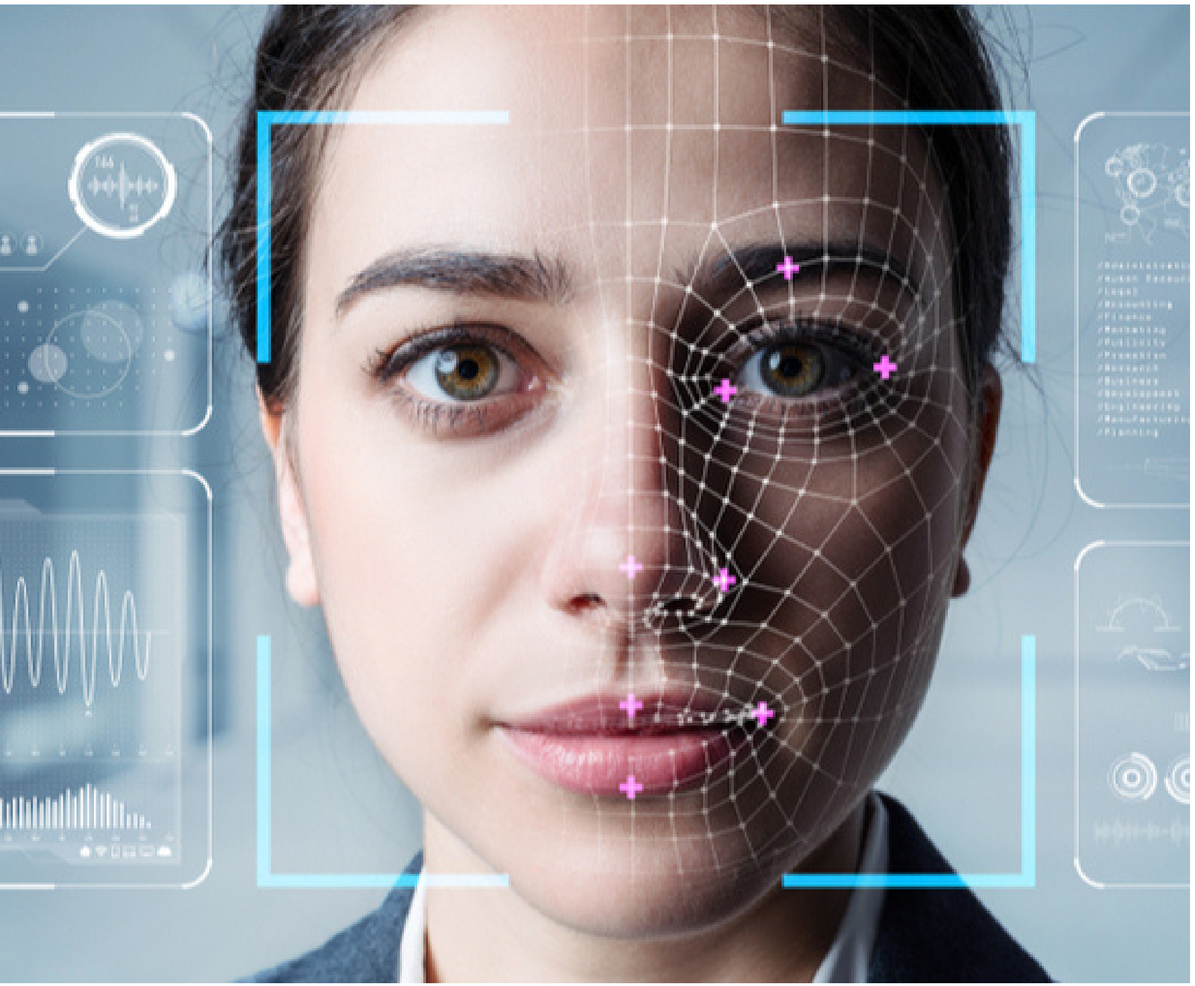
Demo Project
Face Detection and Recognition

