Face Recognition Based on MTCNN and FaceNet

Facility of computer science and artificial intelligence

CS396 Selected 2

Team number: 36

Alaa Eid Mahmoud	201900161
Eslam shabaan Eid	201900142
Abdelrahman Hussein Hamdy	201900412
Bothina Sayed Abdelsalam	201900221
Assem Emad Abdellatef	201900395
Mo'men Mamdouh Khlil	201900864
Asmaa Khaled Tawfeek	201900147
Mohamed Mohaned Farouk	201900738

Paper Details:

Face Recognition Based on MTCNN and FaceNet

Rongrong Jin, Hao Li, Jing Pan, Wenxi Ma, and Jingyu Lin

Publisher: Association for the Advancement of ArtificialIntelligence (www.aaai.org). 2021

Dataset: LFW dataset

Face Recognition Idea:

A face recognition system is a technology that uses a digital image or a video frame from a video source to identify and verify a person.

Facial recognition systems operate in a variety of ways, but in general, they compare chosen facial traits from a given image with faces in a database.

It's also known as a Biometric Artificial Intelligence-based application that analyses patterns based on a person's facial features and form to uniquely identify them.

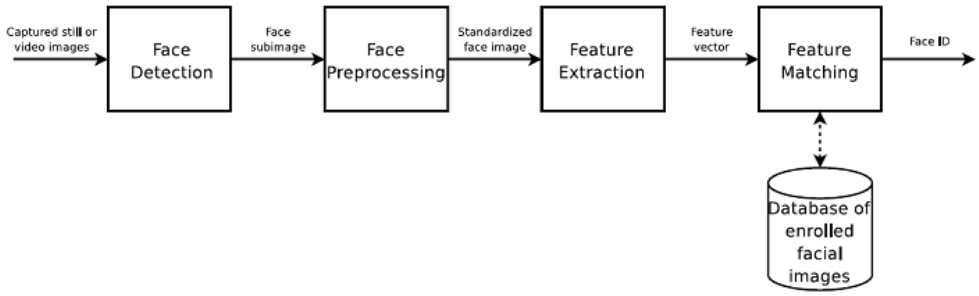
In recent years, facial recognition technologies become more common on because of smartphones and other technologies, such as robotics.

Computerized facial recognition necessitates the assessment of a person's physiological parameters.

Process of Face Recognition:

There are 4 main steps in all face recognition systems shown in figure:

1- Face Detection.
2- Face Analysis.
3- Feature Extraction.
4- Feature Mapping.



Introduction to our problem:

Face recognition performance improves rapidly with the recent deep learning technique.

However, face images in the wild undergo large intra-personal variations, which cause great challenges to face-related applications.

Our project paper will be based on MTCNN and FaceNet which will recover the canonical view of face image.

Traditional face recognition methods use feature operators to model face.

However, with the further research, these algorithms can show strong effectiveness in finding linear structures, but when facing potential nonlinear structures, they often achieve unsatisfactory recognition results.

With Deep Learning and CNN, the accuracy and speed of face recognition have made great strides.

However, results from different networks and models are very different.

Previous face recognition approaches based on deep networks use a classification layer they regard it as a classification task.

The number of SoftMax output is the number of face tags.

Therefore, every time a new sample comes in, the whole model needs to be retrained. FaceNet directly trains its output to be a compact 128-D embedding using a triplet-based loss function based on LMNN.

To achieve better performance, we first use MTCNN to do face detection, which is a mainstream target detection network with high detection accuracy, lightweight and

real-time.

Then use the result of MTCNN as the input of FaceNet to perform face recognition.

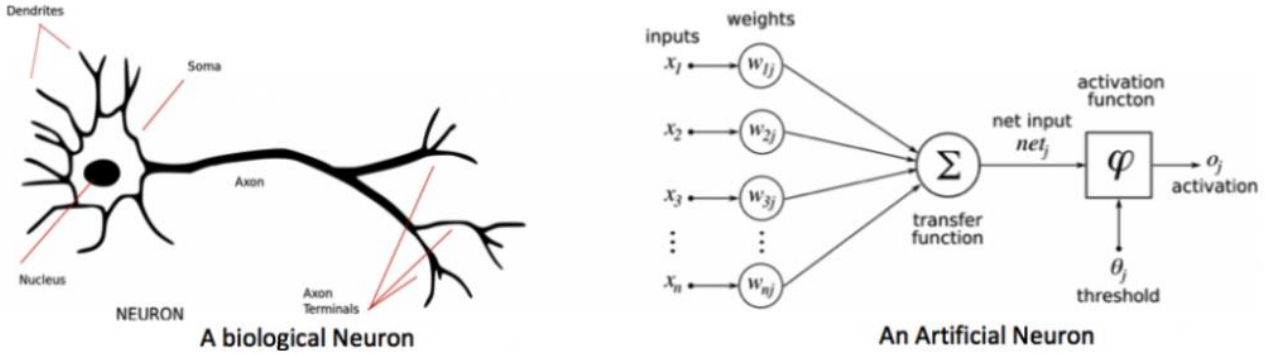
So mainly face recognition divided into two operations:

