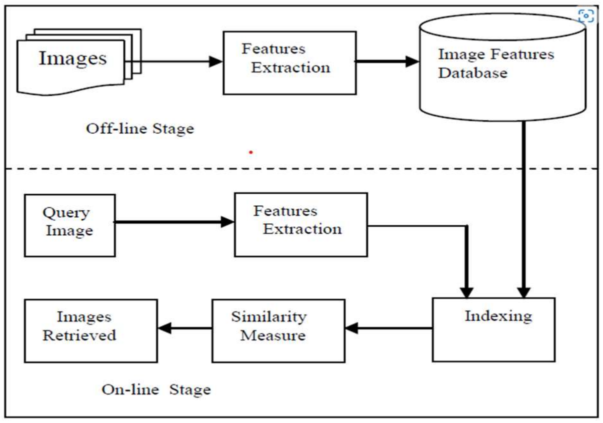
Team: Pothabolu Prasuna Kumari, Nancy Divya JeyaKumar

Reverse Image Search

1. Image Search Engine Baseline:

- Image search Engine gives similar images of given image. We use **content-based image retrieval** concept to build image search engine.
- **Content-based image retrieval (CBIR)** is a system for retrieving relevant images based on a given image. The system consists of an image query and an image database. It works as below
  - Extract features of all images and query image
  - System will calculate similarities between the query with all images on the database.
  - System will retrieve all the images that have a great similarity with the query

- **Content-based image retrieval (CBIR) flow:**



- **Search Engine Baseline Model Implementation:**
  - Download the dataset: The images dataset is downloaded from LFW Face Database : Main (umass.edu).
    Note: Since the dataset is huge and taking long computation time. We used subset (1508 images from this set
  - We use convolutional neural network (CNN) for extracting images features for baseline model
  - Insert the query image (we select image randomly) and extract its features using CNN
  - Calculate the similarities of query image with all images using KNN algorithm. This algorithm calculates distances (Euclidean Distance) between query image features with all images features

$$dist(q, img) = ||q - img||_2 = \sqrt{\sum_{i=1}^{n}(q_i - img_i)^2}$$

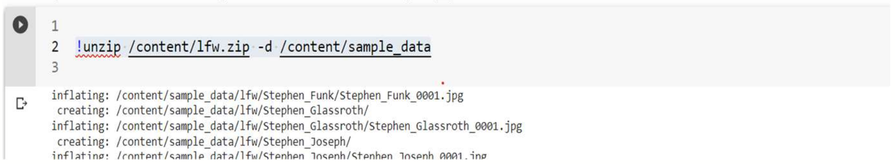
Where:

- q = the query
- img = the image
- n = the number of feature vector element
- i = the position of the vector.

  - Retrieve the most similar result: we get k images that has smaller distances to query image to get similar images.

- **Image search Engine baseline model implementation using python (using google colab):**
  Note: we used same query image for all models to show similar objects using different model
  - Uploaded images dataset lfw.zip to /content/

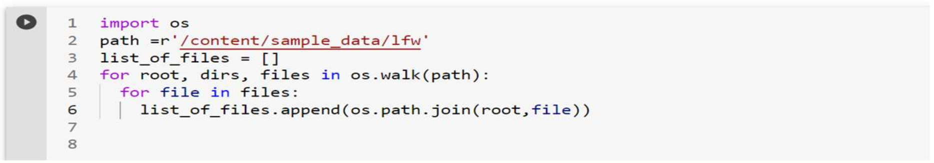
- Unzipped lfw.zip to /content/sample_data using below command
  !unzip /content/lfw.zip -d /content/sample_data



```
!unzip /content/lfw.zip -d /content/sample_data
```

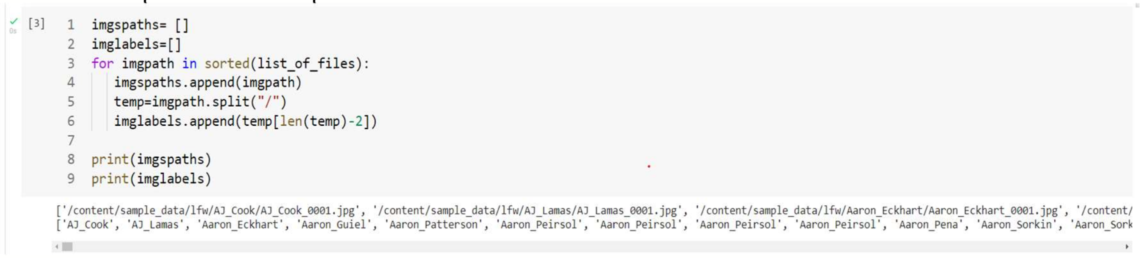
```
inflating: /content/sample_data/lfw/Stephen_Funk/Stephen_Funk_0001.jpg
 creating: /content/sample_data/lfw/Stephen_Glassroth/
inflating: /content/sample_data/lfw/Stephen_Glassroth/Stephen_Glassroth_0001.jpg
 creating: /content/sample_data/lfw/Stephen_Joseph/
inflating: /content/sample_data/lfw/Stephen_Joseph/Stephen_Joseph_0001.jpg
```

- Created files path list



```
import os
path =r'/content/sample_data/lfw'
list_of_files = []
for root, dirs, files in os.walk(path):
  for file in files:
    list_of_files.append(os.path.join(root,file))
```

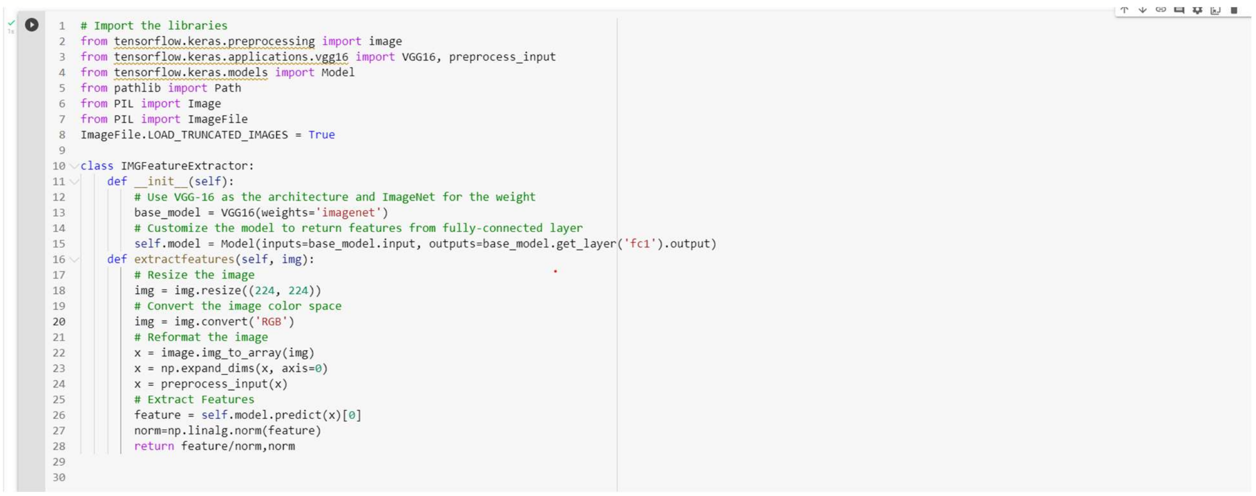
- Created files path list and respective labels in sorted order



```
imgspaths= []
imglabels=[]
for imgpath in sorted(list_of_files):
  imgspaths.append(imgpath)
  temp=imgpath.split("/")
  imglabels.append(temp[len(temp)-2])

print(imgspaths)
print(imglabels)
```

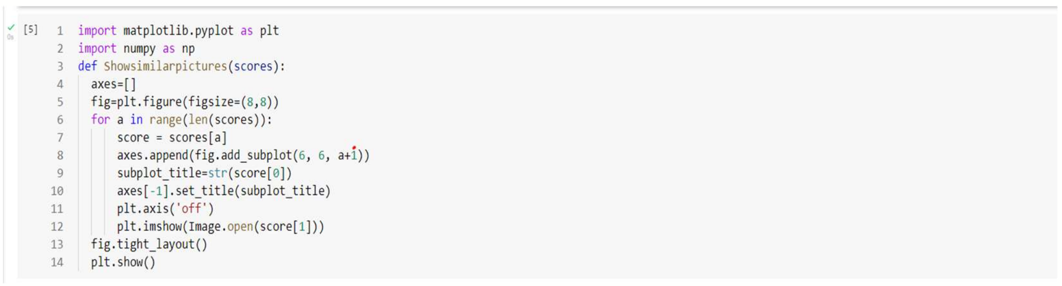
```
['/content/sample_data/lfw/AJ_Cook/AJ_Cook_0001.jpg', '/content/sample_data/lfw/AJ_Lamas/AJ_Lamas_0001.jpg', '/content/sample_data/lfw/Aaron_Eckhart/Aaron_Eckhart_0001.jpg', '/content/
['AJ_Cook', 'AJ_Lamas', 'Aaron_Eckhart', 'Aaron_Guiel', 'Aaron_Patterson', 'Aaron_Peirsol', 'Aaron_Peirsol', 'Aaron_Peirsol', 'Aaron_Peirsol', 'Aaron_Pena', 'Aaron_Sorkin', 'Aaron_Sork
```

- Extract features of image: resized image with (224,224) pixels and converted image to color. Used baseline model using VGG16. Extracted features array from fully connected layer.