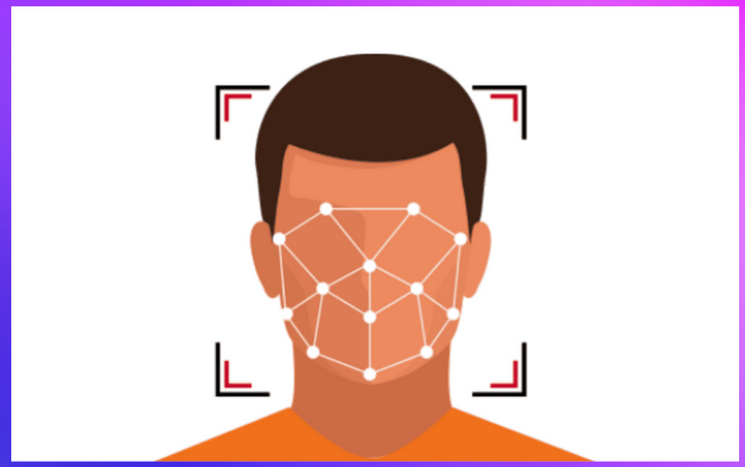
Presented by Aj Aun



Face Detection

Face Recognition

Face Detection



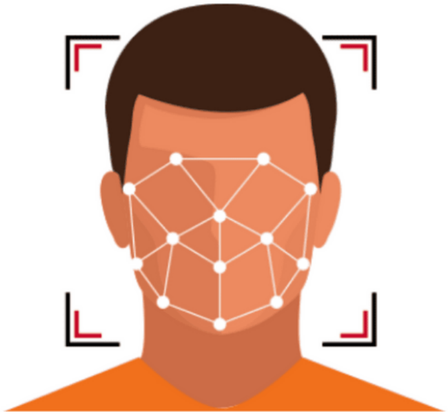
Find and identify human faces in image or video

Models examples

- Cascade Classifier : Haar cascade clasifier, LBP cascade classifier
- ANN Model: Dlib’CNN, OpenCv’s MTCNN

Face Recognition

Identify person



Mr. Somebody

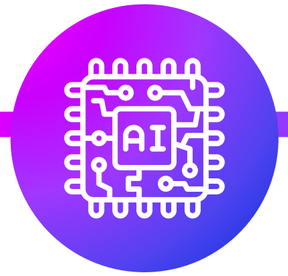
Models examples

- DeepFace
- FaceNet
- python library: face_recognition

Project Steps

Create dataset

Load Images form folders
Extract faces from images
Embed faces
Save dataset to train model



