CECS 551 – Advance Artificial Intelligence

Assignment 8

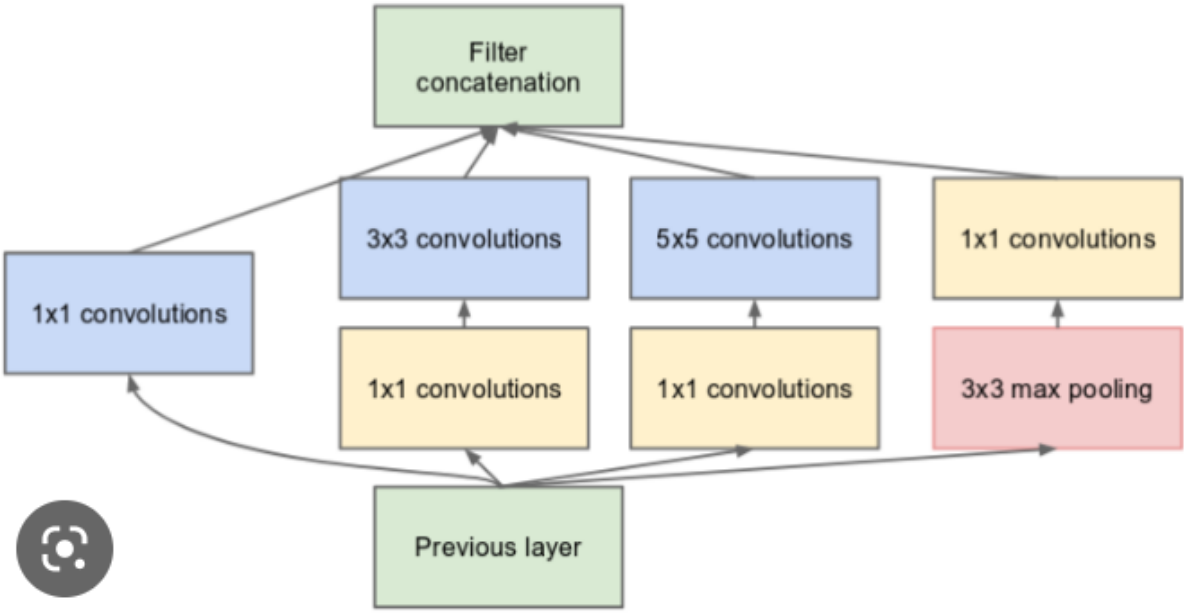
030747585

Satyam Pathak

Facenet:-

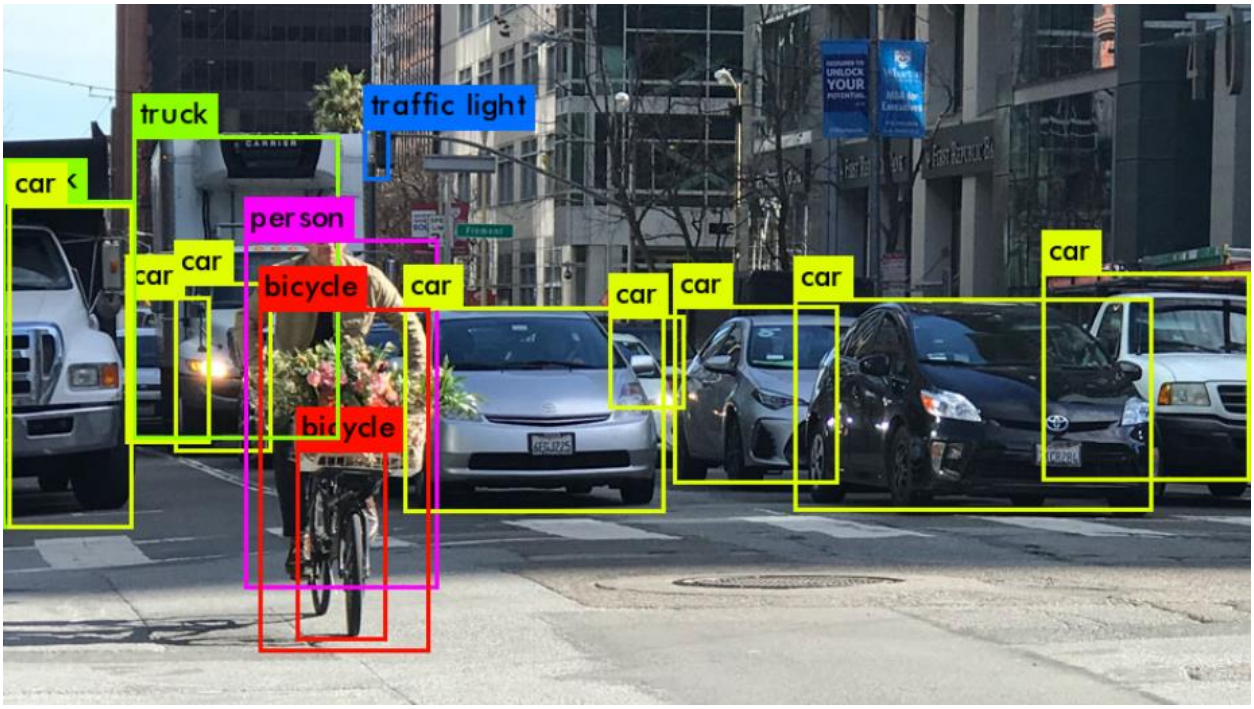
In 2015, Florian Schroff, Dmitry Kalenichenko, and James Philbin, three Google researchers, created FaceNet, a project for facial recognition. To create an embedding from a person's face is the major objective of this research.

Any item may be represented by an embedding as a dense vector. When the cosine distance and Euclidean distance between two faces that are the same are relatively small, this is considered to be a good embedding. Conversely, a Euclidean distance and a cosine similarity that is far away should exist between two embeddings with different sides.



Yolo v3:-

You Only Look Once, Version 3 (YOLOv3) is a real-time object detection system that can recognize particular things in films, live feeds, or still photos. To find an item, the YOLO machine learning technique employs features discovered by a deep convolutional neural network. Joseph Redmon and Ali Farhadi developed YOLO versions 1-3, with version three being a more accurate iteration of the original machine learning method. YOLO's initial version was built in 2016, and version 3, which is the focus of this article, was made two years later in 2018. A better variant of YOLO and YOLOv2 is YOLOv3. The deep learning libraries Keras or OpenCV are used to implement YOLO.



