | vashkar 7 | 0.99957544 | 0.777793587 7 | happy | 0.2523951973 | Yes |
| | vashkar 8 | 0.99954456 | 0.924570844 | happy | 0.2699298354 | Yes |
| | vashkar 9 | 0.9998362 | 0.859311680 4 | neutral | 0.2536698338 | Yes |
| | vashkar 10 | 0.999851 | 0.573244443 9 | neutral | 0.2509350355 | Yes |
| | vashkar 11 | 0.99943036 | 0.818017036 2 | happy | 0.2699298354 | Yes |
| | vashkar 12 | 0.999851 | 0.740201606 9 | neutral | 0.2536698338 | Yes |
| | vashkar 13 | 0.99008536 | 0.605208237 | neutral | 0.2522344062 | Yes |
| | vashkar 14 | 0.984244 | 0.501953820 3 | neutral | 0.2929320884 | Yes |
| | vashkar 15 | 0.9982607 | 0.405656500 9 | neutral | 0.2350173364 | No |

| | | | | | | |
|---|---|---|---|---|---|---|
| | vashkar 16 | 0.99977 | 0.4529060939 | surprise | 0.2429981474 | Yes |
| | vashkar 17 | 0.99955326 | 0.6412776478 | happy | 0.3575839668 | Yes |
| | vashkar 18 | 0.9998617 | 0.6174614199 | surprise | 0.2144606523 | Yes |
| | vashkar 19 | 0.9999856 | 0.3615744584 | neutral | 0.2282850853 | Yes |

# Codes

## 1. Face detection:

In [1]:
```
import numpy as np
import cv2
```

In [2]:
```
# load the image
img = cv2.imread('./images/faces.jpg')


cv2.imshow('faces',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

In [3]:
```
# load the model
net = cv2.dnn.readNetFromCaffe('./models/deploy.prototxt.txt',
                               './models/res10_300x300_ssd_iter_140000_fp16.caffemodel')
```

In [4]:
```
# extract blob
blob = cv2.dnn.blobFromImage(img,1,(300,300),(104,177,123),swapRB=False)
```

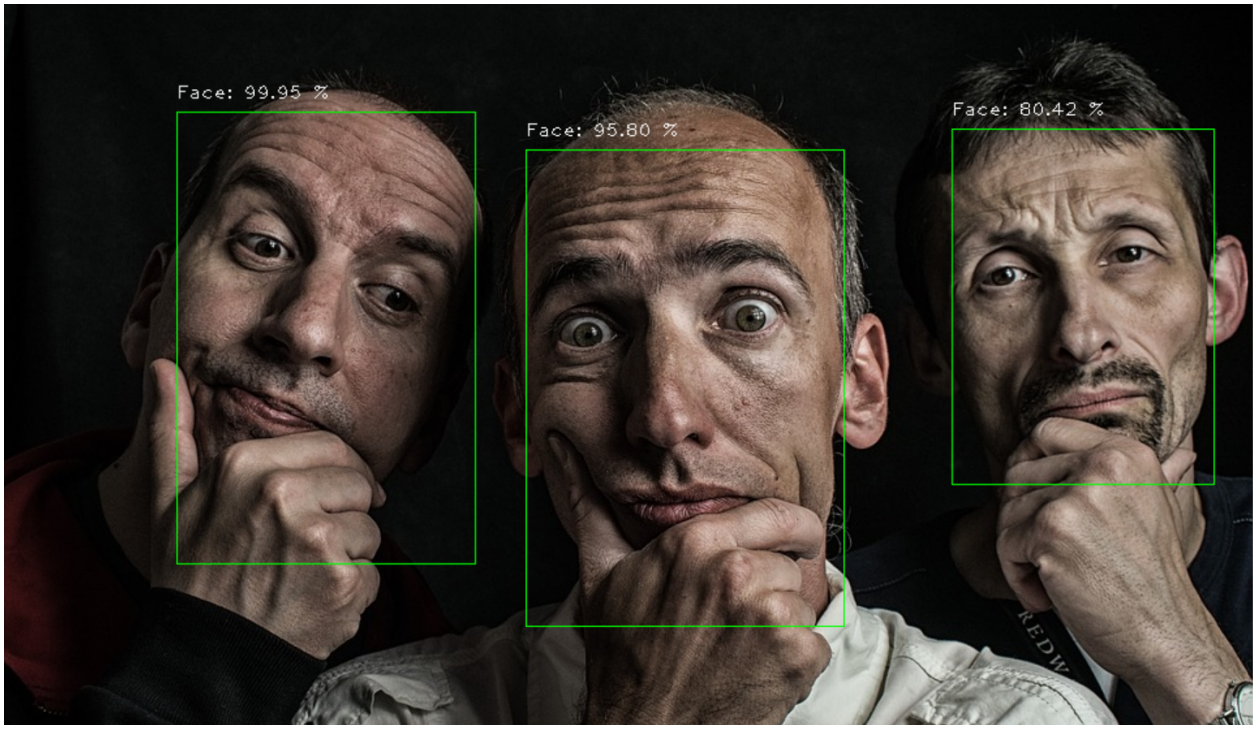
In [5]:
```
# set the blob as input
net.setInput(blob)
# run the model
detections = net.forward()
```

In [6]:
```
detections.shape
```

Out[6]:
```
(1, 1, 200, 7)
```

In [7]:
```
h,w = img.shape[:2]
for i in range(0,detections.shape[2]):
    confidence = detections[0,0,i,2]
    if confidence >= 0.5:
        #print(confidence)
        # bounding box (3:7)
        box = detections[0,0,i,3:7] # normalized bounding box values
        box = box*np.array([w,h,w,h])
        box = box.astype(int)
        startx, starty , endx, endy = box
        # draw the rectangle
        cv2.rectangle(img,(startx,starty),(endx,endy),(0,255,0))

        # put text
        text = 'Face: {:.2f} %'.format(confidence*100)
        cv2.putText(img,text,(startx,starty-10),cv2.FONT_HERSHEY_PLAIN,1,(255,255,255),)

cv2.imshow('face detect',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

## Output:




## 2. Facial Landmark Detection:

In [1]:
```
import numpy as np
import cv2
import dlib
from imutils import face_utils
```

In [3]:
```
img = cv2.imread('./images/faces.jpg')

cv2.imshow('girl',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

In [4]:
```
shape_predictor = dlib.shape_predictor('./models/shape_predictor_68_face_landmarks.dat')
shape_descriptor = dlib.face_recognition_model_v1('./models/dlib_face_recognition_resnet_model_v1.
```

In [5]:
```
img = cv2.imread('./images/girl.png')

face_detector = dlib.get_frontal_face_detector()

faces = face_detector(img)
for box in faces:
    pt1 = box.left(), box.top()
    pt2 = box.right(),box.bottom()
    face_shape = shape_predictor(img,box)
    face_shape_array = face_utils.shape_to_np(face_shape)
#      shape features
    face_descriptors = shape_descriptor.compute_face_descriptor(img,face_shape)
    cv2.rectangle(img,pt1,pt2,(0,255,0))
    for points in face_shape_array:
        #print(points)
        cv2.circle(img,tuple(points),3,(0,0,255),-1)

cv2.imshow('face detection',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