Train Model

Train model by using the collected data

Detect and Predict faces

Detect and predict unseen data

Requirements

Python libraries

- **opencv-python**
- **numpy**
- **scikit-learn**
- **keras-facenet**

OpenCv Cascade Classifier

- **haarcascade_frontalface_default.xml**

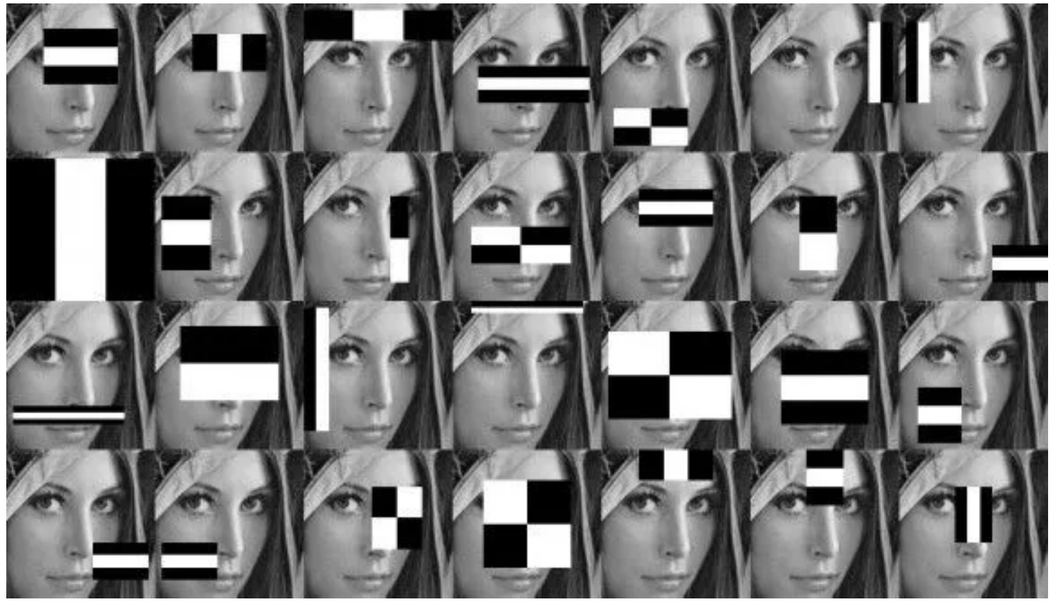
cmd: pip install -r requirements.txt

How does it work?