1- Face Detection
2- Face Recognition.

What are Neural Networks?

Neural Network is a series of algorithms that are trying to act like the human brain and find the relationship between the sets of data.
It also known as Artificial Neural Networks (ANN), which is a subset of machine learning and the umbrella of deep learning algorithms.

To understand the algorithm of ANN we must know how biological neuron work.



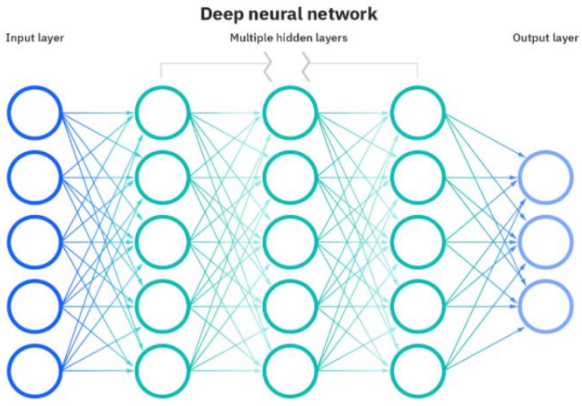
Biological Neuron Vs Artificial Neuron

Biological neurons transfer electrical impulses to other neurons, there are billions of neurons interconnected together inside the brain, and which are in turn connected to the spinal cord spreading nerves throughout the body collectively called the nervous system.

What are Artificial Neural Networks?

They are type of computational model that comprises of numerous processing components that receive inputs and produce outputs based on their activation functions.
The node layer of an ANN is made up of:

1. Input layer.
2. One or more hidden layer.
3. Output layer.



To learn and increase their accuracy over time, neural networks rely on training data.
Once these learning algorithms have been fine-tuned for accuracy, they can be useful tools in computer science and Artificial Intelligence, allowing us to classify and cluster data at rapid speeds.

Convolutional Neural Networks:

It is a class of Artificial Neural Networks that has become dominant in various computer vision tasks. It is useful with processing data that has a grid pattern, such as images. CNN is designed to learn spatial hierarchies of features automatically and adaptively through backpropagation by using multiple building blocks, such as:

1. Convolution layers.
2. pooling layers.
3. Flatten layers.

Hence, using steps like Convolution, Max pooling and, Flattening, they find out the important pixel combinations in the image and represents them as a set of numbers in a vector. Each of these numbers represents a unique feature of the image.

Model Methods:

For accurate face recognition, we train two networks, MTCNN and FaceNet.

MTCNN is a deep cascaded multi-task framework which exploits the inherent correlation between detection and alignment to boost up their performance.

The framework of MTCNN architecture includes three stages of deep convolutional networks to predict face and landmark location.

1- Candidate windows are produced through a fast Proposal Network (P-Net).
2- Refine these candidates in the next stage through a Refinement Network (R-Net).
3- Output Network (O-Net) produces final bounding box.

This affords the exact coordinates of the face.

FaceNet directly learns a mapping from face images to a compact Euclidean space where distances directly correspond to a measure of face similarity.

Once this space has been produced, our tasks can be easily implemented using standard techniques with FaceNet embeddings as feature vectors.