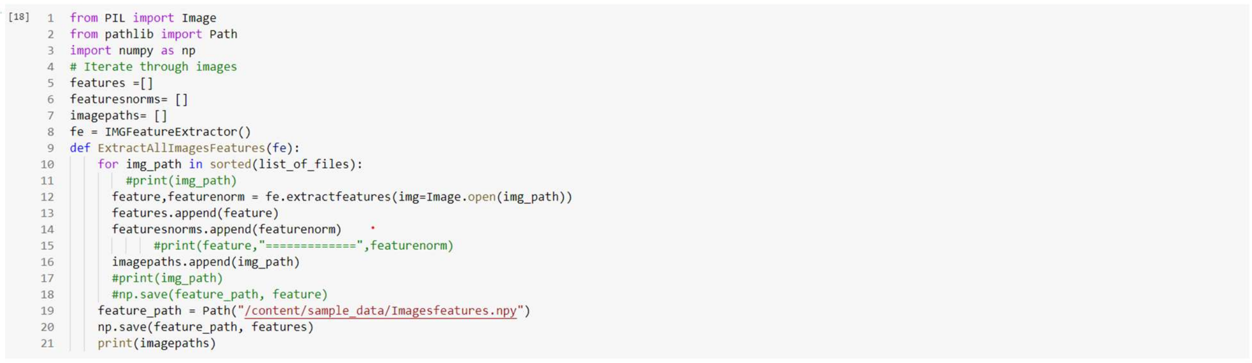
```
# Import the libraries
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.models import Model
from pathlib import Path
from PIL import Image
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True

class IMGFeatureExtractor:
    def __init__(self):
        # Use VGG-16 as the architecture and ImageNet for the weight
        base_model = VGG16(weights='imagenet')
        # Customize the model to return features from fully-connected layer
        self.model = Model(inputs=base_model.input, outputs=base_model.get_layer('fc1').output)
    def extractfeatures(self, img):
        # Resize the image
        img = img.resize((224, 224))
        # Convert the image color space
        img = img.convert('RGB')
        # Reformat the image
        x = image.img_to_array(img)
        x = np.expand_dims(x, axis=0)
        x = preprocess_input(x)
        # Extract Features
        feature = self.model.predict(x)[0]
        norm=np.linalg.norm(feature)
        return feature/norm,norm
```

- Below function is to plot similar images:



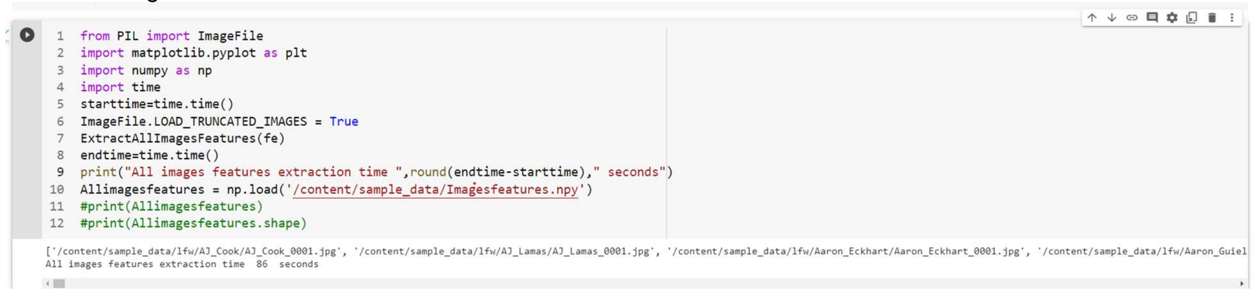
```
import matplotlib.pyplot as plt
import numpy as np
def Showsimilarpictures(scores):
  axes=[]
  fig=plt.figure(figsize=(8,8))
  for a in range(len(scores)):
      score = scores[a]
      axes.append(fig.add_subplot(6, 6, a+1))
      subplot_title=str(score[0])
      axes[-1].set_title(subplot_title)
      plt.axis('off')
      plt.imshow(Image.open(score[1]))
  fig.tight_layout()
  plt.show()
```

- Extract all images features and save in array. It took more than 30 min without GPU



```
[18]
from PIL import Image
from pathlib import Path
import numpy as np
# Iterate through images
features =[]
featuresnorms= []
imagepaths= []
fe = IMGFeatureExtractor()
def ExtractAllImagesFeatures(fe):
    for img_path in sorted(list_of_files):
        #print(img_path)
      feature,featurenorm = fe.extractfeatures(img=Image.open(img_path))
      features.append(feature)
      featuresnorms.append(featurenorm)
          #print(feature,"=============",featurenorm)
      imagepaths.append(img_path)
      #print(img_path)
      #np.save(feature_path, feature)
    feature_path = Path("/content/sample_data/Imagesfeatures.npy")
    np.save(feature_path, features)
    print(imagepaths)
```

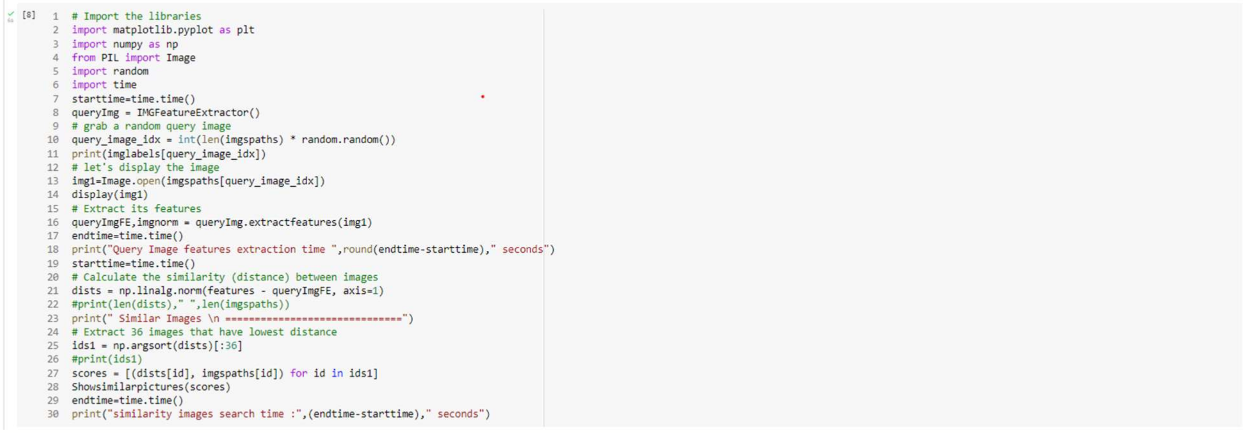
- Saved all images features numpy array in npy file for future use. The features file can be used for search, instead of extracting features again. All images features extraction (1508 images) took 86 seconds using GPU.



```
from PIL import ImageFile
import matplotlib.pyplot as plt
import numpy as np
import time
starttime=time.time()
ImageFile.LOAD_TRUNCATED_IMAGES = True
ExtractAllImagesFeatures(fe)
endtime=time.time()
print("All images features extraction time ",round(endtime-starttime)," seconds")
Allimagesfeatures = np.load('/content/sample_data/Imagesfeatures.npy')
#print(Allimagesfeatures)
#print(Allimagesfeatures.shape)
```