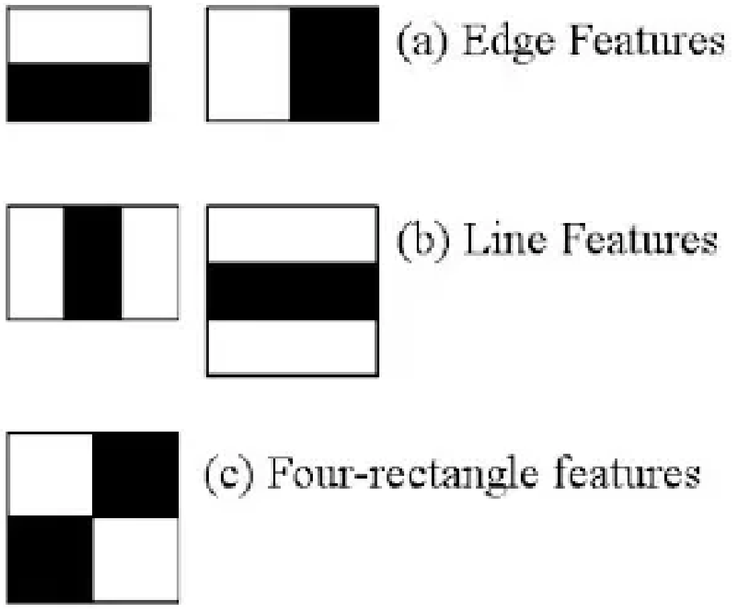
OpenCv Cascade Classifier

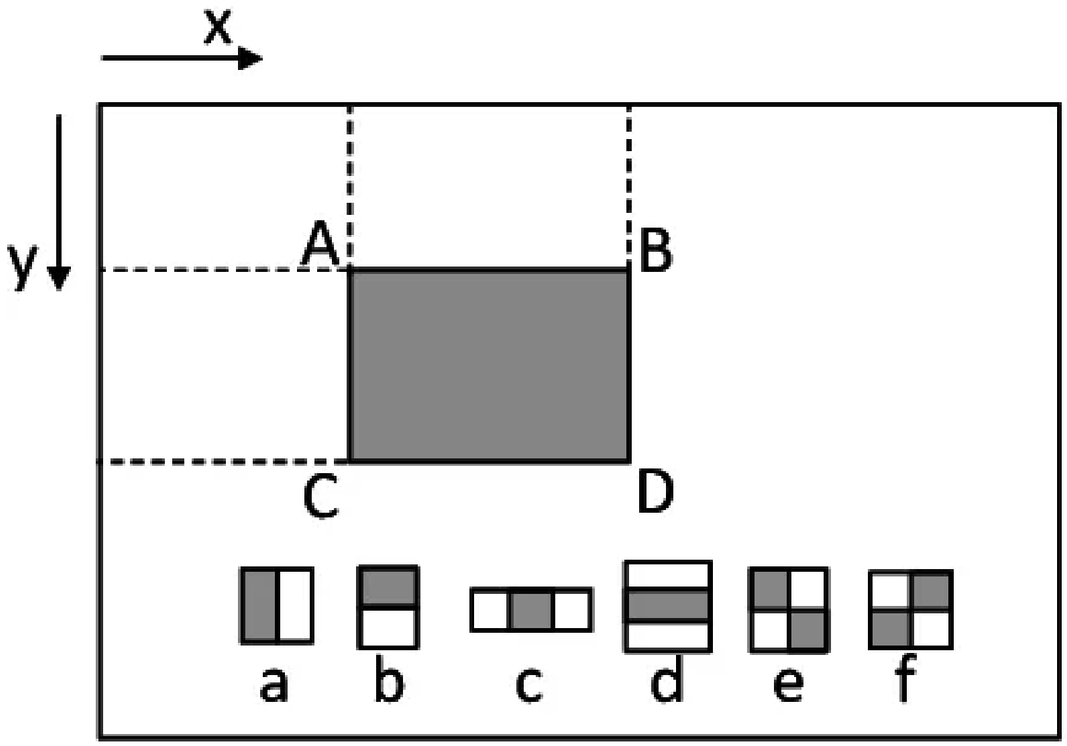
HaarCascadeClassifier

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.



Haar Cascade Classifier





In our project, we will use Haar CascadeClassifier

‘haarcascade_frontalface_default.xml’

- **Import python libraries**
- **Read image that we want to detect**
- **Convert image to grayscale image**

```
import cv2
import numpy as np
from keras_facenet import FaceNet

fname = 'sample_pic.jpg' # image path

# read image
im = cv2.imread(fname)
im = cv2.resize(im, (int(0.5 * im.shape[1]), int(0.5 * im.shape[0])))
gray_im = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
```

Detect faces form grayscale image and put the detected faces in a variable call ‘faces’