MTCNN Idea and Structure:

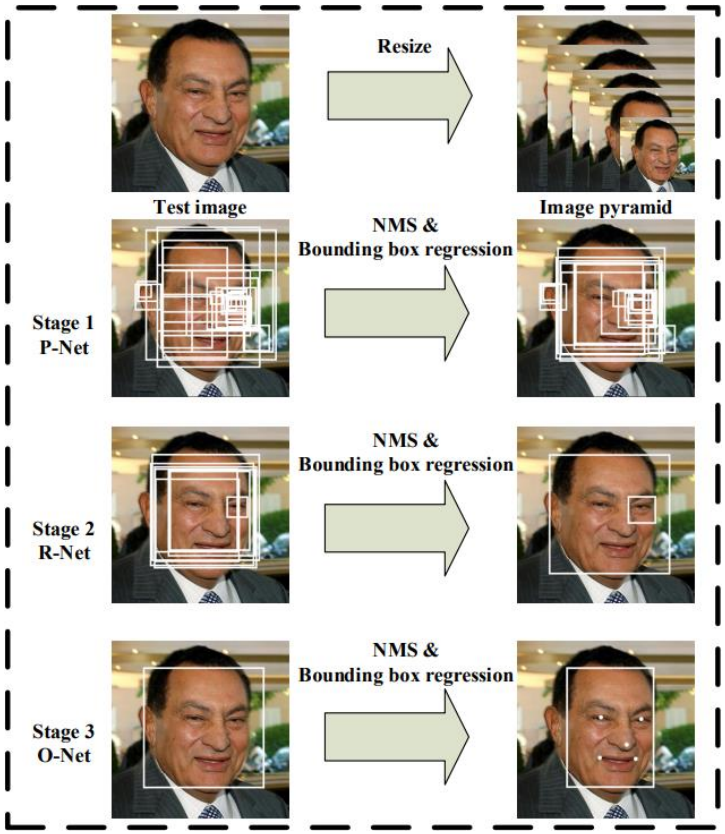
Face detection and alignment in an unconstrained environment are challenging due to various poses, illuminations, and occlusions.

Recent studies show that deep learning approaches can achieve impressive performance on these two tasks, that’s when MTCNN came to light.

MTCNN or Multi-Task Cascaded Convolutional Neural Networks is a neural network which detects faces and facial landmarks on images.

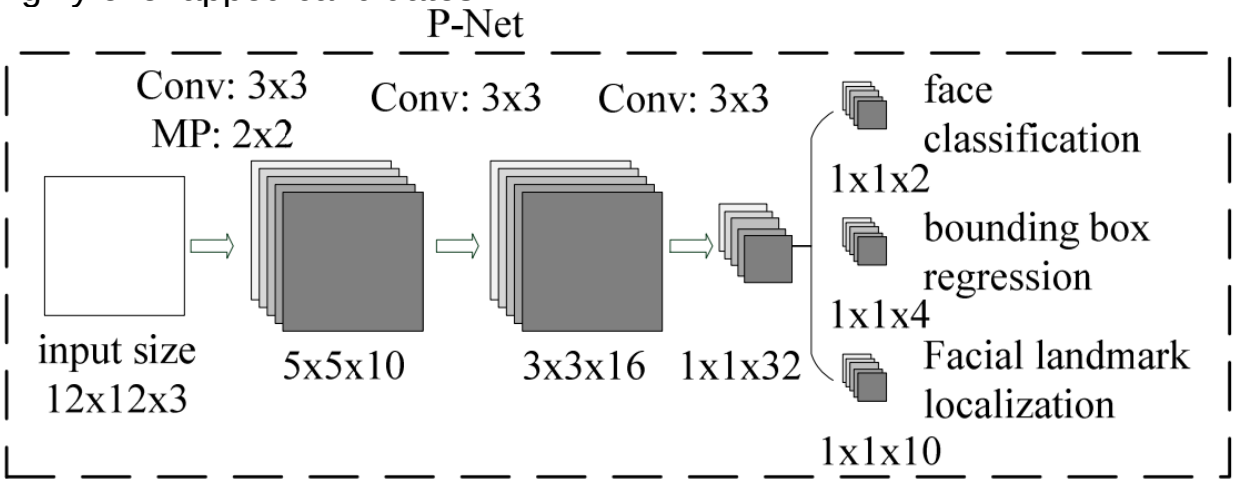
MTCNN idea:
It consists of 4 stages, pre-processing and 3 main stages.