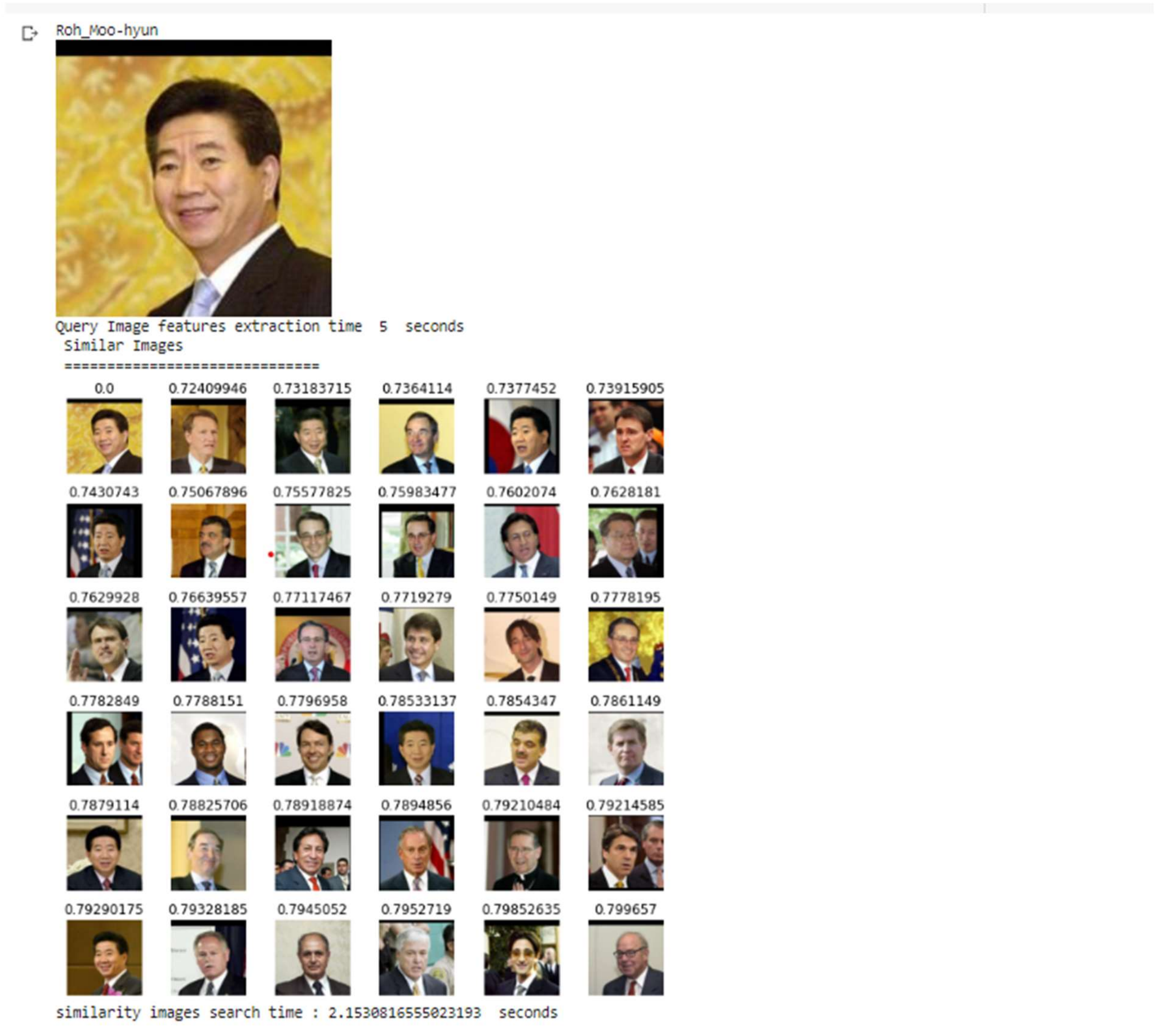
```
['/content/sample_data/lfw/AJ_Cook/AJ_Cook_0001.jpg', '/content/sample_data/lfw/AJ_Lamas/AJ_Lamas_0001.jpg', '/content/sample_data/lfw/Aaron_Eckhart/Aaron_Eckhart_0001.jpg', '/content/sample_data/lfw/Aaron_Guiel
All images features extraction time  86  seconds
```

- Loaded random image and extracted it's features. It took 2 seconds. Found distance between query image features and all images features. Considered 25 images with shortest distance to query image as similar images(it is using KNN and L2 norm). Finding similar images took 1.57 seconds
  Below is code :



```
[8]
# Import the libraries
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
import random
import time
starttime=time.time()
queryImg = IMGFeatureExtractor()
# grab a random query image
query_image_idx = int(len(imgspaths) * random.random())
print(imglabels[query_image_idx])
# let's display the image
img1=Image.open(imgspaths[query_image_idx])
display(img1)
# Extract its features
queryImgFE,imgnorm = queryImg.extractfeatures(img1)
endtime=time.time()
print("Query Image features extraction time ",round(endtime-starttime)," seconds")
starttime=time.time()
# Calculate the similarity (distance) between images
dists = np.linalg.norm(features - queryImgFE, axis=1)
#print(len(dists)," ",len(imgspaths))
print(" Similar Images \n ==============================")
# Extract 36 images that have lowest distance
ids1 = np.argsort(dists)[:36]
#print(ids1)
scores = [(dists[id], imgspaths[id]) for id in ids1]
Showsimilarpictures(scores)
endtime=time.time()
print("similarity images search time :",(endtime-starttime)," seconds")
```

- Query image and similar images:



```
Roh_Moo-hyun
```

```
Query Image features extraction time  5  seconds
 Similar Images
 ==============================
```

```
similarity images search time : 2.1530816555023193  seconds
```

2. **Reverse Image Search Improvement using SKlearn,PCA, Facenet:**

➢ **Image search Engine model implementation using Sklearn library NearestNeighbors:**

- All images features will be fitted to NearestNeighbors model with k value. The neighbors function will give K shortest distances between queryimage features and other images. The function provides K images indexes too.
  This mode took 1.06 seconds without features extraction.
  Note: we used features extracted , same query image for this model too. It is to show all models output is same
- Below is the code