In [8]:
```
img = cv2.imread('./images/girl.png')

face_detector = dlib.get_frontal_face_detector()

faces = face_detector(img)
for box in faces:
    pt1 = box.left(), box.top()
    pt2 = box.right(),box.bottom()
    #print(pt1,pt2)

    face_shape = shape_predictor(img,box)
    face_shape_array = face_utils.shape_to_np(face_shape)
    # shape features
    face_descriptors = shape_descriptor.compute_face_descriptor(img,face_shape)
    cv2.rectangle(img,pt1,pt2,(0,255,0))
    for points in face_shape_array:
        #print(points)
        cv2.circle(img,tuple(points),2,(255,255,0),-2)
        pass


cv2.imshow('girl face detection',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

In [14]:
```
cv2.imwrite('face_des.png',img)
```

Out[14]: True

In [15]:
```
face_descriptors
```

Out[15]:
```
dlib.vector([-0.122912, 0.126352, 0.0877077, -0.103509, -0.11298, -0.0178538, -0.131934, -0.0753639,
0.164523, -0.12853, 0.192641, -0.0809671, -0.195524, -0.0284624, -0.0606989, 0.259382, -0.176004, -0.
122276, -0.0320793, 0.0205416, 0.0374298, 0.0251715, 0.0651995, 0.0893108, -0.103029, -0.403407, -0.0
961287, -0.0729469, -0.0552002, -0.0919216, -0.059494, -0.0250439, -0.190665, -0.0554678, 0.0330582,
0.141298, -0.0299519, -0.13384, 0.161559, -0.0126258, -0.266413, 0.0455602, 0.0607114, 0.236676, 0.18
9006, 0.0315033, 0.0495759, -0.175878, 0.178238, -0.131863, 0.0458499, 0.0938344, 0.0429922, 0.044530
7, -0.000268403, -0.0782624, 0.0709054, 0.204195, -0.173517, -0.0565751, 0.122763, -0.0614678, 0.0166
945, -0.117914, 0.244285, 0.0789957, -0.107951, -0.238579, 0.127426, -0.139137, -0.0498377, 0.038031
8, -0.140566, -0.17438, -0.319862, 0.0184531, 0.285739, 0.137397, -0.131725, 0.087328, -0.0491395, -
0.00242186, 0.0154873, 0.214119, 0.0224107, 0.0557129, -0.0591235, -0.0153325, 0.227827, -0.0595998,
0.0444306, 0.222787, 0.0020873, 0.0398717, -0.00242115, -0.0068744, -0.180852, 0.0780849, -0.123843,
0.00730216, -0.0511493, 0.031348, 0.013019, 0.111372, -0.204943, 0.146884, -0.0678012, 0.00914753, 0.
0163435, 0.00893977, -0.0564449, -0.132546, 0.127748, -0.220445, 0.122949, 0.177668, 0.120745, 0.1363
64, 0.141978, 0.125994, 0.00625331, 0.0390894, -0.258206, 0.0297453, 0.123886, -0.0511775, 0.0872768,
0.0406906])
```

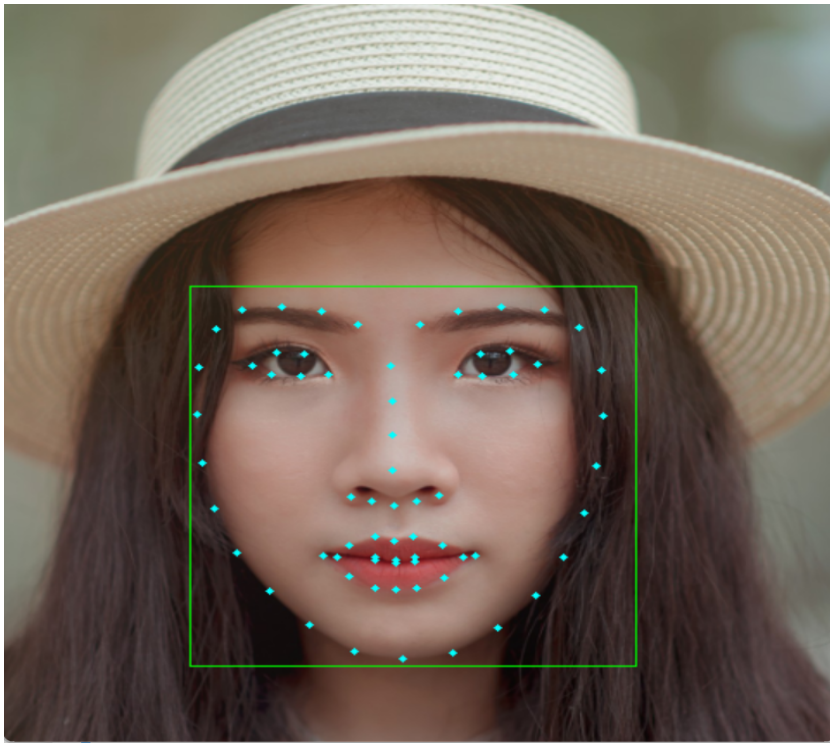
In [16]:
```
len(face_descriptors)
```

Out[16]:
```
128
```

In [17]:
```
np.array(face_descriptors)
```

Out[17]:
```
array([-1.22912213e-01,  1.26351953e-01,  8.77076909e-02, -1.03509061e-01,
       -1.12980135e-01, -1.78537704e-02, -1.31933898e-01, -7.53638893e-02,
        1.64522991e-01, -1.28530487e-01,  1.92640916e-01, -8.09670836e-02,
       -1.95523769e-01, -2.84623615e-02, -6.06988817e-02,  2.59382337e-01,
       -1.76003590e-01, -1.22276329e-01, -3.20792906e-02,  2.05416139e-02,
        3.74297760e-02,  2.51714792e-02,  6.51995316e-02,  8.93107578e-02,
       -1.03029013e-01, -4.03407305e-01, -9.61286873e-02, -7.29469061e-02,
       -5.52001856e-02, -9.19215679e-02, -5.94939515e-02, -2.50439215e-02,
       -1.90664873e-01, -5.54677956e-02,  3.30582410e-02,  1.41297787e-01,
       -2.99518667e-02, -1.33840293e-01,  1.61558568e-01, -1.26257930e-02,
       -2.66412973e-01,  4.55601588e-02,  6.07113540e-02,  2.36675903e-01,
        1.89006224e-01,  3.15033235e-02,  4.95759398e-02, -1.75878108e-01,
        1.78237543e-01, -1.31862834e-01,  4.58499268e-02,  9.38344300e-02,
        4.29921821e-02,  4.45307270e-02, -2.68403441e-04, -7.82624334e-02,
        7.09053650e-02,  2.04194829e-01, -1.73517063e-01, -5.65751232e-02,
        1.22762546e-01, -6.14678040e-02,  1.66945271e-02, -1.17913753e-01,
        2.44284824e-01,  7.89956748e-02, -1.07950762e-01, -2.38579124e-01,
        1.27425671e-01, -1.39136806e-01, -4.98377346e-02,  3.80317718e-02,
       -1.40565664e-01, -1.74379915e-01, -3.19862038e-01,  1.84531212e-02,
        2.85738647e-01,  1.37397021e-01, -1.31725237e-01,  8.73280168e-02,
       -4.91395369e-02, -2.42185732e-03,  1.54872620e-02,  2.14119360e-01,
        2.24107150e-02,  5.57129011e-02, -5.91234565e-02, -1.53325303e-02,
        2.27827296e-01, -5.95998168e-02,  4.44306023e-02,  2.22787216e-01,
        2.08729692e-03,  3.98716554e-02, -2.42114952e-03, -6.87440438e-03,
       -1.80852115e-01,  7.80848637e-02, -1.23843193e-01,  7.30215618e-03,
       -5.11492938e-02,  3.13479751e-02,  1.30189862e-02,  1.11372054e-01,
       -2.04942659e-01,  1.46883994e-01, -6.78011551e-02,  9.14752949e-03,
        1.63435247e-02,  8.93976539e-03, -5.64449206e-02, -1.32546321e-01,
        1.27748236e-01, -2.20445201e-01,  1.22949466e-01,  1.77667737e-01,
        1.20744668e-01,  1.36364058e-01,  1.41978398e-01,  1.25993639e-01,
        6.25331327e-03,  3.90893780e-02, -2.58205682e-01,  2.97453497e-02,
        1.23885550e-01, -5.11774980e-02,  8.72768238e-02,  4.06905897e-02])
```

# Output:



# 3. Face Data Preprocessing:

In [1]:
```
import numpy as np
import cv2
import pandas as pd
import os
import pickle
```

## Face Detection Model

- Load the models

In [2]:
```
face_detection_model = './models/res10_300x300_ssd_iter_140000.caffemodel'
face_detection_proto = './models/deploy.prototxt.txt'
face_descriptor = './models/openface.nn4.small2.v1.t7'
# load models using cv2 dnn
detector_model = cv2.dnn.readNetFromCaffe(face_detection_proto,face_detection_model)
descriptor_model = cv2.dnn.readNetFromTorch(face_descriptor)
```

In [3]:
```
!dir vashkar(data)
```