Pre-processing stage is initially resizing the image to different scales to build an image pyramid, which is the input of the following three-stage cascaded framework:



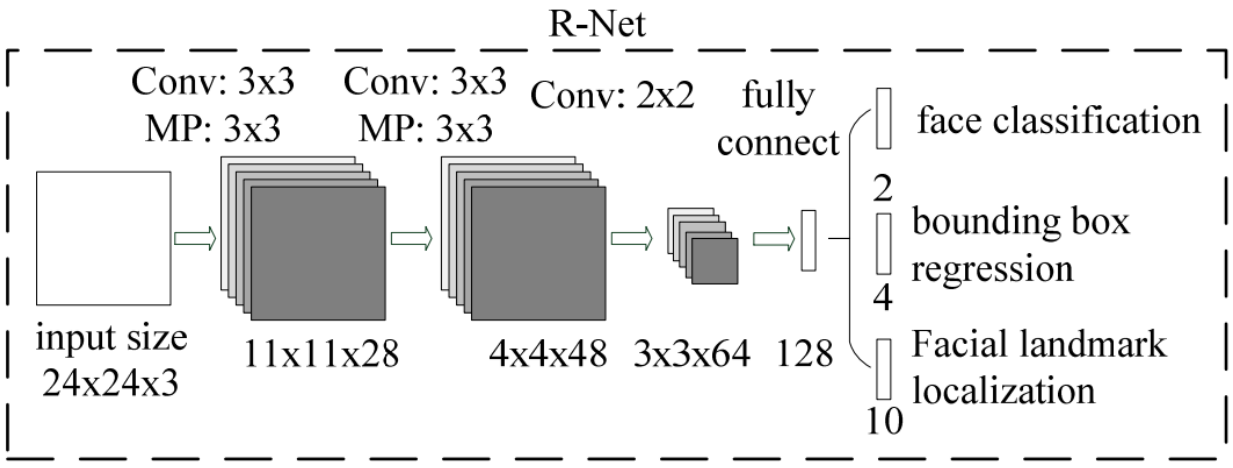
Stage 1: The Proposal Network (P-Net)

We exploit a fully convolutional network, called Proposal Network (P-Net), to obtain the candidate facial windows and their bounding box regression vectors.
Then candidates are calibrated based on the estimated bounding box regression vectors. After that, we employ non-maximum suppression (NMS) to merge highly overlapped candidates.



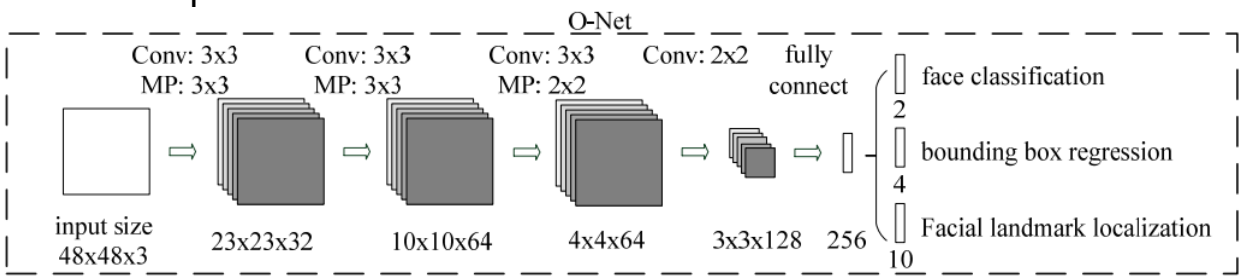
Stage 2: The Refine Network (R-Net)

All candidates are fed to another CNN, called Refine Network (R-Net), which further rejects a large number of false candidates, performs calibration with bounding box regression, and conducts NMS which merge more overlapped candidates.



Stage 3: The Output Network (O-Net)

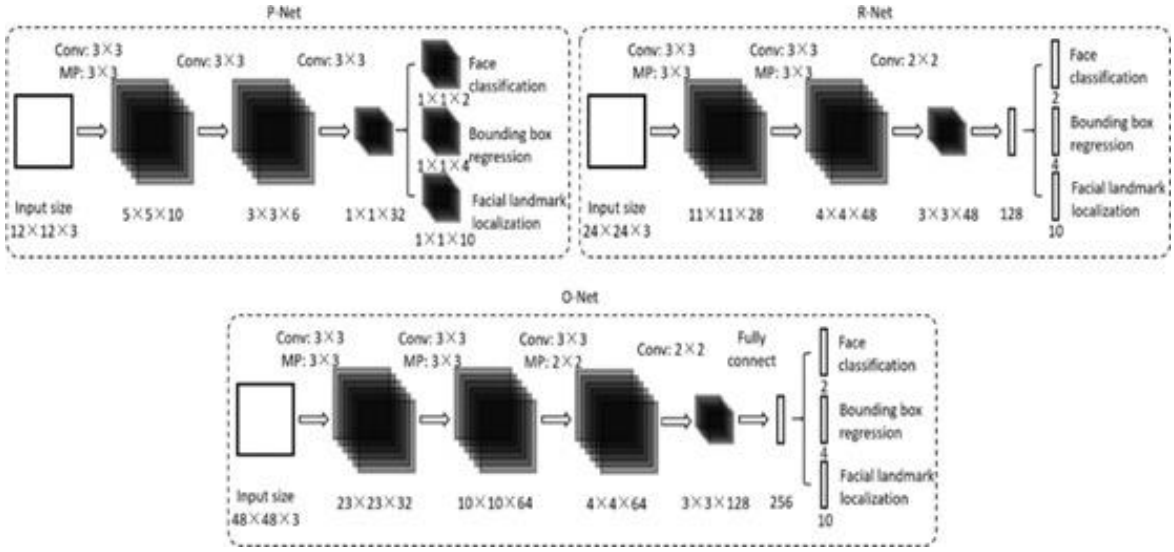
This stage is similar to the second stage, but in this stage, we aim to identify face regions with more supervision. In particular, the network will output five facial landmarks' positions.