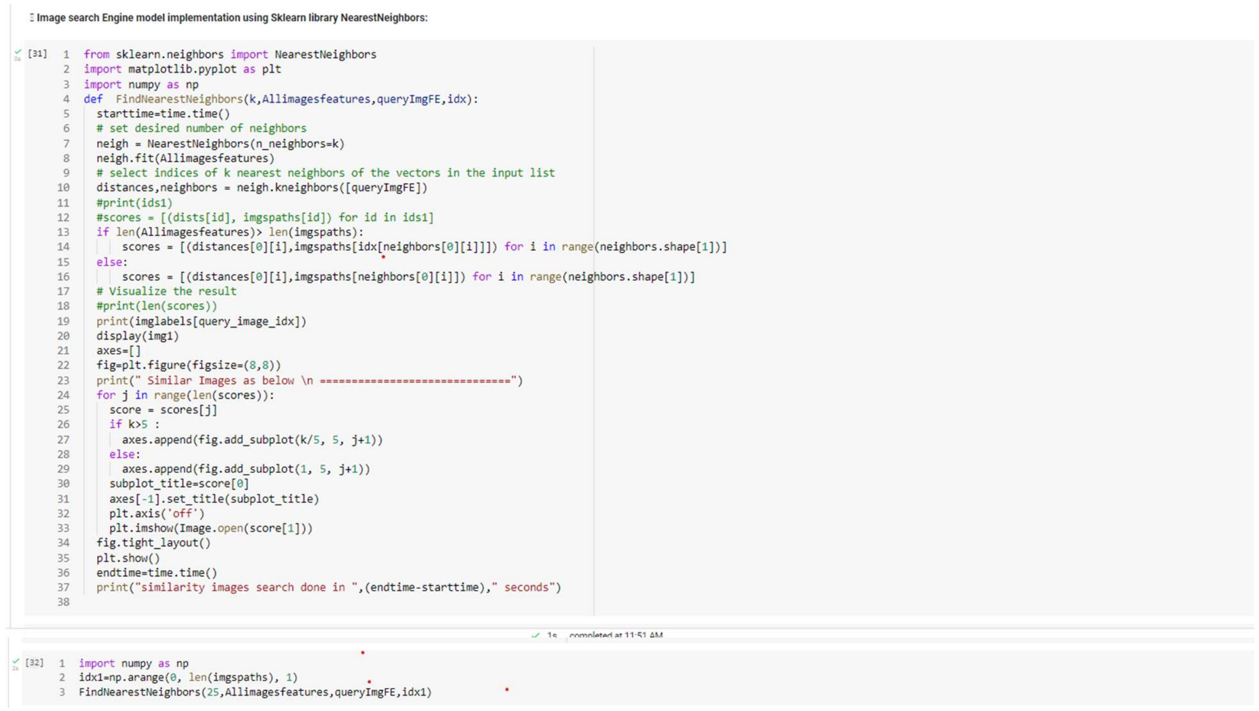


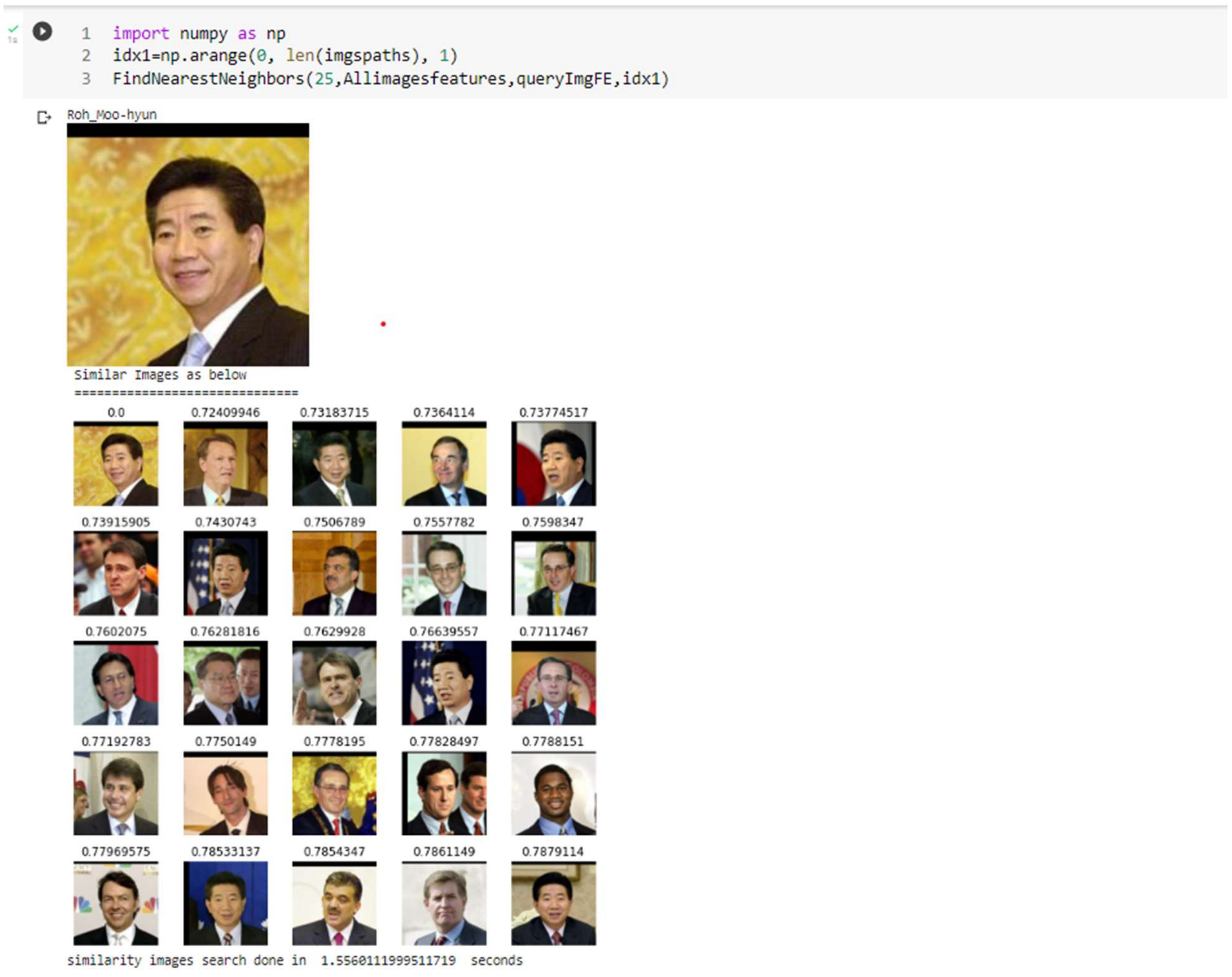
Image search Engine model implementation using Sklearn library NearestNeighbors:

[31]
```python
from sklearn.neighbors import NearestNeighbors
import matplotlib.pyplot as plt
import numpy as np
def  FindNearestNeighbors(k,Allimagesfeatures,queryImgFE,idx):
  starttime=time.time()
  # set desired number of neighbors
  neigh = NearestNeighbors(n_neighbors=k)
  neigh.fit(Allimagesfeatures)
  # select indices of k nearest neighbors of the vectors in the input list
  distances,neighbors = neigh.kneighbors([queryImgFE])
  #print(ids1)
  #scores = [(dists[id], imgspaths[id]) for id in ids1]
  if len(Allimagesfeatures)> len(imgspaths):
      scores = [(distances[0][i],imgspaths[idx[neighbors[0][i]]]) for i in range(neighbors.shape[1])]
  else:
      scores = [(distances[0][i],imgspaths[neighbors[0][i]]) for i in range(neighbors.shape[1])]
  # Visualize the result
  #print(len(scores))
  print(imglabels[query_image_idx])
  display(img1)
  axes=[]
  fig=plt.figure(figsize=(8,8))
  print(" Similar Images as below \n ==============================")
  for j in range(len(scores)):
    score = scores[j]
    if k>5 :
      axes.append(fig.add_subplot(k/5, 5, j+1))
    else:
      axes.append(fig.add_subplot(1, 5, j+1))
    subplot_title=score[0]
    axes[-1].set_title(subplot_title)
    plt.axis('off')
    plt.imshow(Image.open(score[1]))
  fig.tight_layout()
  plt.show()
  endtime=time.time()
  print("similarity images search done in ",(endtime-starttime)," seconds")

```

✓ 1s completed at 11:51 AM

[32]
```python
import numpy as np
idx1=np.arange(0, len(imgspaths), 1)
FindNearestNeighbors(25,Allimagesfeatures,queryImgFE,idx1)
```

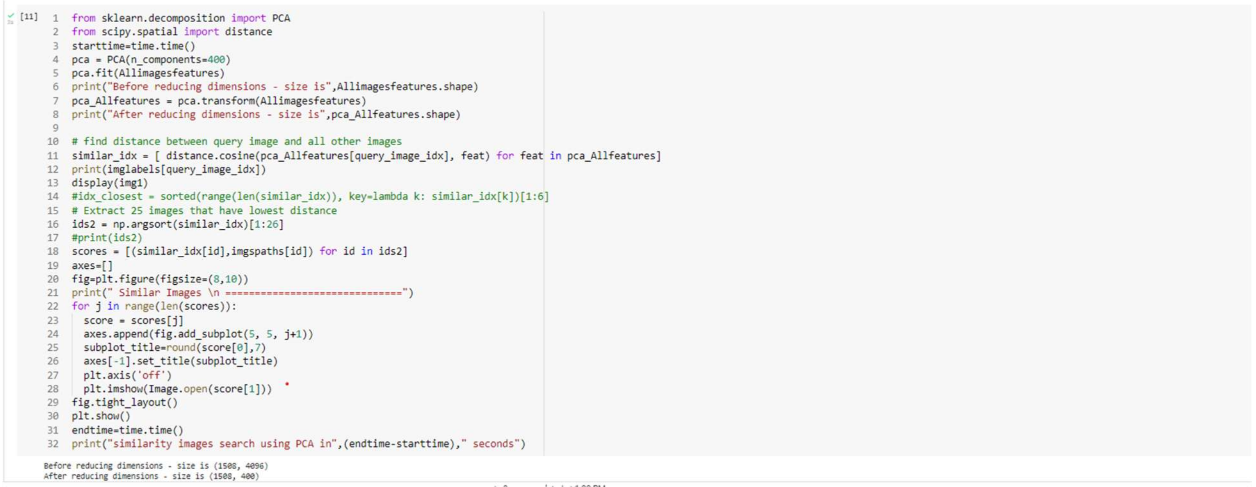
- Below are similar objects are found using search engine with SKlearn.



```python
import numpy as np
idx1=np.arange(0, len(imgspaths), 1)
FindNearestNeighbors(25,Allimagesfeatures,queryImgFE,idx1)
```

```
Roh_Moo-hyun
Similar Images as below
==============================
    0.0      0.72409946  0.73183715  0.7364114   0.73774517
0.73915905   0.7430743   0.7506789   0.7557782   0.7598347
0.7602075    0.76281816  0.7629928   0.76639557  0.77117467
0.77192783   0.7750149   0.7778195   0.77828497  0.7788151
0.77969575   0.78533137  0.7854347   0.7861149   0.7879114
similarity images search done in  1.5560111999511719  seconds
```