```
 Volume in drive D has no label.
 Volume Serial Number is 9080-4D15

 Directory of D:\final year project\4_MachineLearning_jupyter_notebook_models_test_data\vashkar(data)

17-11-2022  13:04    <DIR>          .
18-11-2022  01:51    <DIR>          ..
15-11-2022  20:49    <DIR>          Cristiano Ronaldo
15-11-2022  20:50    <DIR>          Lionel Messi
16-11-2022  23:59    <DIR>          Ms Dhoni
17-11-2022  00:00    <DIR>          Roger Federer
15-11-2022  19:46    <DIR>          Virat kohli
               0 File(s)              0 bytes
               7 Dir(s)   2,109,587,456 bytes free
```

In [4]:
```
def helper(image_path):
    img = cv2.imread(image_path)
    # step-1: face detection
    image = img.copy()
    h,w = image.shape[:2]
    img_blob = cv2.dnn.blobFromImage(image,1,(300,300),(104,177,123),swapRB=False,crop=False)
    # set the input
    detector_model.setInput(img_blob)
    detections = detector_model.forward()

    if len(detections) > 0:
        i = np.argmax(detections[0,0,:,2])# consider the face with max confidence score
        confidence = detections[0,0,i,2]
        if confidence > 0.5:
            box = detections[0,0,i,3:7]*np.array([w,h,w,h])
            (startx,starty,endx,endy) = box.astype('int')
            # step-2: Feature Extraction or Embedding
            roi = image[starty:endy,startx:endx].copy()
            # get the face descriptors
            faceblob = cv2.dnn.blobFromImage(roi,1/255,(96,96),(0,0,0),swapRB=True,crop=True)
            descriptor_model.setInput(faceblob)
            vectors = descriptor_model.forward()

            return vectors
    return None

```

## apply helper function to all images and get face descriptors

In [56]:
```
data = dict(data=[],label=[])
```

In [57]:
```
folders = os.listdir('vashkar(data)')
for folder in folders:
    filenames = os.listdir('vashkar(data)/{}'.format(folder))
    for filename in filenames:
        try:

            vector = helper('./vashkar(data)/{}/{}'.format(folder,filename))
            if vector is not None:
                data['data'].append(vector)
                data['label'].append(folder)
                print('Feature Extracted Sucessfully')

        except:
            pass
```

```
Feature Extracted Sucessfully
```

```
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
```

In [58]:
```
1 data.keys()
```

Out[58]:
```
dict_keys(['data', 'label'])
```

In [59]:
```
1 pd.Series(data['label']).value_counts()
```

Out[59]:
```
Virat kohli          870
Ms Dhoni             599
Lionel Messi         114
Roger Federer        111
Cristiano Ronaldo    109
dtype: int64
```

In [60]:
```
1 # save the data
2 pickle.dump(data,open('data_face_features.pickle',mode='wb'))
```

In [ ]:
```
1
```

# 4. Train Machine Learning Model of Faces:

In [108]:
```
import numpy as np
import pandas as pd
import pickle
```

## 1. Data

- Load data from pickle file
- split the data into independent and dependent
- split to train and test set

In [109]:
```
data = pickle.load(open('data_face_features.pickle',mode='rb'))
```

In [110]:
```
X = np.array(data['data']) # indendepent variable
y = np.array(data['label']) # dependent variable
```

In [111]:
```
X.shape , y.shape
```

Out[111]: ((1803, 1, 128), (1803,))

In [112]:
```
X = X.reshape(-1,128)
X.shape
```

Out[112]: (1803, 128)

In [113]:
```
# split the data into train and test
from sklearn.model_selection import train_test_split
```

In [114]:
```
x_train,x_test,y_train,y_test = train_test_split(X,y,train_size=0.8,random_state=0)
```

In [115]:
```
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Out[115]: ((1442, 128), (361, 128), (1442,), (361,))