CNN Architecture:

Using 3*3 filters reduce the computing and increase the depth to get a better performance.

We apply PReLU as nonlinearity activation function after the convolution and fully connection layers (except output layers).



Training:

We use three tasks to train our CNN detector:

1- Face classification: The learning objective is formulated as a two-class classification problem. For each sample, we use the cross-entropy loss.

$$L_i^{det} = -(y_i^{det} log(p_i) + (1 - y_i^{det})(1 - log(p^i)))$$

2- Bounding box regression: For each candidate window, we predict the offset between it and the nearest ground truth. The learning objective is formulated as a regression problem, and we employ the Euclidean loss for each sample.

$$L_i^{box} = \left\| \hat{y}_i^{box} - y_i^{box} \right\|_2^2$$

3- Facial landmark localization: Like bounding box regression task, is formulated as a regression problem and we minimize the Euclidean loss.

$$L_i^{landdmark} = \left\| \hat{y}_i^{landmark} - y_i^{landmark} \right\|_2^2$$

What is FaceNet:

FaceNet is the name of facial recognition pre-trained model that Google Researchers had proposed it in 2015 in a paper titled with:
(FaceNet: A unified Embedding for Face Recognition and Clustering).
It achieved state-of-the-art results in many benchmarks Face Recognition datasets.

How FaceNet works?

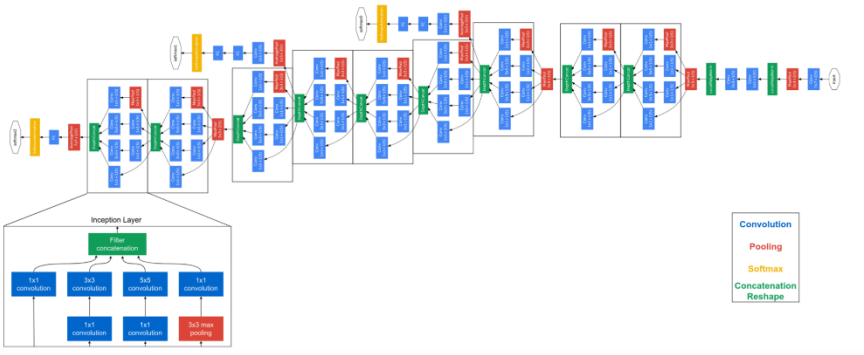
FaceNet takes an image of the person's face as input and outputs a vector of 128 numbers which represent the most important features of a face. In machine learning, this vector is called embedding. Why embedding? Because all the important information from an image is *embedded* into this vector. Basically, FaceNet takes a person's face and compresses it into a vector of 128 numbers. Ideally, embeddings of similar faces are also similar.

Why FaceNet in our model:

FaceNet directly trains its output to be a compact 128-D embedding using a triplet based loss function based on LMNN.
Our triplets consist of two matching face thumbnails and a non-matching face thumbnail, and the loss aims to separate the positive pair from the negative by a distance margin.
The thumbnails are tight crops of the face area.

The network is trained in the embedding space directly correspond to face similarity faces of the same person have small distances and faces of distinct people have large distances.