- **Image search Engine model implementation using PCA(Principal component analysis):**
  - **PCA (Dimensionality deduction) is algorithm used for dimensionality deduction**
  - The pca object stores the actual transformation matrix of all imagesfeatures which was fit in the pca object. We can now use it to transform any original feature vector (of length 4096 colums) into a reduced 700-dimensional feature vector
  - The assumption we can now make is that two images which have similar content, should produce similar feature vectors. we use same query image and compute a measurement of the dissimilarity (or distance) of that image's PCA feature vector to every other image's feature vector. The dissimilarity metric we use is cosine distance.
  - The list similar_idx contains the image's similarity to every other one. We can sort that list and find the indexes of the most similar images. The next cell will sort them, and then find the most similar items, and return the indexes 25 most similar images. Notice we take from indexes 1:26 rather than 0:25 because the most similar image to the query image, will trivially be the query image itself, since it is included in the distance calculation
  - This model took total time 3.7 seconds using extracted features
  - Below is the code

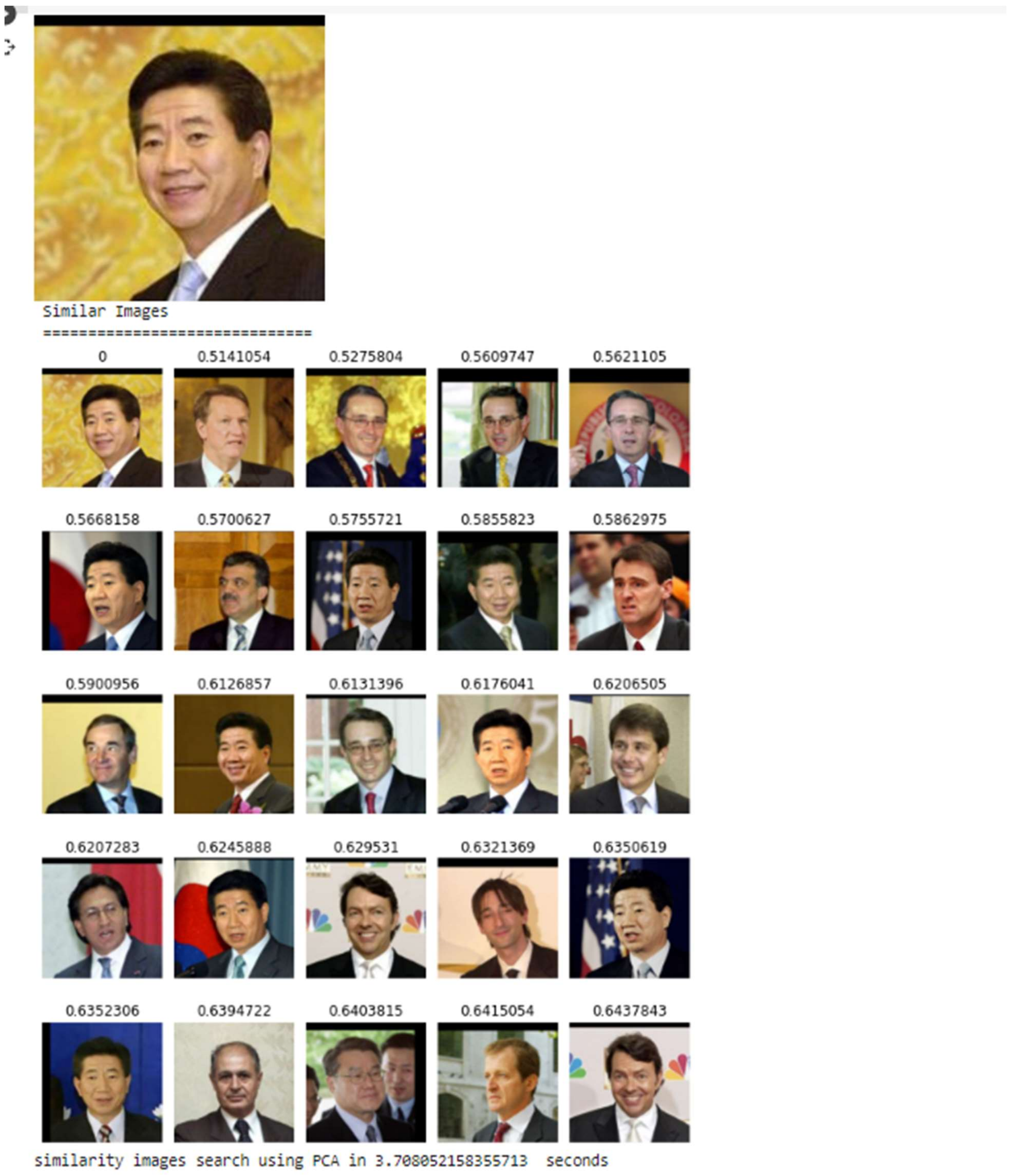


[11]

```python
from sklearn.decomposition import PCA
from scipy.spatial import distance
starttime=time.time()
pca = PCA(n_components=400)
pca.fit(Allimagesfeatures)
print("Before reducing dimensions - size is",Allimagesfeatures.shape)
pca_Allfeatures = pca.transform(Allimagesfeatures)
print("After reducing dimensions - size is",pca_Allfeatures.shape)

# find distance between query image and all other images
similar_idx = [ distance.cosine(pca_Allfeatures[query_image_idx], feat) for feat in pca_Allfeatures]
print(imglabels[query_image_idx])
display(img1)
#idx_closest = sorted(range(len(similar_idx)), key=lambda k: similar_idx[k])[1:6]
# Extract 25 images that have lowest distance
ids2 = np.argsort(similar_idx)[1:26]
#print(ids2)
scores = [(similar_idx[id],imgspaths[id]) for id in ids2]
axes=[]
fig=plt.figure(figsize=(8,10))
print(" Similar Images \n ==============================")
for j in range(len(scores)):
  score = scores[j]
  axes.append(fig.add_subplot(5, 5, j+1))
  subplot_title=round(score[0],7)
  axes[-1].set_title(subplot_title)
  plt.axis('off')
  plt.imshow(Image.open(score[1]))
fig.tight_layout()
plt.show()
endtime=time.time()
print("similarity images search using PCA in",(endtime-starttime)," seconds")
```

```
Before reducing dimensions - size is (1508, 4096)
After reducing dimensions - size is (1508, 400)
```

- Below are similar objects are found using search engine with PCA.

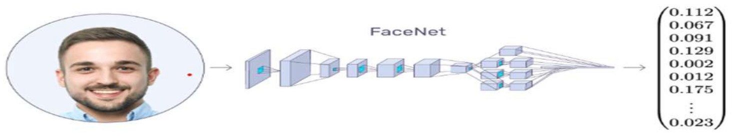


```
Similar Images
===============================
     0          0.5141054   0.5275804   0.5609747   0.5621105
 0.5668158      0.5700627   0.5755721   0.5855823   0.5862975
 0.5900956      0.6126857   0.6131396   0.6176041   0.6206505
 0.6207283      0.6245888   0.629531    0.6321369   0.6350619
 0.6352306      0.6394722   0.6403815   0.6415054   0.6437843
similarity images search using PCA in 3.708052158355713  seconds
```

- **Image search Engine model implementation using FaceNet,MTCNN and Keras:**
  - To improve image search engine baseline model, we used FaceNet,MTCNN and sklearn nearest neighbors algorithm.