## 2. Train Machine Learning

In [116]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import classification_report, accuracy_score, f1_score
```

In [ ]:
```

```

In [117]:
```
def get_report(model, x_train,y_train,x_test,y_test):
    y_pred_train = model.predict(x_train)
    y_pred_test = model.predict(x_test)

    # accuracy score
    acc_train = accuracy_score(y_train,y_pred_train)
    acc_test = accuracy_score(y_test,y_pred_test)

    # f1 score
    f1_score_train = f1_score(y_train,y_pred_train,average='macro')
```

```
    f1_score_test = f1_score(y_test,y_pred_test,average='macro')


    print('Accuracy Train = %0.2f'%acc_train)
    print('Accuracy Test = %0.2f'%acc_test)
    print('F1 Score Train = %0.2f'%f1_score_train)
    print('F1 Score Test = %0.2f'%f1_score_test)
```

### Decision Tree

In [118]:
```
model_decision = tree.DecisionTreeClassifier()
model_decision.fit(x_train,y_train)
```

Out[118]:
```
DecisionTreeClassifier()
```

In [119]:
```
get_report(model_decision,x_train,y_train,x_test,y_test)
```

```
Accuracy Train = 1.00
Accuracy Test = 0.69
F1 Score Train = 1.00
F1 Score Test = 0.54
```

### Logistic Regression

In [14]:
```
model_logistic = LogisticRegression()
model_logistic.fit(x_train,y_train) # training logistic regression
```

Out[14]:
```
LogisticRegression()
```

In [15]:
```
get_report(model_logistic,x_train,y_train,x_test,y_test)
```

```
Accuracy Train = 0.83
Accuracy Test = 0.80
F1 Score Train = 0.77
F1 Score Test = 0.70
```

### Support Vector Machines

In [16]:
```
model_svc = SVC(probability=True)
model_svc.fit(x_train,y_train)
```

Out[16]:
```
SVC(probability=True)
```

In [17]:
```
get_report(model_svc,x_train,y_train,x_test,y_test)
```

```
Accuracy Train = 0.90
Accuracy Test = 0.83
F1 Score Train = 0.86
F1 Score Test = 0.73
```

### Random Forest

In [18]:
```
model_rf = RandomForestClassifier(n_estimators=10,)
model_rf.fit(x_train,y_train)
```

Out[18]:
```
RandomForestClassifier(n_estimators=10)
```

In [19]:
```
get_report(model_rf,x_train,y_train,x_test,y_test)
```

```
Accuracy Train = 1.00
Accuracy Test = 0.75
F1 Score Train = 0.99
F1 Score Test = 0.55
```

## Voting Classifier

In [20]:
```
model_voting = VotingClassifier(estimators=[
    ('logistic',LogisticRegression()),
    ('svm',SVC(probability=True)),
    ('rf',RandomForestClassifier())
], voting='soft',weights=[2,3,1]) #voting = soft
```

In [21]:
```
model_voting.fit(x_train,y_train)
```

Out[21]:
```
VotingClassifier(estimators=[('logistic', LogisticRegression()),
                             ('svm', SVC(probability=True)),
                             ('rf', RandomForestClassifier())],
                 voting='soft', weights=[2, 3, 1])
```

In [22]:
```
get_report(model_voting,x_train,y_train,x_test,y_test)
```

```
Accuracy Train = 0.91
Accuracy Test = 0.83
F1 Score Train = 0.89
F1 Score Test = 0.74
```

# 3. Parameter Tuning

In [128]:
```
from sklearn.model_selection import GridSearchCV
```

In [129]:
```
model_grid = GridSearchCV(model_voting,
                          param_grid={
                              'svm__C':[3,5,7,10],
                              'svm__gamma':[0.1,0.3,0.5],
                              'rf__n_estimators':[5,10,20],
                              'rf__max_depth':[3,5,7],
                              'voting':['soft','hard']
                          },scoring='accuracy',cv=3,n_jobs=1,verbose=2)
```

In [130]:
```
model_grid.fit(x_train,y_train)
```

```
Fitting 3 folds for each of 216 candidates, totalling 648 fits
[CV] END rf__max_depth=3, rf__n_estimators=5, svm__C=3, svm__gamma=0.1, voting=soft; total time=
0.4s
[CV] END rf__max_depth=3, rf__n_estimators=5, svm__C=3, svm__gamma=0.1, voting=soft; total time=
0.3s
[CV] END rf__max_depth=3, rf__n_estimators=5, svm__C=3, svm__gamma=0.1, voting=soft; total time=
0.4s
[CV] END rf__max_depth=3, rf__n_estimators=5, svm__C=3, svm__gamma=0.1, voting=hard; total time=
0.3s
[CV] END rf__max_depth=3, rf__n_estimators=5, svm__C=3, svm__gamma=0.1, voting=hard; total time=
0.4s
[CV] END rf__max_depth=3, rf__n_estimators=5, svm__C=3, svm__gamma=0.1, voting=hard; total time=
0.3s
[CV] END rf__max_depth=3, rf__n_estimators=5, svm__C=3, svm__gamma=0.3, voting=soft; total time=
0.3s
[CV] END rf__max_depth=3, rf__n_estimators=5, svm__C=3, svm__gamma=0.3, voting=soft; total time=
0.3s
[CV] END rf__max_depth=3, rf__n_estimators=5, svm__C=3, svm__gamma=0.3, voting=soft; total time=
0.4s
```

```
[CV] END rf__max_depth=7, rf__n_estimators=20, svm__C=10, svm__gamma=0.5, voting=hard; total time=
0.6s
```

Out[27]:
```
GridSearchCV(cv=3,
             estimator=VotingClassifier(estimators=[('logistic',
                                                     LogisticRegression()),
                                                    ('svm',
                                                     SVC(probability=True)),
                                                    ('rf',
                                                     RandomForestClassifier())],
                                        voting='soft', weights=[2, 3, 1]),
             n_jobs=1,
             param_grid={'rf__max_depth': [3, 5, 7],
                         'rf__n_estimators': [5, 10, 20],
                         'svm__C': [3, 5, 7, 10], 'svm__gamma': [0.1, 0.3, 0.5],
                         'voting': ['soft', 'hard']},
             scoring='accuracy', verbose=2)
```

In [28]:
```
model_best_estimator = model_grid.best_estimator_
```

In [29]:
```
model_grid.best_score_
```

Out[29]:
```
0.850209343959344
```

In [30]:
```
model_grid.best_estimator_
```

Out[30]:
```
VotingClassifier(estimators=[('logistic', LogisticRegression()),
                             ('svm', SVC(C=10, gamma=0.5, probability=True)),
                             ('rf',
                              RandomForestClassifier(max_depth=7,
                                                     n_estimators=10))],
                 voting='soft', weights=[2, 3, 1])
```

### 4. Save Model

In [107]:
```
pickle.dump(model_best_estimator,open('./vashkar(models)/machinelearning_face_person_identity.pkl'
```

In [ ]:
```

```

# 5. Emotion Data Preprocessing:

In [1]:
```
import numpy as np
import cv2
import pandas as pd
import os
import pickle
```

### Face Detection Model

- Load the models

In [2]:
```
face_detection_model = './models/res10_300x300_ssd_iter_140000.caffemodel'
face_detection_proto = './models/deploy.prototxt.txt'
face_descriptor = './models/openface.nn4.small2.v1.t7'
# load models using cv2 dnn
detector_model = cv2.dnn.readNetFromCaffe(face_detection_proto,face_detection_model)
descriptor_model = cv2.dnn.readNetFromTorch(face_descriptor)
```

In [3]:
```
!dir emotion_data
```