Experiment results:-

Here we can see the 1200 images that are given as tensor. Tensor are image as vectors

```
torch.Size([256, 256, 3])
tensor([[[0.0000, 0.0000, 0.0000,  ..., 0.0039, 0.0039, 0.0039],
         [0.0000, 0.0000, 0.0000,  ..., 0.0039, 0.0039, 0.0039],
         [0.0000, 0.0000, 0.0000,  ..., 0.0039, 0.0039, 0.0039],
         ...,
         [0.0000, 0.0000, 0.0000,  ..., 0.2353, 0.1804, 0.1529],
         [0.0000, 0.0000, 0.0000,  ..., 0.2824, 0.2275, 0.1882],
         [0.0000, 0.0000, 0.0000,  ..., 0.2824, 0.2275, 0.1882]],

        [[0.0000, 0.0000, 0.0000,  ..., 0.0039, 0.0039, 0.0039],
         [0.0000, 0.0000, 0.0000,  ..., 0.0039, 0.0039, 0.0039],
         [0.0000, 0.0000, 0.0000,  ..., 0.0039, 0.0039, 0.0039],
         ...,
         [0.0000, 0.0000, 0.0000,  ..., 0.1608, 0.1137, 0.0863],
         [0.0000, 0.0000, 0.0000,  ..., 0.2039, 0.1608, 0.1216],
         [0.0000, 0.0000, 0.0000,  ..., 0.2078, 0.1608, 0.1216]],

        [[0.0000, 0.0000, 0.0000,  ..., 0.0039, 0.0039, 0.0039],
         [0.0000, 0.0000, 0.0000,  ..., 0.0039, 0.0039, 0.0039],
         [0.0000, 0.0000, 0.0000,  ..., 0.0039, 0.0039, 0.0039],
         ...,
         [0.0000, 0.0000, 0.0000,  ..., 0.1255, 0.0863, 0.0588],
         [0.0000, 0.0000, 0.0000,  ..., 0.1686, 0.1333, 0.0941],
         [0.0000, 0.0000, 0.0000,  ..., 0.1686, 0.1333, 0.0941]]])
hahahaah 30
```