- 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. opensource implementations of the model and pretrained modes are available.

- 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. This model takes RGB images of 160×160 and generates an embedding of size 128 for an image.



FaceNet takes an image of a face as input and outputs the embedding vector.

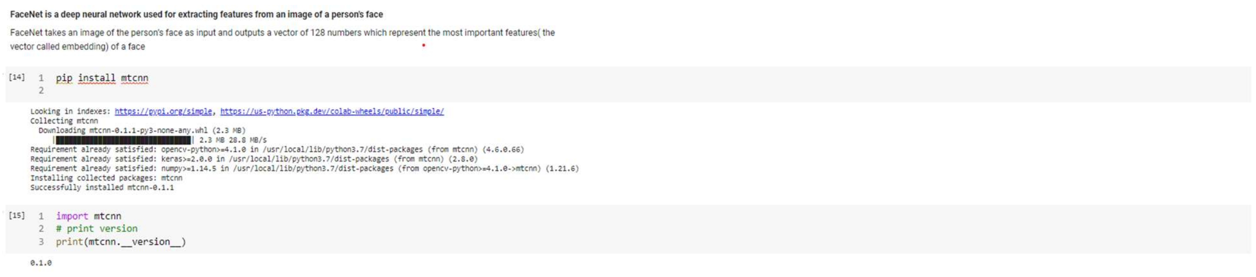
- Multi-task Cascaded Convolutional Networks (MTCNN) is a framework developed as a solution for both face detection and face alignment. The process consists of three stages of convolutional networks that are able to recognize faces and landmark location such as eyes, nose, and mouth.

- NearestNeighbors algorithm to find nearest neighbors (images)

➢ Below are implementation steps of image search engine using Facenet,MTCNN and KNN:
- Upload image dataset
- Detect all faces from all images using MTCNN.
- Load query image (we select image randomly)
- Detect all faces from query image
- Load Facenet pretrained model
- Extract features from all faces of dataset images using Facenet
- Extract features from all faces of query image
- Calculate the similarities of query image with all images using KNN algorithm. This algorithm calculates distances (Euclidean Distance) between query image features with all images features
- Retrieve the most similar result: we get k images that has smaller distances to query image to get similar images

➢ Below are implementation steps of image search engine using python code:

- Please install mtcnn and verify installation by getting version number
  Commands:
  Pip install mtcnn
  Import mtcnn
  Print(mtcnn.__version__)



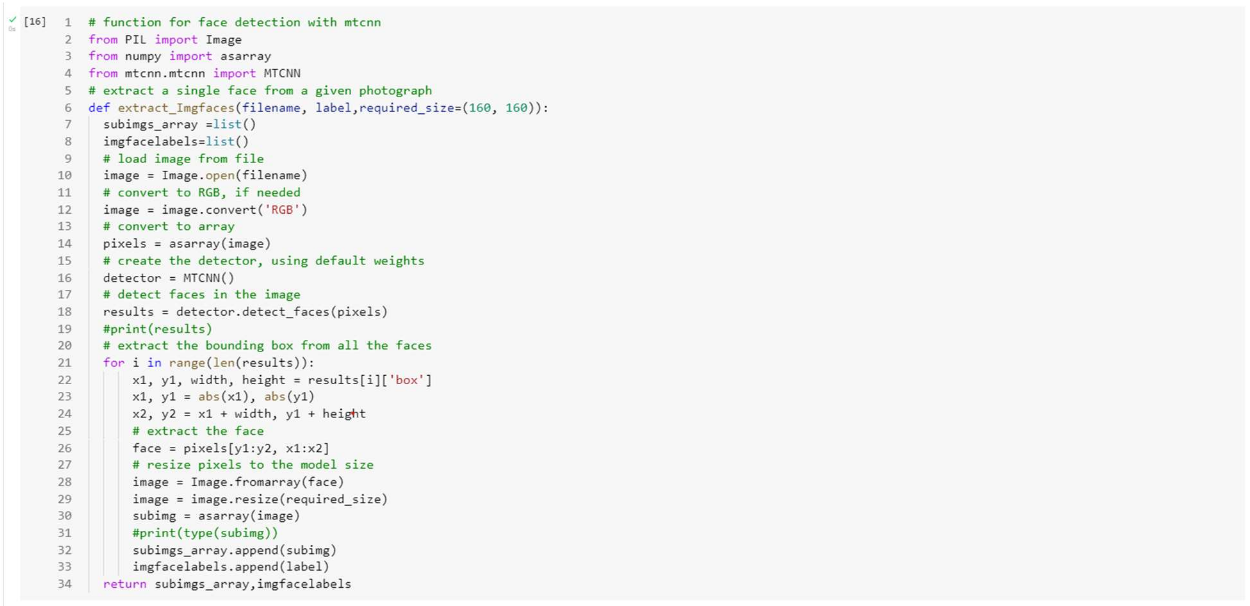
FaceNet is a deep neural network used for extracting features from an image of a person's face

FaceNet takes an image of the person's face as input and outputs a vector of 128 numbers which represent the most important features(the vector called embedding) of a face

```
[14]
pip install mtcnn

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting mtcnn
  Downloading mtcnn-0.1.1-py3-none-any.whl (2.3 MB)
     |████████████████████████████████| 2.3 MB 28.8 MB/s
Requirement already satisfied: opencv-python>=4.1.0 in /usr/local/lib/python3.7/dist-packages (from mtcnn) (4.6.0.66)
Requirement already satisfied: keras>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from mtcnn) (2.8.0)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from opencv-python>=4.1.0->mtcnn) (1.21.6)
Installing collected packages: mtcnn
Successfully installed mtcnn-0.1.1
```

```
[15]
import mtcnn
# print version
print(mtcnn.__version__)

0.1.0
```

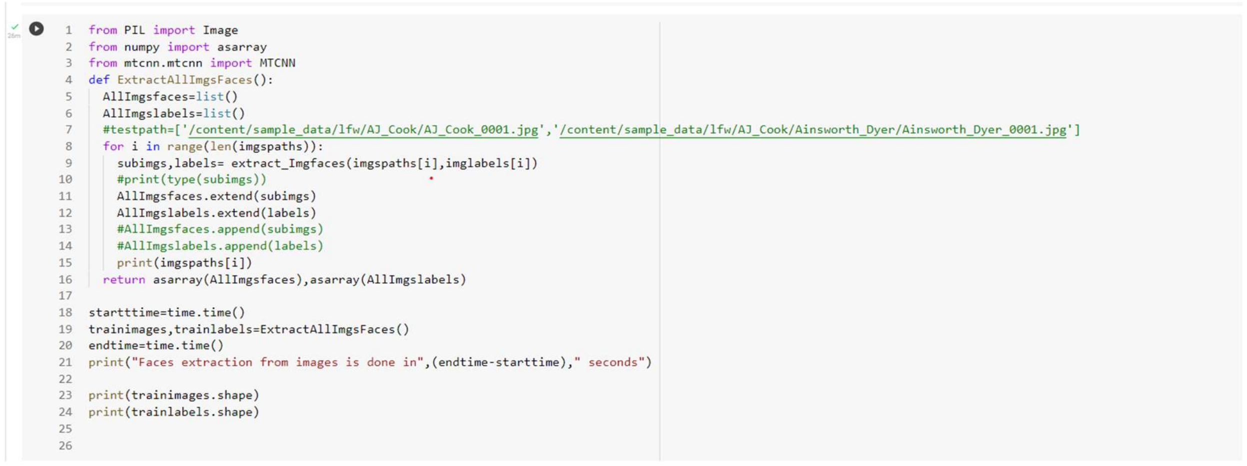
  - Below is the code to extract faces from a image using MTCNN