```
# detect face using cascade classifier
cascade = 'haarcascade_frontalface_default.xml'
face_detector = cv2.CascadeClassifier(cascade)

faces = face_detector.detectMultiScale(gray_im, scaleFactor=1.3, minNeighbors=5)
```

scaleFactor: Parameter specifying how much the image size is reduced at each image scale.
minNeighbors: Parameter specifying how many neighbors each candidate rectangle should have to retain it.

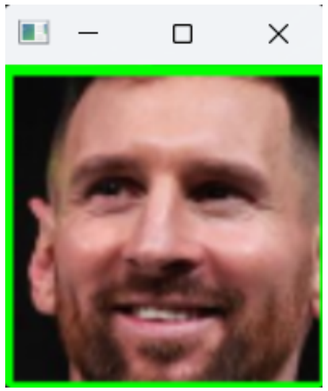
```
# draw rectangle over detected faces
for (x, y,w,h) in faces:
    cv2.rectangle(im, (x,y), (x+w,y+h), (0,255,0), 3)

# crop only face
face = im[y:y+h, x:x+w]
face = cv2.resize(face, (160, 160))
```

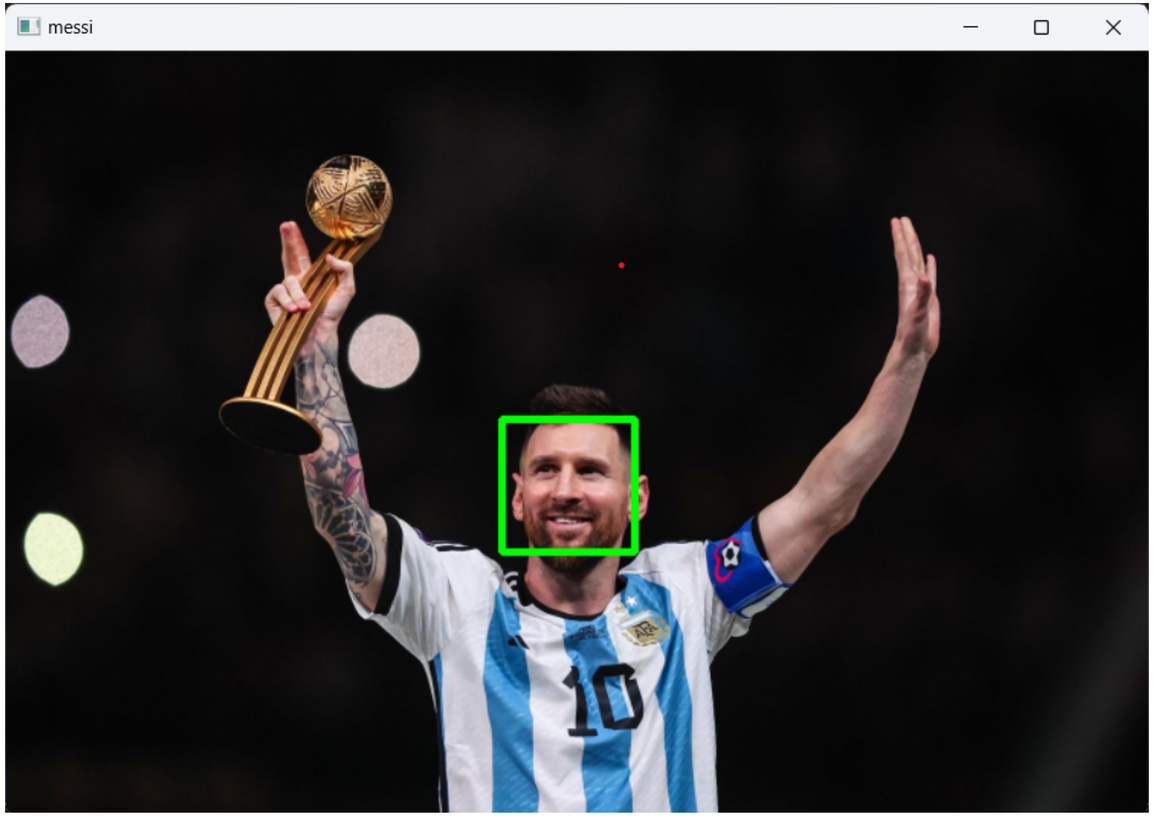
Draw green rectangle around detected face