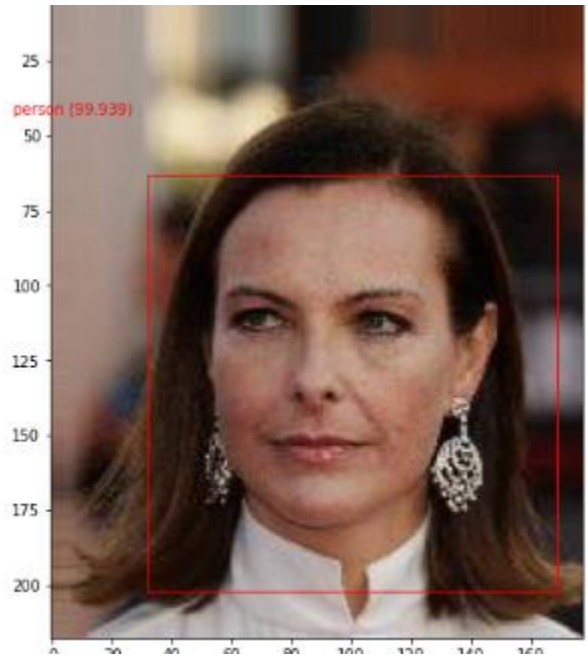
We used the model Resnet18 for facial detection :- Here is the snippet of the archticutre

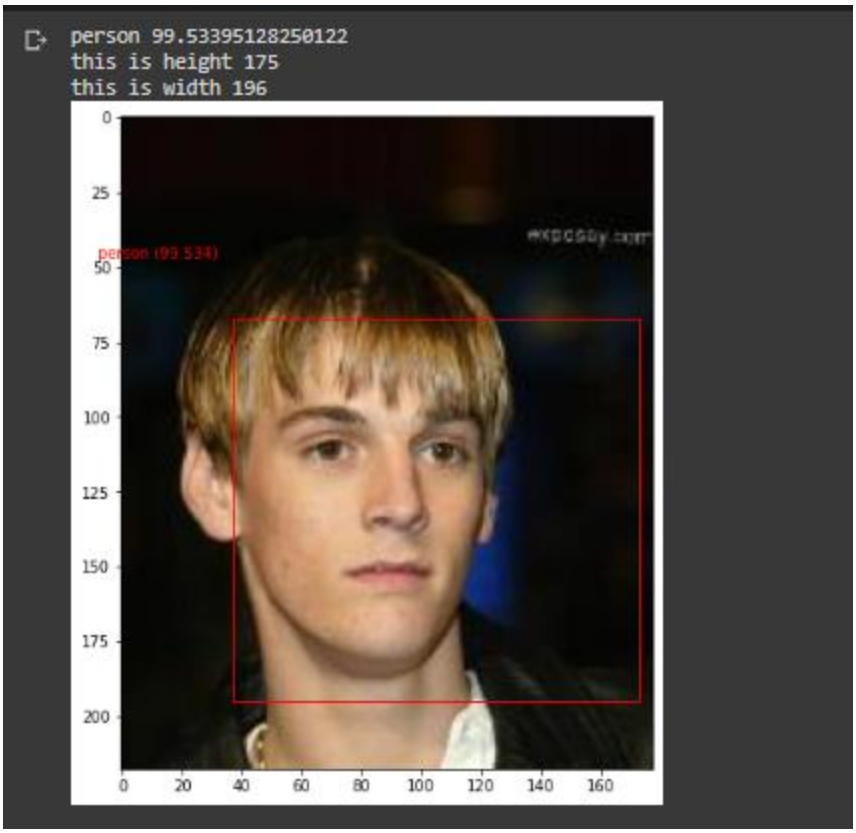
```
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
```