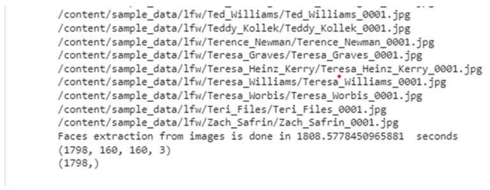
```
[16]
# function for face detection with mtcnn
from PIL import Image
from numpy import asarray
from mtcnn.mtcnn import MTCNN
# extract a single face from a given photograph
def extract_Imgfaces(filename, label,required_size=(160, 160)):
  subimgs_array =list()
  imgfacelabels=list()
  # load image 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)
  #print(results)
  # extract the bounding box from all the faces
  for i in range(len(results)):
      x1, y1, width, height = results[i]['box']
      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)
      subimg = asarray(image)
      #print(type(subimg))
      subimgs_array.append(subimg)
      imgfacelabels.append(label)
  return subimgs_array,imgfacelabels
```

- Below is the code to extract faces from all images using MTCNN. It took more than 30 minutes from 1508 images

  Note: the code written to extract all faces from image

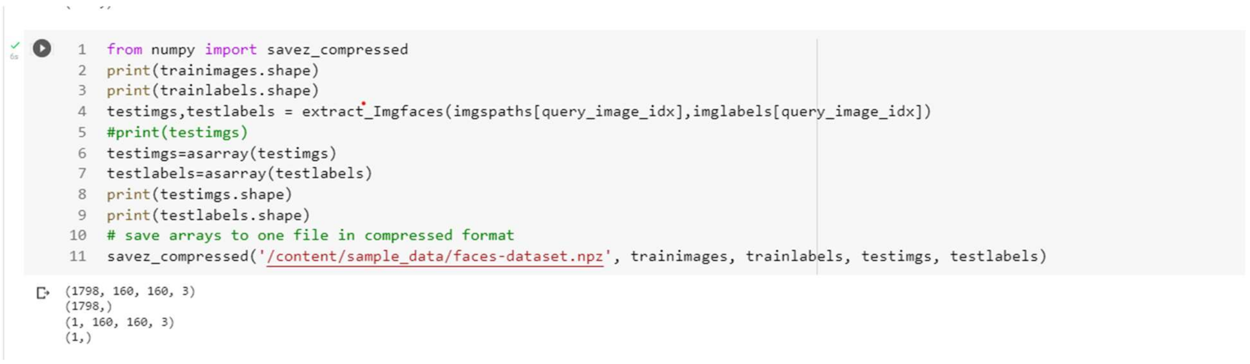


```
from PIL import Image
from numpy import asarray
from mtcnn.mtcnn import MTCNN
def ExtractAllImgsFaces():
  AllImgsfaces=list()
  AllImgslabels=list()
  #testpath=['/content/sample_data/lfw/AJ_Cook/AJ_Cook_0001.jpg','/content/sample_data/lfw/AJ_Cook/Ainsworth_Dyer/Ainsworth_Dyer_0001.jpg']
  for i in range(len(imgspaths)):
    subimgs,labels= extract_Imgfaces(imgspaths[i],imglabels[i])
    #print(type(subimgs))
    AllImgsfaces.extend(subimgs)
    AllImgslabels.extend(labels)
    #AllImgsfaces.append(subimgs)
    #AllImgslabels.append(labels)
    print(imgspaths[i])
  return asarray(AllImgsfaces),asarray(AllImgslabels)

starttime=time.time()
trainimages,trainlabels=ExtractAllImgsFaces()
endtime=time.time()
print("Faces extraction from images is done in",(endtime-starttime)," seconds")

print(trainimages.shape)
print(trainlabels.shape)
```



```
/content/sample_data/lfw/Ted_Williams/Ted_Williams_0001.jpg
/content/sample_data/lfw/Teddy_Kollek/Teddy_Kollek_0001.jpg
/content/sample_data/lfw/Terence_Newman/Terence_Newman_0001.jpg
/content/sample_data/lfw/Teresa_Graves/Teresa_Graves_0001.jpg
/content/sample_data/lfw/Teresa_Heinz_Kerry/Teresa_Heinz_Kerry_0001.jpg
/content/sample_data/lfw/Teresa_Williams/Teresa_Williams_0001.jpg
/content/sample_data/lfw/Teresa_Worbis/Teresa_Worbis_0001.jpg
/content/sample_data/lfw/Teri_Files/Teri_Files_0001.jpg
/content/sample_data/lfw/Zach_Safrin/Zach_Safrin_0001.jpg
Faces extraction from images is done in 1808.5778450965881  seconds
(1798, 160, 160, 3)
(1798,)
```

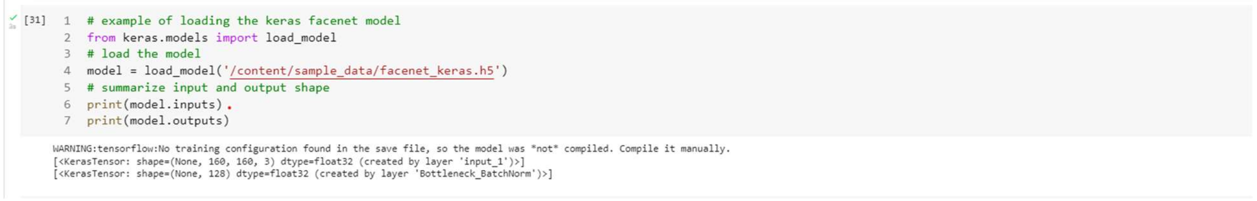
- Below is the code to extract faces from query images using MTCNN. We save extracted faces array and respective images labels at npz file. It is just avoid faces detection again and again



```
from numpy import savez_compressed
print(trainimages.shape)
print(trainlabels.shape)
testimgs,testlabels = extract_Imgfaces(imgspaths[query_image_idx],imglabels[query_image_idx])
#print(testimgs)
testimgs=asarray(testimgs)
testlabels=asarray(testlabels)
print(testimgs.shape)
print(testlabels.shape)
# save arrays to one file in compressed format
savez_compressed('/content/sample_data/faces-dataset.npz', trainimages, trainlabels, testimgs, testlabels)
```

```
(1798, 160, 160, 3)
(1798,)
(1, 160, 160, 3)
(1,)
```

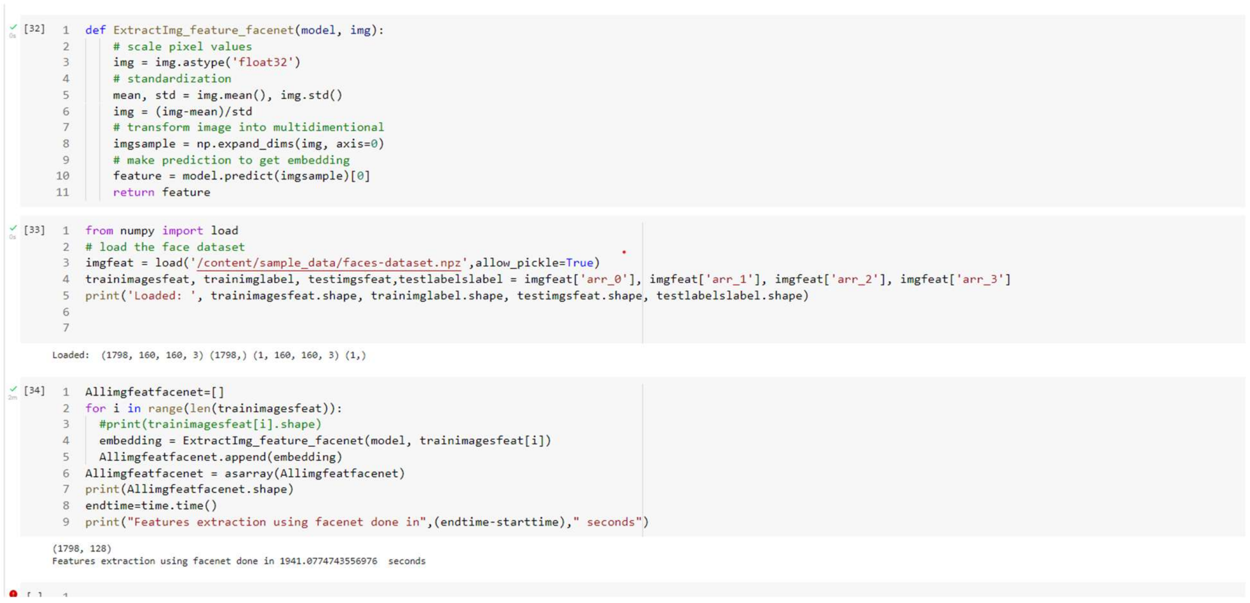
- Below is the code to Load Facenet pretrained model



```
[31]
# example of loading the keras facenet model
from keras.models import load_model
# load the model
model = load_model('/content/sample_data/facenet_keras.h5')
# summarize input and output shape
print(model.inputs)
print(model.outputs)
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model was *not* compiled. Compile it manually.
[<KerasTensor: shape=(None, 160, 160, 3) dtype=float32 (created by layer 'input_1')>]
[<KerasTensor: shape=(None, 128) dtype=float32 (created by layer 'Bottleneck_BatchNorm')>]
```