```
 Volume in drive D has no label.
 Volume Serial Number is 9080-4D15

 Directory of D:\final year project\4_MachineLearning_jupyter_notebook_models_test_data\emotion_data

03-10-2022  02:12    <DIR>          .
01-11-2022  23:20    <DIR>          ..
03-10-2022  02:09    <DIR>          angry
03-10-2022  02:10    <DIR>          disgust
03-10-2022  02:10    <DIR>          fear
03-10-2022  02:11    <DIR>          happy
03-10-2022  02:11    <DIR>          neutral
03-10-2022  02:12    <DIR>          sad
03-10-2022  02:13    <DIR>          surprise
               0 File(s)              0 bytes
               9 Dir(s)  44,899,143,680 bytes free
```

In [3]:
```
def helper(image_path):
    img = cv2.imread(image_path)
    # step-1: face detection
    image = img.copy()
    h,w = image.shape[:2]
    img_blob = cv2.dnn.blobFromImage(image,1,(300,300),(104,177,123),swapRB=False,crop=False)
    # set the input
    detector_model.setInput(img_blob)
    detections = detector_model.forward()

    if len(detections) > 0:
        i = np.argmax(detections[0,0,:,2])# consider the face with max confidence score
        confidence = detections[0,0,i,2]
        if confidence > 0.5:
            box = detections[0,0,i,3:7]*np.array([w,h,w,h])
            (startx,starty,endx,endy) = box.astype('int')
            # step-2: Feature Extraction or Embedding
            roi = image[starty:endy,startx:endx].copy()
            # get the face descriptors
            faceblob = cv2.dnn.blobFromImage(roi,1/255,(96,96),(0,0,0),swapRB=True,crop=True)
            descriptor_model.setInput(faceblob)
            vectors = descriptor_model.forward()

            return vectors
    return None
```

## apply helper function to all images and get face descriptors

In [4]:
```
data = dict(data=[],label=[])
```

In [5]:
```
folders = os.listdir('emotion_data')
for folder in folders:
    filenames = os.listdir('emotion_data/{}'.format(folder))
    for filename in filenames:
        try:

            vector = helper('./emotion_data/{}/{}'.format(folder,filename))
            if vector is not None:
                data['data'].append(vector)
                data['label'].append(folder)
                print('Feature Extracted Sucessfully')

        except:
            pass
```

```
Feature Extracted Sucessfully
```

```
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
Feature Extracted Sucessfully
```

In [9]:
```
data.keys()
```

Out[9]:
```
dict_keys(['data', 'label'])
```

In [10]:
```
pd.Series(data['label']).value_counts()
```

Out[10]:
```
neutral     292
sad         260
happy       244
fear        207
angry       189
surprise    181
disgust      40
dtype: int64
```

In [11]:
```
# save the data
pickle.dump(data,open('data_face_features_emotion.pickle',mode='wb'))
```

In [ ]:
```

```

# 6. Train Machine Learning Model of Emotions:

In [1]:
```
import numpy as np
import pandas as pd
import pickle
```

## 1. Data

- Load data from pickle file
- split the data into independent and dependent
- split to train and test set

In [2]:
```
data = pickle.load(open('data_face_features_emotion.pickle',mode='rb'))
```

In [3]:
```
X = np.array(data['data']) # indendepent variable
y = np.array(data['label']) # dependent variable
```

In [4]:
```
X.shape , y.shape
```

Out[4]:
```
((1413, 1, 128), (1413,))
```

In [5]:
```
X = X.reshape(-1,128)
X.shape
```

Out[5]:
```
(1413, 128)
```

In [6]:
```
# split the data into train and test
from sklearn.model_selection import train_test_split
```

In [7]:
```
x_train,x_test,y_train,y_test = train_test_split(X,y,train_size=0.8,random_state=0)
```

In [8]:
```
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Out[8]:
```
((1130, 128), (283, 128), (1130,), (283,))
```

## 2. Train Machine Learning

In [9]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import classification_report, accuracy_score, f1_score
```

### Logistic Regression

In [10]:
```
model_logistic = LogisticRegression()
model_logistic.fit(x_train,y_train) # training logistic regression
```

Out[10]:
```
LogisticRegression()
```

In [11]:
```
def get_report(model, x_train,y_train,x_test,y_test):
    y_pred_train = model.predict(x_train)
    y_pred_test = model.predict(x_test)

    # accuracy score
    acc_train = accuracy_score(y_train,y_pred_train)
    acc_test = accuracy_score(y_test,y_pred_test)

    # f1 score
    f1_score_train = f1_score(y_train,y_pred_train,average='macro')
    f1_score_test = f1_score(y_test,y_pred_test,average='macro')


    print('Accuracy Train = %0.2f'%acc_train)
    print('Accuracy Test = %0.2f'%acc_test)
    print('F1 Score Train = %0.2f'%f1_score_train)
    print('F1 Score Test = %0.2f'%f1_score_test)
```

In [12]:
```
get_report(model_logistic,x_train,y_train,x_test,y_test)
```

```
Accuracy Train = 0.31
Accuracy Test = 0.23
F1 Score Train = 0.23
```

## Support Vector Machines

In [13]:
```
model_svc = SVC(probability=True)
model_svc.fit(x_train,y_train)
```

Out[13]:
```
SVC(probability=True)
```

In [14]:
```
get_report(model_svc,x_train,y_train,x_test,y_test)
```

```
Accuracy Train = 0.42
Accuracy Test = 0.25
F1 Score Train = 0.32
F1 Score Test = 0.17
```

### Random Forest

In [15]:
```
model_rf = RandomForestClassifier(n_estimators=10,)
model_rf.fit(x_train,y_train)
```

Out[15]:
```
RandomForestClassifier(n_estimators=10)
```

In [16]:
```
get_report(model_rf,x_train,y_train,x_test,y_test)
```