Show the results

```
cv2.imshow('messi', im)
cv2.imshow('face', face)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Crop detected face

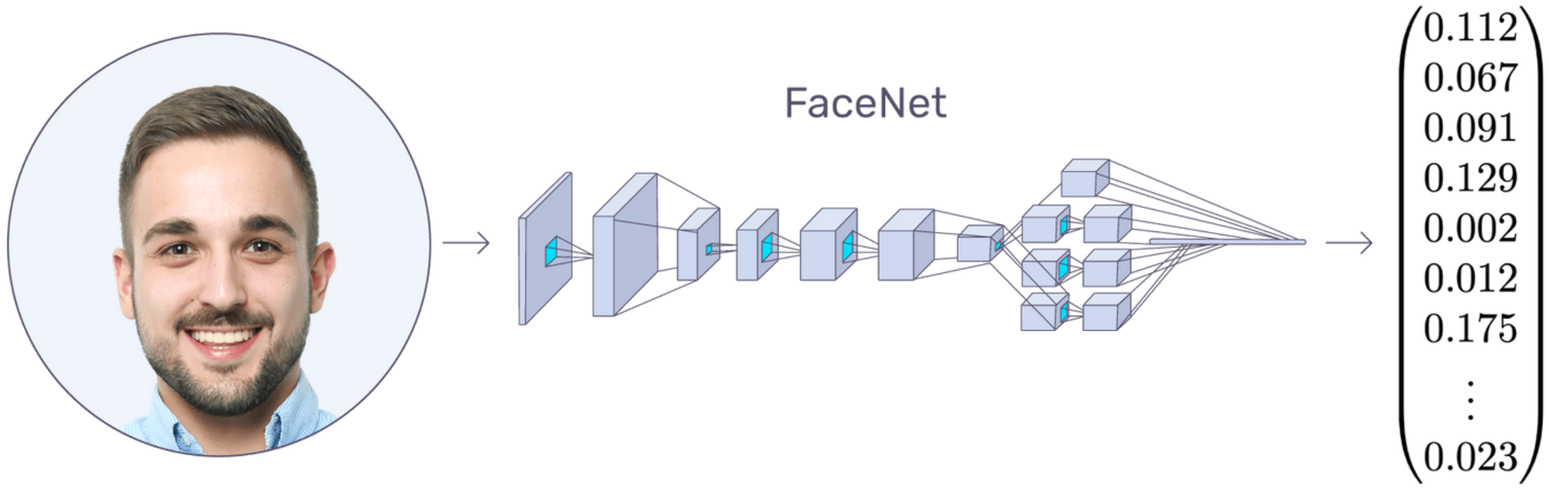


Draw rectangle around the detected face

FaceNet

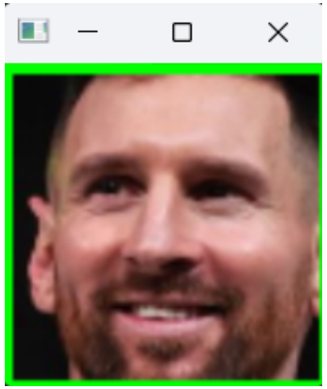
FaceNet is a face recognition system developed in 2015 by researchers at Google that achieved then state-of-the-art results on a range of face recognition benchmark datasets.

The FaceNet system can be used to extract high-quality features from faces, called face embeddings, that can then be used to train a face identification system.



Embed detected face