FaceNet Architecture



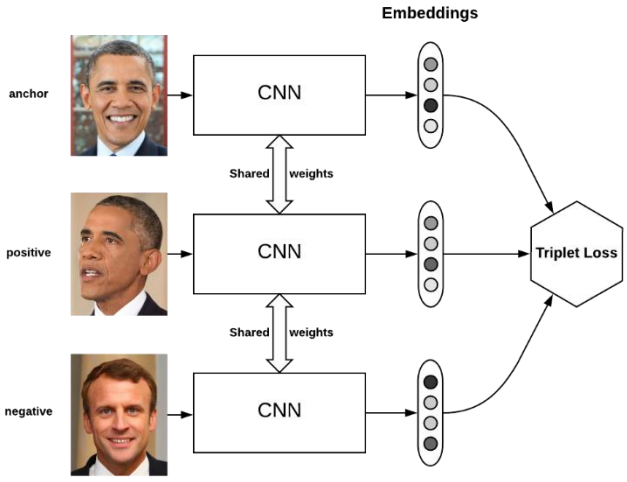
How FaceNet Learns?

End-to-End Learning:

Instead of using the traditional softmax method to do classification learning, FaceNet extracted a certain layer as a feature to learn a coding method from the image to The European space, and then do face recognition face verification and face clustering based on this code.

Given the model details and treating it as a black box, the most important part of our approach lies in the end-to-end learning of the whole system.

To this end we employ the triplet loss that directly reflects what we want to achieve in face verification, recognition, and clustering.



Triplet Loss:

The triplet loss is more suitable for face verification.

The motivation is that the loss from (Sun, Wang, and Tang 2014) encourages all faces of one identity to be projected onto a single point in the embedding space.

The triplet loss, however, tries to enforce a margin between each pair of faces from one person to all other faces.

This allows the faces for one identity to live on a manifold, while still enforcing the distance and thus discriminability to other identities.

Generating all possible triplets would result in many triplets that are easily satisfied. These triplets would not contribute to the training and result in slower convergence, as they would still be passed through the network.

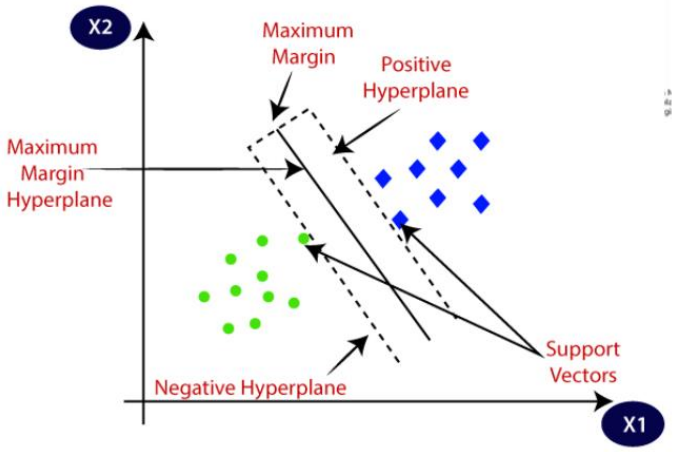
Data Augmentation

The image augmentation technology uses a series of random changes to the training images to generate similar but different training sample.
The image augmentation is that randomly changing the training samples can reduce the model's dependence on certain attributes and improve the generalization ability of the model.

Support Vector Machine Algorithm (SVM):

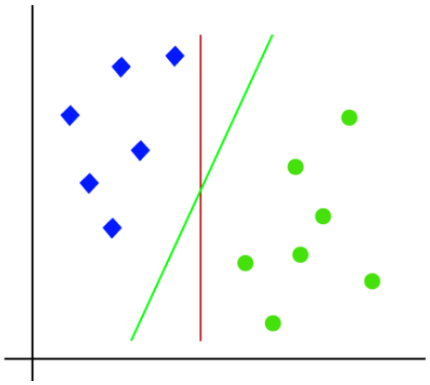
SVM is one of the most popular Supervised Learning algorithms, which is used for Classification problems in Machine Learning.
The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:

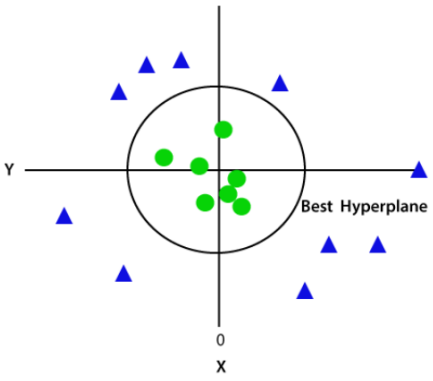


SVM can be of two types:

Linear SVM: Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.



Non-linear SVM: Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.



About dataset:

Pins Face Recognition

This dataset from Kaggle, the images has been collected from Pinterest and cropped. There are 105 celebrities and 17,534 faces.