```
Accuracy Train = 0.99
Accuracy Test = 0.47
F1 Score Train = 0.99
F1 Score Test = 0.48
```

## Voting Classifier

In [17]:
```
model_voting = VotingClassifier(estimators=[
    ('logistic',LogisticRegression()),
    ('svm',SVC(probability=True)),
    ('rf',RandomForestClassifier())
], voting='soft',weights=[2,3,1])
```

In [18]:
```
model_voting.fit(x_train,y_train)
```

Out[18]:
```
VotingClassifier(estimators=[('logistic', LogisticRegression()),
                             ('svm', SVC(probability=True)),
                             ('rf', RandomForestClassifier())],
                 voting='soft', weights=[2, 3, 1])
```

In [19]:
```
get_report(model_voting,x_train,y_train,x_test,y_test)
```

```
Accuracy Train = 0.84
Accuracy Test = 0.40
F1 Score Train = 0.84
F1 Score Test = 0.38
```

## 3. Parameter Tuning

In [20]:
```
from sklearn.model_selection import GridSearchCV
```

In [21]:
```
model_grid = GridSearchCV(model_voting,
                          param_grid={
                              'svm__C':[3,5,7,10],
                              'svm__gamma':[0.1,0.3,0.5],
                              'rf__n_estimators':[5,10,20],
                              'rf__max_depth':[3,5,7],
                              'voting':['soft','hard']
                          },scoring='accuracy',cv=3,n_jobs=1,verbose=2)
```

In [22]:
```
model_grid.fit(x_train,y_train)
```

```
Fitting 3 folds for each of 216 candidates, totalling 648 fits
[CV] END rf__max_depth=3, rf__n_estimators=5, svm__C=3, svm__gamma=0.1, voting=soft; total time=
0.6s
[CV] END rf__max_depth=3, rf__n_estimators=5, svm__C=3, svm__gamma=0.1, voting=soft; total time=
0.6s
[CV] END rf__max_depth=3, rf__n_estimators=5, svm__C=3, svm__gamma=0.1, voting=soft; total time=
0.6s
0.8s
```

Out[22]:
```
GridSearchCV(cv=3,
             estimator=VotingClassifier(estimators=[('logistic',
                                                     LogisticRegression()),
                                                    ('svm',
                                                     SVC(probability=True)),
                                                    ('rf',
                                                     RandomForestClassifier())],
                                        voting='soft', weights=[2, 3, 1]),
             n_jobs=1,
             param_grid={'rf__max_depth': [3, 5, 7],
                         'rf__n_estimators': [5, 10, 20],
                         'svm__C': [3, 5, 7, 10], 'svm__gamma': [0.1, 0.3, 0.5],
                         'voting': ['soft', 'hard']},
             scoring='accuracy', verbose=2)
```

In [48]:
```
model_best_estimator = model_grid.best_estimator_
```

In [49]:
```
model_grid.best_score_
```

Out[49]:
```
0.3026459356246591
```

## 4. Save Model

In [50]:
```
pickle.dump(model_best_estimator,open('./models/machinelearning_face_emotion.pkl',mode='wb'))
```

In [ ]:
```

```

# 7. Pipeline all the models:

In [6]:
```
import numpy as np
import cv2
import sklearn
import pickle
```

In [10]:
```
# face detection
face_detector_model = cv2.dnn.readNetFromCaffe('./models/deploy.prototxt.txt',
                                               './models/res10_300x300_ssd_iter_140000.caffemodel')
# feature extraction
face_feature_model = cv2.dnn.readNetFromTorch('./models/openface.nn4.small2.v1.t7')
# face recognition
face_recognition_model = pickle.load(open('./models/machinelearning_face_person_identity.pkl',
                                          mode='rb'))
# emotion recognition model
emotion_recognition_model = pickle.load(open('./models/machinelearning_face_emotion.pkl',mode='rb'
```

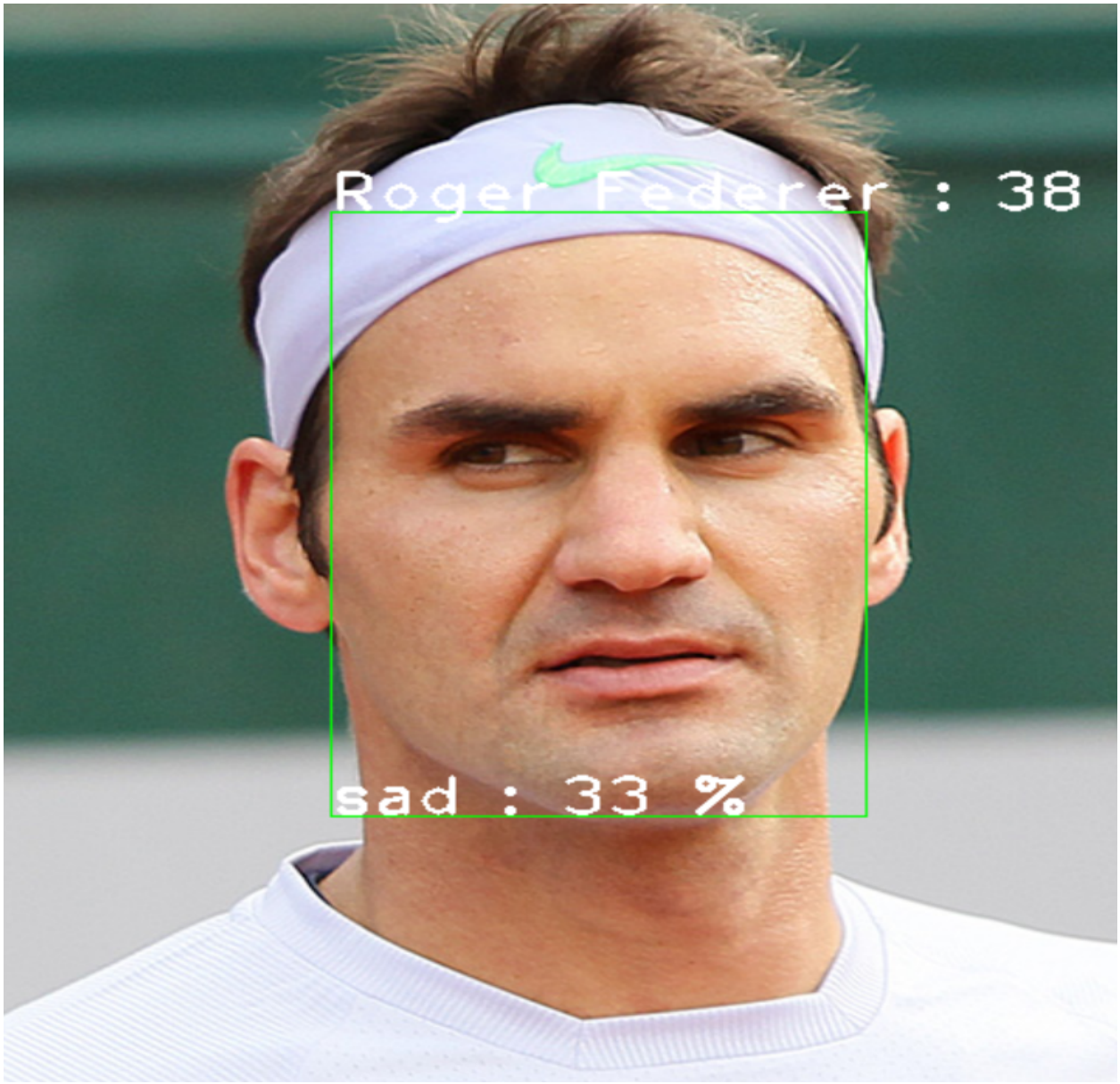
In [21]:
```
def pipeline_model(path):
    # pipeline model
    img = cv2.imread(path)
    image = img.copy()
    h,w = img.shape[:2]
    # face detection
    img_blob = cv2.dnn.blobFromImage(img,1,(300,300),(104,177,123),swapRB=False,crop=False)
    face_detector_model.setInput(img_blob)
    detections = face_detector_model.forward()