Now for getting the output vector of the following input :-

Img[0] that is :-


0
50
100
150
200
250
0
50
100
150
200
250

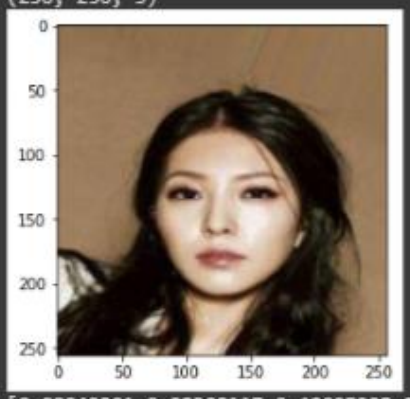

25
person (99.939)
50
75
100
125
150
175
200



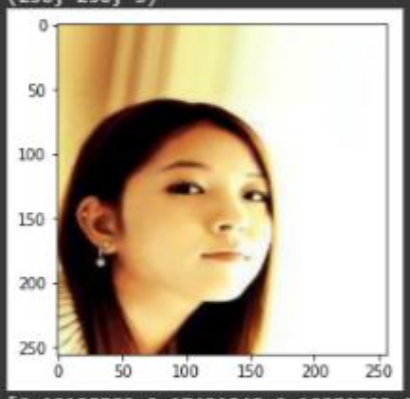
We obtain the following result : -

Only shoing the top three images after obtaing the eucladian distance :-

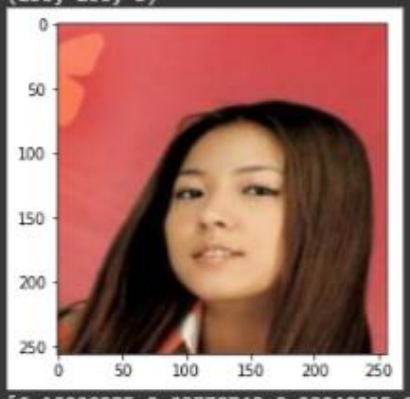
```
Image:
(256, 256, 3)
```



```
[0.09243301 0.99309117 0.40605205 0.19034068 0.08203603 0.08804234
 0.26584607 2.26206708 0.40302461 0.06514911 0.55163515 0.53381789]
Image:
(256, 256, 3)
```