The images are split into 105 directories, each directory represents a person.
We split the dataset into 70% Train, 20% Validation and 10% Test. Then converting them into RGB, then into np arrays to be ready for MTCNN stage.

Code Implementation:

1 – MTCNN

We imported mtcnn model as MTCNN & other necessary functions for face detection.

```
[ ]  # confirm mtcnn was installed correctly
     import mtcnn
     # print version
     print(mtcnn.__version__)

     0.1.0
```

```
[ ]  # function for face detection with mtcnn
     from PIL import Image
     from numpy import asarray
     from mtcnn.mtcnn import MTCNN
```

As written extract function takes a path of image that we want to extract face from it and size of required face image is a parameter.

after that we are taking an object from MTCNN model to use face detector then passing our image to it and save the result in array called results

that array contains a dictionary for each face in image, so we are tacking only first dictionary index [0] and we need only face that have key “box”.

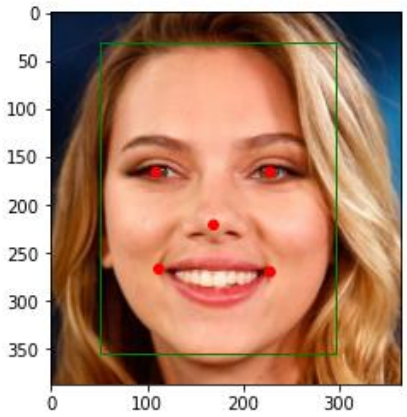
now only thing remining is cropping that face by its coordinates and store it in array called face, then resizing it and converting it to NumPy array then return it.

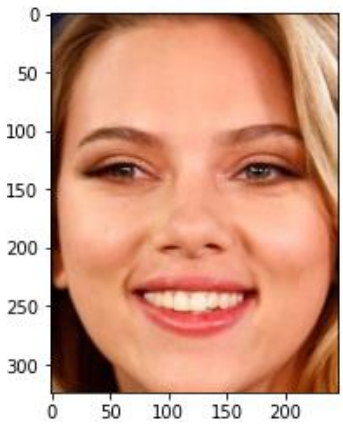
The output size (160,160)

```
[ ]  # extract a single face from a given photograph
     filename = '/content/drive/MyDrive/Selected_Project/Image1.jpg'
     def extract_face(filename, required_size=(160, 160)):
       # load image from file
       image = Image.open(filename)
       # convert to RGB, if needed
       image = image.convert('RGB')
       # convert to array
       pixels = asarray(image)
       # create the detector, using default weights
       detector = MTCNN()
       # detect faces in the image
       results = detector.detect_faces(pixels)
       # extract the bounding box from the first face
       x1, y1, width, height = results[0]['box']
       # bug fix
       x1, y1 = abs(x1), abs(y1)
       x2, y2 = x1 + width, y1 + height
       # extract the face
       face = pixels[y1:y2, x1:x2]
       # resize pixels to the model size
       image = Image.fromarray(face)
       image = image.resize(required_size)
       face_array = asarray(image)
       return face_array

     # load the photo and extract the face
     pixels = extract_face('/content/drive/MyDrive/Selected/Image1.jpg')
```

Example for MTCNN output:





The load_faces() function below will load all of the faces into a list for a given directory.

```
# load images and extract faces for all images in a directory
def load_faces(directory):
  faces = list()
  # enumerate files
  for filename in listdir(directory):
    # path
    path = directory + filename
    # get face
    try:
      face = extract_face(path)
      faces.append(face)
    except:
      continue
  return faces

# load a dataset that contains one subdir for each class that in turn contains images
def load_dataset(directory):
  X, y = list(), list()
  # enumerate folders, on per class
  for subdir in listdir(directory):
    # path
    path = directory + subdir + '/'
    # skip any files that might be in the dir
    if not isdir(path):
      continue
```

Then We can call the load_faces() function for each subdirectory in the ‘train‘ or ‘val‘ folders. Each face has one label, which we can take from the directory name.