    # machcine results
    machinlearning_results = dict(face_detect_score = [],
                                  face_name = [],
                                  face_name_score = [],
                                  emotion_name = [],
                                  emotion_name_score = [],
                                  count = [])
    count = 1
    if len(detections) > 0:
        for i , confidence in enumerate(detections[0,0,:,2]):
            if confidence > 0.5:
                box = detections[0,0,i,3:7]*np.array([w,h,w,h])
                startx,starty,endx,endy = box.astype(int)

            cv2.rectangle(image,(startx,starty),(endx,endy),(0,255,0))

            # feature extraction
            face_roi = img[starty:endy,startx:endx]
            face_blob = cv2.dnn.blobFromImage(face_roi,1/255,(96,96),(0,0,0),swapRB=True,crop=True)
            face_feature_model.setInput(face_blob)
            vectors = face_feature_model.forward()

            # predict with machine learning
            face_name = face_recognition_model.predict(vectors)[0]
            face_score = face_recognition_model.predict_proba(vectors).max()
            # EMOTION
            emotion_name = emotion_recognition_model.predict(vectors)[0]
            emotion_score = emotion_recognition_model.predict_proba(vectors).max()

            text_face = '{} : {:.0f} %'.format(face_name,100*face_score)
            text_emotion = '{} : {:.0f} %'.format(emotion_name,100*emotion_score)
            cv2.putText(image,text_face,(startx,starty),cv2.FONT_HERSHEY_PLAIN,2,(255,255,255),2)
            cv2.putText(image,text_emotion,(startx,endy),cv2.FONT_HERSHEY_PLAIN,2,(255,255,255),2)

            machinlearning_results['count'].append(count)
            machinlearning_results['face_detect_score'].append(confidence)
            machinlearning_results['face_name'].append(face_name)
```

```
            machinlearning_results['emotion_name'].append(emotion_name)
            machinlearning_results['emotion_name_score'].append(emotion_score)

            count += 1


eturn image, machinlearning_results
```

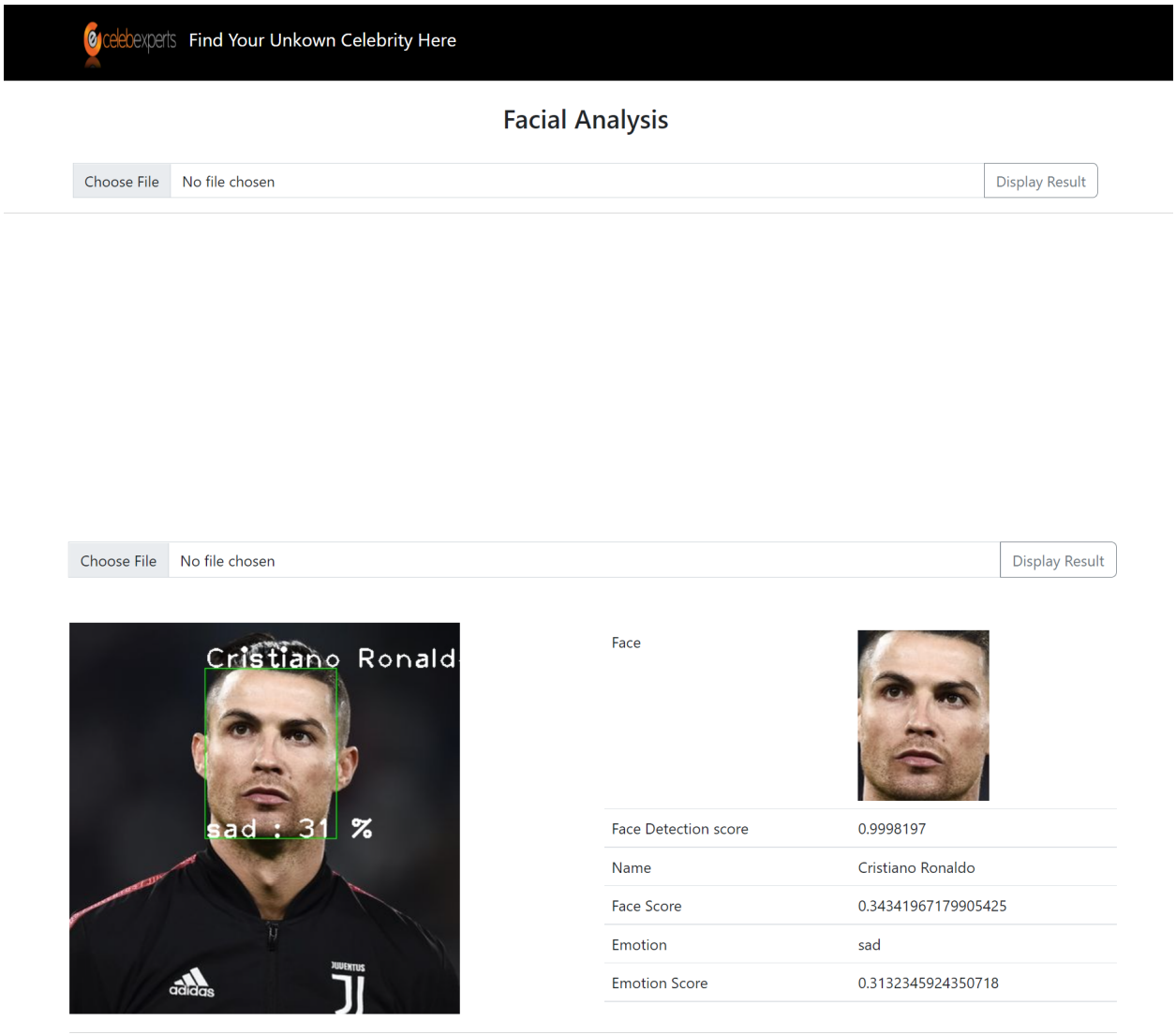
In [24]:

```
img, results = pipeline_model(r"D:\final year project\4_MachineLearning_jupyter_notebook_models_te
cv2.imshow('detection',img)
cv2.waitKey(0)
cv2.destroyAllWindows()

```

## Output:



# 8. UI


celebexperts
Find Your Unkown Celebrity Here
Facial Analysis
Choose File
No file chosen
Display Result
Choose File
No file chosen
Display Result
Cristiano Ronald
sad : 31 %
adidas
JUVENTUS
Face
Face Detection score
0.9998197
Name
Cristiano Ronaldo
Face Score
0.34341967179905425
Emotion
sad
Emotion Score
0.3132345924350718

# CONCLUSION

Face recognition systems are part of facial image processing applications and their significance as a research area are increasing recently. Implementations of the system are crime prevention, video surveillance, person verification, and similar security activities.

Main goal of the project is to study and implement a face recognition system. The goal is reached by face detection and recognition methods. Neural network  is used for face recognition.

We have accomplished the project using Algorithms namely Logistic Regression,Random Forest, Support Vector Machine then we run a voting classifier to combine all three machine learning algorithms which gives a 85% accuracy.

# CURRENT AND FUTURE WORK

Face recognition models are implemented and tested. Test results show that the model has acceptable performance. On the other hand, the model has some future work for improvement and implementation.

The study presented in this report is the first of a two part project. Here, the objectives were to study Machine Learning Algorithms and its implementation. In the first part we have created a model which can recognize a sports celebrity face and the emotion of the face. In the Second Part at first we increased the accuracy score of the model which is now for face recognition it is 85% and for emotion it is 30%. And secondly we want to predict the age of the person.

# BIBLIOGRAPHY


1. **Caffe** - Caffe is a deep learning framework, originally developed at University of California, Berkeley. It is open source, under a BSD license. It is written in C++, with a Python interface.

2. **Tensorflow** - TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

3. **PyTorch** - PyTorch is a machine learning framework based on the Torch library, used for applications such as computer vision and natural language processing, originally developed by Meta AI and now part of the Linux Foundation umbrella.

4. **Darknet -** Darknet is an open source neural network framework written in C and CUDA. It is fast, easy to install, and supports CPU and GPU computation.

5. **Logistic Regression** - Logistic regression aims to solve classification problems. It does this by predicting categorical outcomes, unlike linear regression that predicts a continuous outcome.

6. **Support Vector Machines (SVM)** - Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

7. **Random Forest -** A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

8. **Voting Classifier -** A voting classifier is a machine learning estimator that trains various base models or estimators and predicts on the basis of

aggregating the findings of each base estimator. The aggregating criteria can be combined with a decision of voting for each estimator output.

9. **Parameter Tuning -** Image result for parameter tuning in machine learning Hyperparameter tuning consists of finding a set of optimal hyperparameter values for a learning algorithm while applying this optimized algorithm to any data set.

10. **Model -** Model is a file that has been trained to recognize certain types of patterns. You train a model over a set of data, providing it an algorithm that it can use to reason over and learn from those data.

11. **CV2 -** cv2 is the module import name for opencv-python, "Unofficial pre-built CPU-only OpenCV packages for Python". The traditional OpenCV has many complicated steps involving building the module from scratch, which is unnecessary. I would recommend remaining with the opencv-python library.

12. **Train Data -** Training data is the data you use to train an algorithm or machine learning model to predict the outcome you design your model to predict. If you are using supervised learning or some hybrid that includes that approach, your data will be enriched with data labeling or annotation.

13. **Test Data -** What is Testing Data? Once your machine learning model is built (with your training data), you need unseen data to test your model. This data is called testing data, and you can use it to evaluate the performance and progress of your algorithms' training and adjust or optimize it for improved results.

14. **Pipeline -** Pipeline is an independently executable workflow of a complete machine learning task.

15. **Feature extraction -** Feature extraction refers to the process of transforming raw data into numerical features that can be processed while preserving the information in the original data set. It yields better results than applying machine learning directly to the raw data.

16. **Python -** Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation. Python is dynamically-typed and garbage-collected.

17. **Django -** Django is a free and open-source, Python-based web framework that follows the model–template–views architectural pattern.

18. **HTML -** The HyperText Markup Language or HTML is the standard markup language for documents designed to be displayed in a web browser.

19. **Numpy** - NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

20. **Sklearn** - Scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language.

21. **Pillow -** Python Imaging Library is a free and open-source additional library for the Python programming language that adds support for opening, manipulating, and saving many different image file formats.

22. **Django** - Django is a free and open-source, Python-based web framework that follows the model–template–views architectural pattern. It is maintained by the Django Software Foundation, an independent organization established in the US as a 501 non-profit.

23. **Sqlite3** - SQLite is a database engine written in the C programming language. It is not a standalone app; rather, it is a library that software developers embed in their apps.

24. **UI -** The user interface (UI) is the point of human-computer interaction and communication in a device. This can include display screens,

keyboards, a mouse and the appearance of a desktop. It is also the way through which a user interacts with an application or a website.

25. **CSS -** Cascading Style Sheets is a style sheet language used for describing the presentation of a document written in a markup language such as HTML or XML.

26. **JavaScript** - JavaScript, often abbreviated as JS, is a programming language that is one of the core technologies of the World Wide Web, alongside HTML and CSS. As of 2022, 98% of websites use JavaScript on the client side for webpage behavior, often incorporating third-party libraries.

27. **BLOB -** BLOB stands for a “Binary Large Object,” a data type that stores binary data. Binary Large Objects (BLOBs) can be complex files like images or videos, unlike other data strings that only store letters and numbers. A BLOB will hold multimedia objects to add to a database; however, not all databases support BLOB storage.

# Thank You

**TOP SHEET**

## PARTIAL FULFILLMENT OF BTECH DEGREE

PROJECT TITLE: **CELEBRITY FACE RECOGNITION**

AUTHORS:

1. ABID
2. PRATTAY
3. VASHKAR
4. USUF

**CONTENT**
**ABSTRACT**
**ACKNOWLEDGEMENT**
**INTRODUCTION**
**THEORY**

1. OBJECTIVE *(TO IDENTIFY A FACE WHETHER IT IS RECOGNIZED OR NOT AND ITS SOCIAL IMPACT OR USE)*
2. DEVELOPMENT LIFE CYCLE (FLOW CHART)
    a. DATA COLLECTION *(process)*
    b. DATA CLEANING *(crop)*
    c. MODEL TRAINING *(FEATURES, ALGORITHM USED)*
    d. MODEL TESTING
    e. UI