- Below is the code extract features from a face image of full image dataset. It took more than 30 minutes.
  The features will be 128 array
  Faces array with dimension is 160,160,3 for each face



```
[32]
def ExtractImg_feature_facenet(model, img):
    # scale pixel values
    img = img.astype('float32')
    # standardization
    mean, std = img.mean(), img.std()
    img = (img-mean)/std
    # transform image into multidimentional
    imgsample = np.expand_dims(img, axis=0)
    # make prediction to get embedding
    feature = model.predict(imgsample)[0]
    return feature
```

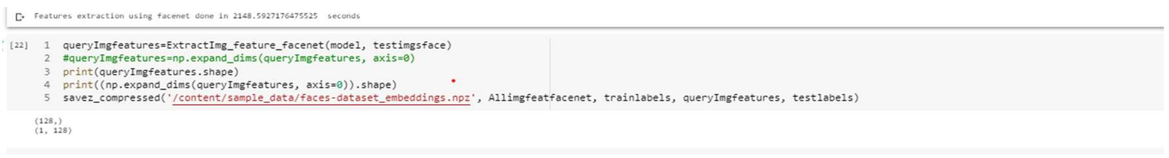
```
[33]
from numpy import load
# load the face dataset
imgfeat = load('/content/sample_data/faces-dataset.npz',allow_pickle=True)
trainimagesfeat, trainimglabel, testimgsfeat,testlabelslabel = imgfeat['arr_0'], imgfeat['arr_1'], imgfeat['arr_2'], imgfeat['arr_3']
print('Loaded: ', trainimagesfeat.shape, trainimglabel.shape, testimgsfeat.shape, testlabelslabel.shape)
```

```
Loaded:  (1798, 160, 160, 3) (1798,) (1, 160, 160, 3) (1,)
```

```
[34]
Allimgfeatfacenet=[]
for i in range(len(trainimagesfeat)):
  #print(trainimagesfeat[i].shape)
  embedding = ExtractImg_feature_facenet(model, trainimagesfeat[i])
  Allimgfeatfacenet.append(embedding)
Allimgfeatfacenet = asarray(Allimgfeatfacenet)
print(Allimgfeatfacenet.shape)
endtime=time.time()
print("Features extraction using facenet done in",(endtime-starttime)," seconds")
```

```
(1798, 128)
Features extraction using facenet done in 1941.0774743556976  seconds
```

- Below is the code save to extracted features,labels of all faces of dataset and query image faces features,labels in npz file for future use.

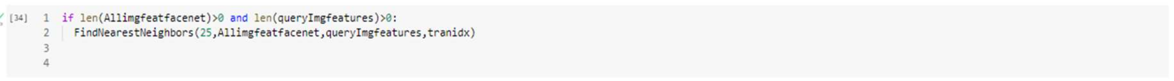


```
Features extraction using facenet done in 2148.5927176475525  seconds
```

```
[22]
queryImgfeatures=ExtractImg_feature_facenet(model, testimgsface)
#queryImgfeatures=np.expand_dims(queryImgfeatures, axis=0)
print(queryImgfeatures.shape)
print((np.expand_dims(queryImgfeatures, axis=0)).shape)
savez_compressed('/content/sample_data/faces-dataset_embeddings.npz', Allimgfeatfacenet, trainlabels, queryImgfeatures, testlabels)
```

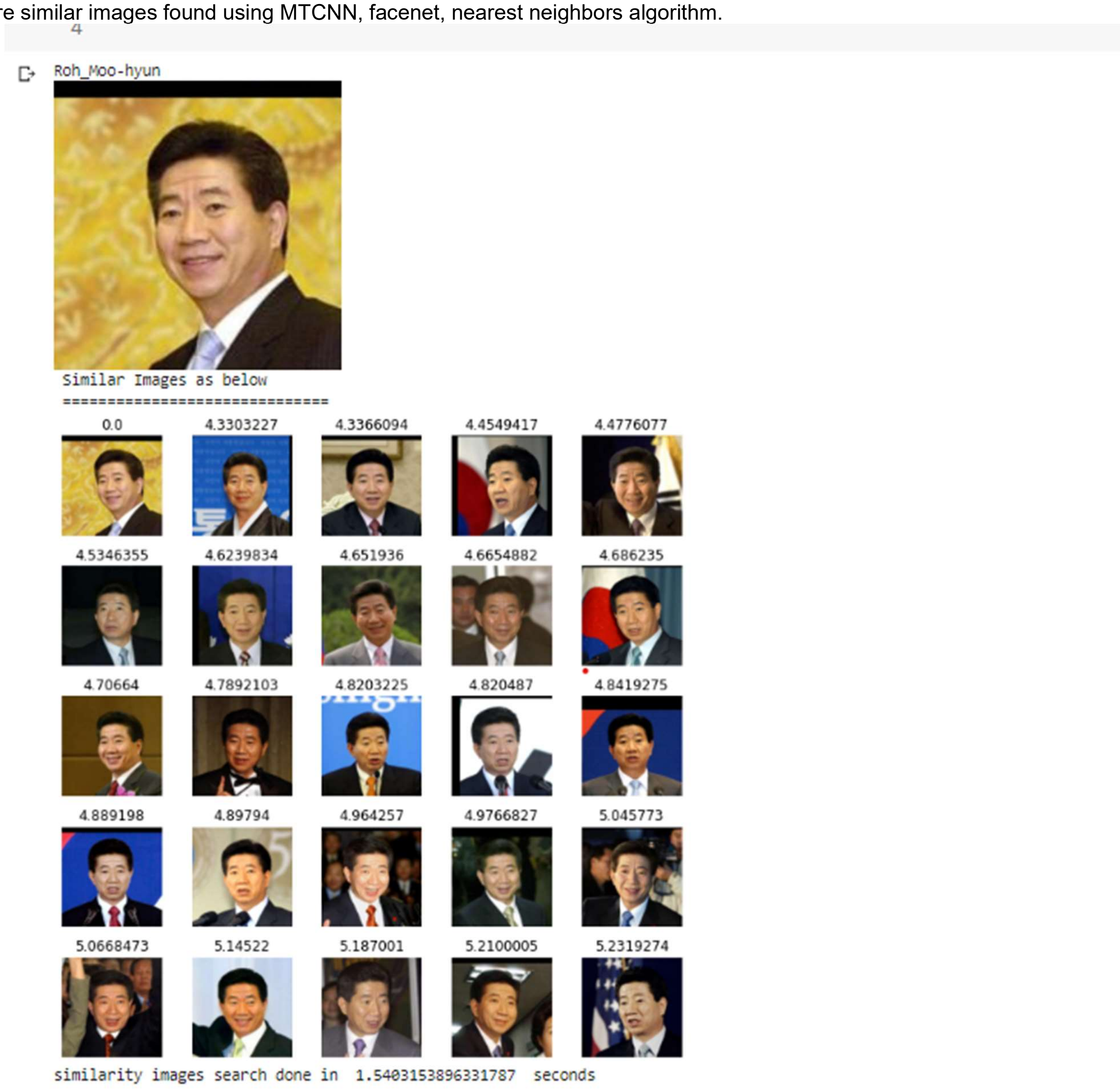
```
(128,)
(1, 128)
```

- We find distance between features(extracted using facenet) to find similar images using nearest neighbors algorithm. It took 1.3 seconds



```
[34]
if len(Allimgfeatfacenet)>0 and len(queryImgfeatures)>0:
  FindNearestNeighbors(25,Allimgfeatfacenet,queryImgfeatures,tranidx)
```

- Below are similar images found using MTCNN, facenet, nearest neighbors algorithm.



```
Roh_Moo-hyun
```

```
Similar Images as below
==============================
```

```
similarity images search done in  1.5403153896331787  seconds
```