Then we call this function for the ‘train’ and ‘val’ folders to load all of the data, then save the results in a single compressed NumPy array file via the savez_compressed() function.

```
# load train dataset
trainX, trainy = load_dataset('/content/drive/MyDrive/Selected/datasete_split/train/')
```

```
# load test dataset
testX, testy = load_dataset('/content/drive/MyDrive/Selected/datasete_split/val/')
print(testX.shape, testy.shape)
# save arrays to one file in compressed format
savez_compressed('pins-faces-dataset.npz', trainX, trainy, testX, testy)
```

2 – FaceNet

First we prepared the image to meet the expectations of the FaceNet model, This specific implementation of the FaceNet model expects that the pixel values are standardized.

```
# get the face embedding for one face
def get_embedding(model, face_pixels):
  # scale pixel values
  face_pixels = face_pixels.astype('float32')
  # standardize pixel values across channels (global)
  mean, std = face_pixels.mean(), face_pixels.std()
  face_pixels = (face_pixels - mean) / std
  # transform face into one sample
  samples = expand_dims(face_pixels, axis=0)
  # make prediction to get embedding
  yhat = model.predict(samples)
  return yhat[0]
```

Then we expand the dimensions so that the face array is one sample.

We used the model to make a prediction and extract the embedding vector.

Finally The get_embedding() function defined below implements these behaviors and will return a face embedding given a single image of a face and the loaded FaceNet model.

First, we load our detected faces dataset using the load() NumPy function.

Next, we load our FaceNet model ready for converting faces into face embeddings.

Then, convert each face in train and test into embedding using the get_embedding() function and compressed them via the savez_compressed() function.

```
# load the face dataset
data = load('/content/drive/MyDrive/Selected_Project/pins-faces-dataset.npz')
trainX, trainy, testX, testy = data['arr_0'], data['arr_1'], data['arr_2'], data['arr_3']
print('Loaded: ', trainX.shape, trainy.shape, testX.shape, testy.shape)
# load the facenet model
model = load_model('/content/drive/MyDrive/Selected_Project/facenet_keras.h5')
print('Loaded Model')
# convert each face in the train set to an embedding
newTrainX = list()
for face_pixels in trainX:
  embedding = get_embedding(model, face_pixels)
  newTrainX.append(embedding)
newTrainX = asarray(newTrainX)
print(newTrainX.shape)
# convert each face in the test set to an embedding
newTestX = list()
for face_pixels in testX:
  embedding = get_embedding(model, face_pixels)
  newTestX.append(embedding)
newTestX = asarray(newTestX)
print(newTestX.shape)
# save arrays to one file in compressed format
savez_compressed('pins-faces-embeddings.npz', newTrainX, trainy, newTestX, testy)
```

3 – Linear Support Vector Machine (SVM)

Now we will use Linear Support Vector Machine (SVM) for Face classification because the method is very effective at separating the face embedding vectors. We fit a linear SVM to the training data using the SVC class in scikit-learn and setting the ‘kernel‘ attribute to ‘linear‘. We also set the ‘probability‘ to ‘True‘.

First, we normalize the face embedding vector because the vectors are compared to each other using a distance metric.

By normalization we means scaling the values until the length or magnitude of the vectors is 1 or unit length, we achieved that using the Normalizer class in scikit-learn.

Next, we converted the string target variables for each celebrity name to be an integers via the LabelEncoder class in scikit-learn.

```
# develop a classifier for the Pins Faces Dataset
from numpy import load
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import Normalizer
from sklearn.svm import SVC

# load dataset
data = load('/content/drive/MyDrive/Selected/pins-faces-embeddings.npz')
trainX, trainy, testX, testy = data['arr_0'], data['arr_1'], data['arr_2'], data['arr_3']
print('Dataset: train=%d, test=%d' % (trainX.shape[0], testX.shape[0]))
# normalize input vectors
in_encoder = Normalizer(norm='l2')
trainX = in_encoder.transform(trainX)
testX = in_encoder.transform(testX)
# label encode targets
out_encoder = LabelEncoder()
out_encoder.fit(trainy)
trainy = out_encoder.transform(trainy)
testy = out_encoder.transform(testy)
# fit model
model = SVC(kernel='linear', probability=True)
model.fit(trainX, trainy)
# predict
yhat_train = model.predict(trainX)
yhat_test = model.predict(testX)
# score
score_train = accuracy_score(trainy, yhat_train)
score_test = accuracy_score(testy, yhat_test)
# summarize
print('Accuracy: train=%.3f, test=%.3f' % (score_train*100, score_test*100))
```

we evaluate the model by using the fit model to make a prediction for each example in the train and test datasets and then calculating the classification accuracy.

Results:

The paper achieves classification accuracy 92.86%±0.12 on LFW and using NN3 for classification

We achieve Classification accuracy on training 99.598% and on testing 98.231% on Pins Face Recognition dataset and using SVM for classification.

```
Dataset: train=12185, test=3449
Accuracy: train=99.598, test=98.231
```