```python
# Embedding face
embedder = FaceNet()
face = face.astype('float32')
face = np.expand_dims(face, axis=0)
embedded_face = embedder.embeddings(face)
print(type(embedded_face))
print('shape {}'.format(embedded_face.shape))
print(embedded_face[0])
```

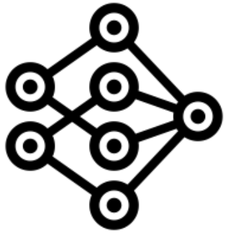


```
<class 'numpy.ndarray'>
shape (1, 512)
[ 1.65038509e-03 -1.45818936e-02 -1.84955820e-02  5.20322062e-02
 -6.56271428e-02 -1.61289722e-02 -5.22982422e-03  1.74717791e-02
  2.54099965e-02 -2.34829169e-02 -6.10550866e-03 -8.33897218e-02
  8.59125424e-03  2.74478621e-03 -7.14841159e-03  1.65900693e-03
 -1.30072823e-02 -1.47942696e-02  6.46465570e-02 -3.03735491e-02
 -8.24237391e-02  3.61343957e-02 -2.78507266e-02  8.68310686e-03
 -2.14373898e-02 -5.36523610e-02 -7.23546371e-02  2.54830029e-02
 -3.29461806e-02  4.85876463e-02 -5.59909269e-02  3.46090049e-02
 -2.92124283e-02 -2.16146372e-03 -4.88782786e-02  1.90453157e-02
 -1.46299200e-02  6.55608699e-02 -1.20242015e-02 -3.24143879e-02
 -1.85494274e-02  1.12573236e-01  6.07148446e-02  3.90291512e-02
  9.78511050e-02  2.43419074e-02  3.30425762e-02  1.32907426e-03
  6.21200353e-03 -2.42273454e-02 -6.89142244e-03  3.79897021e-02
  2.33215727e-02  5.85455634e-03 -1.54733965e-02  5.37426397e-02
  7.00638350e-03  1.78110749e-02  5.83855547e-02 -1.67913325e-02
  2.43807267e-02 -7.20830485e-02  1.04984261e-01 -5.26763313e-02
  2.07493156e-02  6.62739202e-02 -1.57232806e-02  5.80619350e-02
 -5.54347374e-02  6.59403279e-02 -3.01683601e-02  2.57662125e-02
  4.90715243e-02 -7.84386974e-03  2.25344282e-02 -7.12857693e-02
 -4.92605492e-02  4.97595854e-02  4.66299849e-03 -4.49189693e-02
  5.75855337e-02  7.97579661e-02 -5.50461076e-02  4.93327416e-02
```

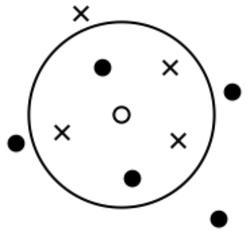
Train model

use embedded face to train classification model

SVC



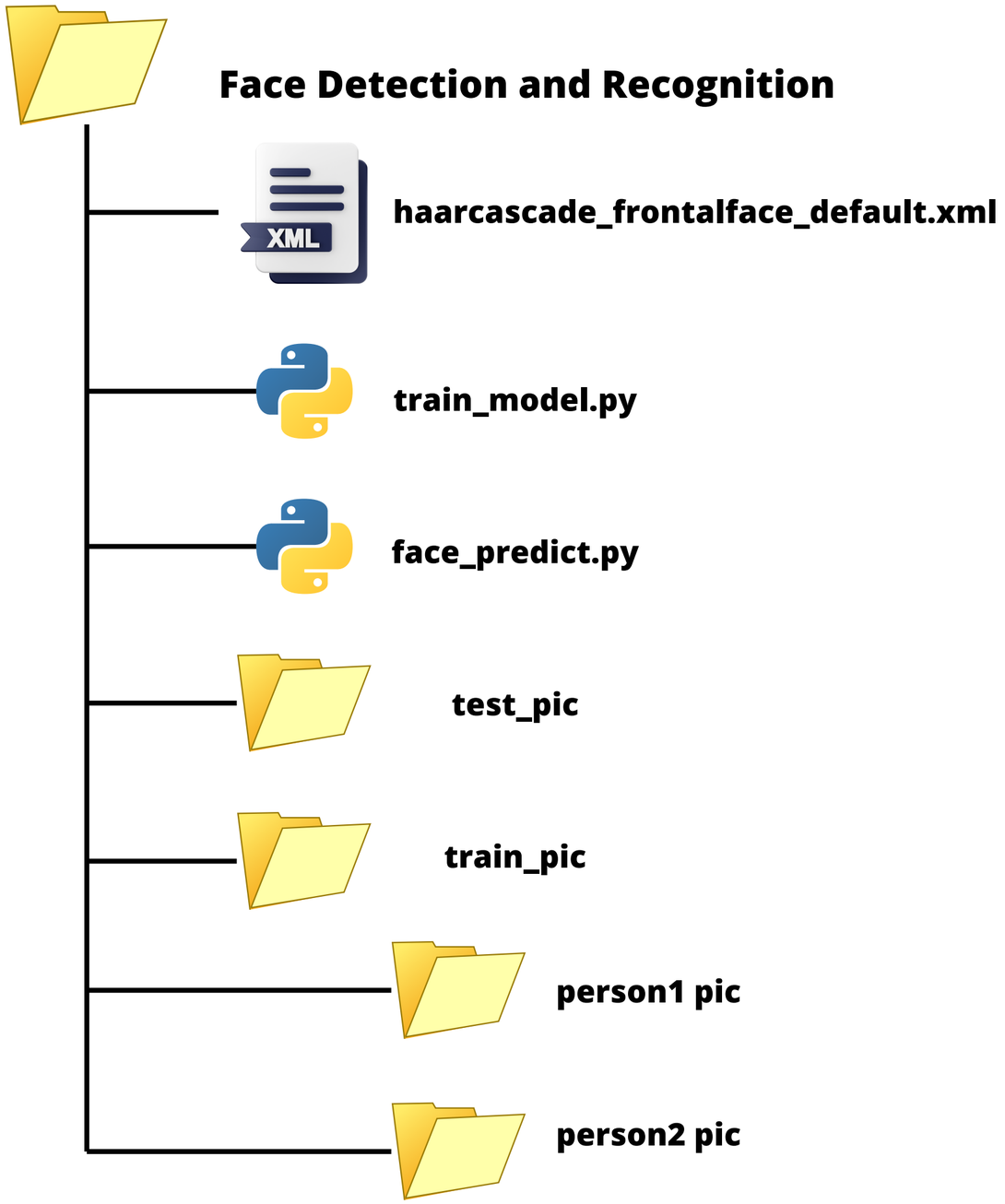
ANN



KNN

Other classification models

Project architecture




train_model.py

train_model.py

```
import cv2
import os
import numpy as np
from keras_facenet import FaceNet
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
import pickle
```

Import required libraries

train_model.py

```
# Create dataset to train Model
class CreateDataset:
    def __init__(self, directory):
        self.directory = directory
        self.target_size = (160, 160)
        self.X = []
        self.y = []
        self.detector = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
```

Create CreateDataset class to facilitate dataset creation work flow

train_model.py

```
def extract_face(self, fname):
    img = cv2.imread(fname)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = self.detector.detectMultiScale(img_gray,scaleFactor=1.3, minNeighbors=5)
    x, y, w, h = faces[0]
    face = img_rgb[y:y+h, x:x+w]
    face_arr = cv2.resize(face, self.target_size)
    return face_arr
```

extract_face(): convert image to grayscale and detect faces with Haarcascade classifier
get face position and resize face image to (160, 160) for passing to FaceNet

train_model.py