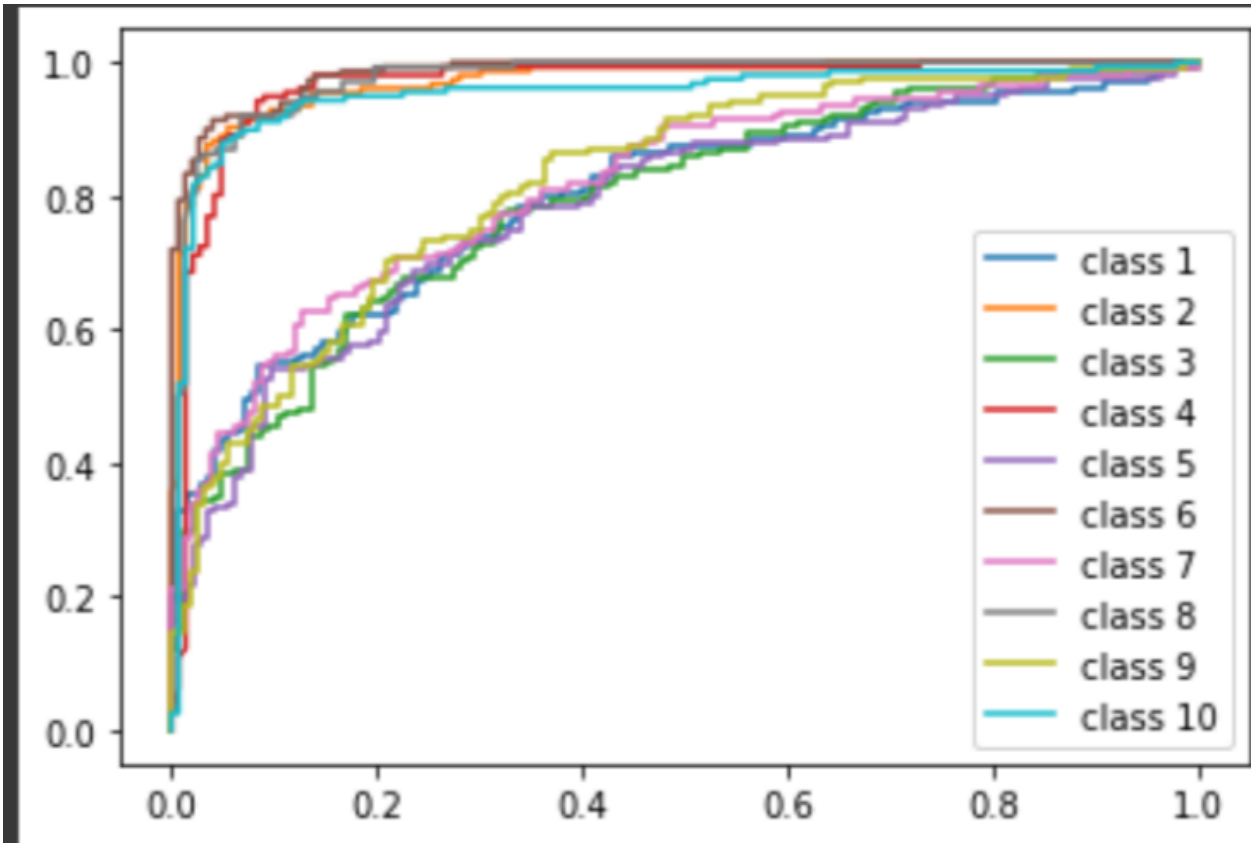
```
[0.19125582 0.17431848 0.10251503 0.48230773 0.24541034 0.03979106
 0.53805101 1.00800025 0.90545791 0.28462517 0.61402524 0.34574249]
Image:
(256, 256, 3)
```



```
[0.10200955 0.03778748 0.29042295 0.11400769 0.03048952 0.16116159
 0.61696339 1.90867174 1.17349637 0.13907434 0.22038138 1.27591133]
```

Recall vs Tou graph :-

Here class 1 – 10 are different folders of people so each class belongs to a person and the plot is recall vs tou.



Reference : -

Unified Embedding for Face Recognition and Clustering using FaceNet | Engineering Education (EngEd) Program | Section

YOLOv3: Real-Time Object Detection Algorithm (Guide) - viso.ai

Face Recognition with FaceNet | Kaggle

davidsandberg/facenet: Face recognition using Tensorflow (github.com)

timesler/facenet-pytorch: Pretrained Pytorch face detection (MTCNN) and facial recognition (InceptionResnet) models (github.com)

tbmoon/facenet: FaceNet for face recognition using pytorch (github.com)

facenet-trained-models · GitHub Topics

Face Recognition System Using FaceNet in Keras | Kaggle

R4j4n/Face-recognition-Using-Facenet-On-Tensorflow-2.X (github.com)