```
def load_faces(self, dir):
    FACES = []
    for im_name in os.listdir(dir):
        try:
            path = dir + im_name
            face = self.extract_face(path)
            FACES.append(face)
        except Exception as e:
            pass
    return FACES
```

load_face(): read .jpg files, pass to extract_face() and append to a list

train_model.py

```
def load_classes(self):
    for sub_dir in os.listdir(self.directory):
        path = self.directory + '/' + sub_dir +'/'
        FACES = self.load_faces(path)
        labels = [sub_dir for _ in range(len(FACES))]
        self.X.extend(FACES)
        self.y.extend(labels)
    return np.asarray(self.X), np.asarray(self.y)
```

load_class(): read each sub directories and pass to load_faces()
to create an array of X(features) and an array of y (target/class)

train_model.py

```
# FaceNet: faces --> vector
embedder = FaceNet()
def get_embedding(face_img):
    face_img = face_img.astype('float32')
    face_img = np.expand_dims(face_img, axis=0)
    face_vec = embedder.embeddings(face_img)
    return face_vec[0]
```

get_embedding(): take face image into vector

train_model.py

```
# Create dataset
im_dir = 'train_pic'
create_dataset = CreateDataset(im_dir)
X, y = create_dataset.load_classes()
EMBEDDED_X = []
for img in X:
    EMBEDDED_X.append(get_embedding(img))
EMBEDDED_X = np.asarray(EMBEDDED_X)
np.savez_compressed('face_dataset.npz', EMBEDDED_X, y)
```

create dataset with EMBEDDED_X (features) and y (target/class)

train_model.py　train SVM model

```
# Train SMV model
encoder = LabelEncoder()
y = encoder.fit_transform(y)
X_train, X_test, y_train, y_test = train_test_split(EMBEDDED_X, y,
                                                    shuffle=True, test_size=0.2,
                                                    random_state=1)
print(X_train)
print(y_train)
model = SVC(kernel='linear', probability=True)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
score = accuracy_score(y_test, y_pred)
print('score: {}'.format(score))
print('confusion metrix\n', confusion_matrix(y_test, y_pred))
print('classification report\n', classification_report(y_test, y_pred))

# save model
pickle.dump(model, open('my_face_rec_model.pkl', 'wb'))
```

face_predict.py

face_predict.py

```
import numpy as np
import pickle
from keras_facenet import FaceNet
import cv2
from sklearn.preprocessing import LabelEncoder

im_test = 'test_pic/messi_ronaldo_test_im.jpg'

facenet = FaceNet()
faces_embedding = np.load('face_dataset.npz')
y = faces_embedding['arr_1']
encoder = LabelEncoder()
encoder.fit(y)
cascade = 'haarcascade_frontalface_default.xml'
model = pickle.load(open('my_face_rec_model.pkl', 'rb'))
```

face_predict.py

```
face_detector = cv2.CascadeClassifier(cascade)

image = cv2.imread(im_test)
rgb_im = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
gray_im = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

faces = face_detector.detectMultiScale(gray_im, scaleFactor=1.3, minNeighbors=5)
```

face_predict.py

```
faces = face_detector.detectMultiScale(gray_im, scaleFactor=1.3, minNeighbors=5)
for x,y,w,h in faces:
    img = rgb_im[y:y+h, x:x+w]
    img = cv2.resize(img,(160,160))
    img = img.astype('float32')
    img = np.expand_dims(img, axis=0)
    y_pred = facenet.embeddings(img)
    target = model.predict_proba(y_pred)
    target_val = np.max(target)
    target_lable = [np.argmax(target)]
    if target_val > 0.7:
        final_name = encoder.inverse_transform(target_lable)[0]
    else:
        final_name = 'unknown'
    print(target)
    cv2.rectangle(image, (x,y), (x+w, y+h), (0, 255, 0), 3)
    cv2.putText(image, str(final_name), (x,y - 10),cv2.FONT_HERSHEY_SIMPLEX,
                1, (0,255, 0),
                3, cv2.LINE_AA)
cv2.imshow('Face recognition', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```