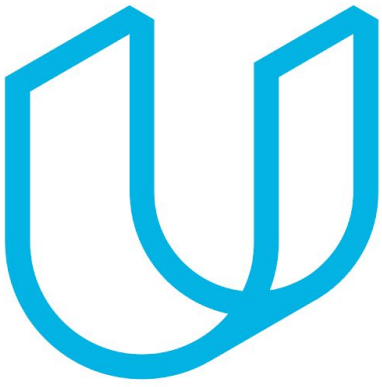
UDACITY MACHINE LEARNING NANODEGREE 2019

CAPSTONE PROJECT

CELEBRITY DETECTION USING DEEP LEARNING

DEEPAK SHARMA

January 25, 2020



UDACITY

1.1) Project overview

In our daily life, we meet people, we remember some of them by their names and when we meet them next time we recognize them this is a very basic task performed by our brain involuntarily when we first meet the person our brain tries to capture features on the face of the person like skin tone, face structure, hairstyle and various different aspect of the person and labeled the persons as per the name this is possible because our brain is getting trained to recognize people from the birth similarly a neural network can be trained to achieve this capability, a neural network can be trained to recognize a face.

The training of neural network is done by following phases, a model first learns to detect basic geometric pattern present on the image like edges and lines then the model learns the parts of faces like eyes, nose, hairline, jawline by joining the edges it learned previously after this the models try to remember the position and distance between eyes, nose, mouth and etc.

This much training is enough for a model to detect a face on an image. Identification of distinct faces require additional layers and training images, Every layer in a network is used for a specific purpose like the task of a single layer will be to only detect edges and then refine the information to the next layer by activating the different neuron for different faces, likewise, each layer has their specific function.

1.2) Problem Statement

There are 10 classes of celebrity which are labeled as per the name of the celebrity. The Goal of the model is to classify the person in a given image to one of the classes available with decent confidence based on how similar the person appears as one of the celebrities. The model should identify the celebrity if present or find a celebrity which looks similar to the person on the image

1.3) Metrics

The evaluation metric for this problem will be the ‘Classification Accuracy’ which is defined as the percentage of correct predictions.

Accuracy = correct classifications/number of classifications

Classification Accuracy was seemed to be the optimal choice metric as it is presumed that the dataset will be relatively symmetrical (as we will explore in the next section) with this being a multi-class classifier whereby the target data classes will be generally uniform in size. Other metrics such as Precision, Recall (or combined as the F1 score) were ruled out as they are more applicable to classification challenges that contain a relatively tiny target class in an unbalanced data set.

2 Analysis
2.1 Data Exploration and Visualization
2.1.1 Celebrity dataset

The images are collected from various sources like google images, Pinterest, and many other websites. For this project, we are using images of celebrity as the data, The dataset contains 1500 images of 10 different classes, which are :

- Alia Bhatt
- Ananya Pandey
- Emma Watson
- Hritik Roshan
- Jackie Chan
- Ranbir Kapoor
- Salman Khan
- Scarlett Johnson
- Shahrukh Khan
- Will Smith

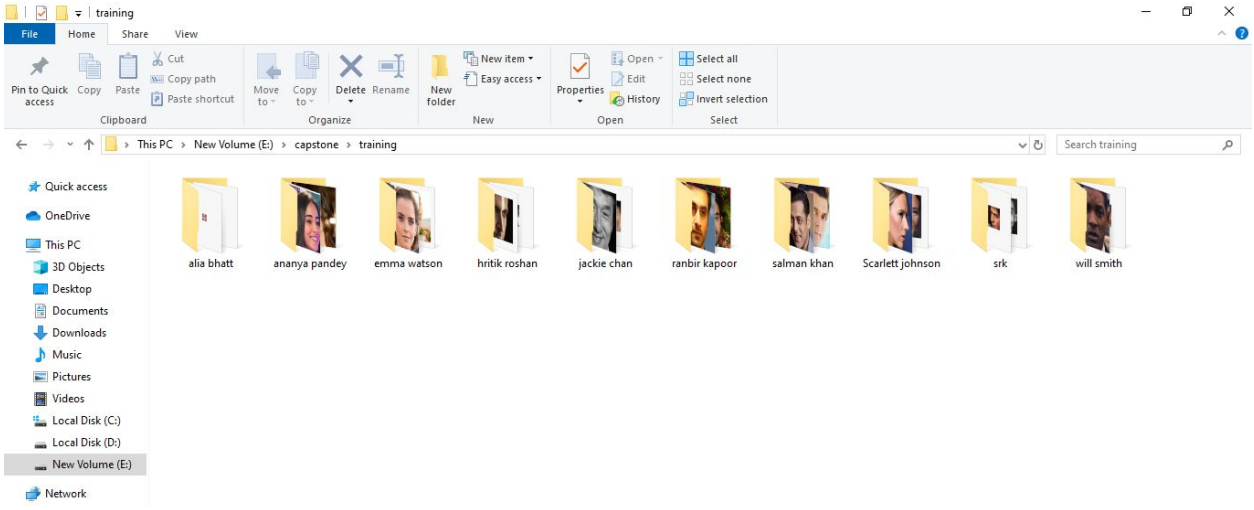
Distribution of Images

The whole dataset is divided into two subfolders “training” folder and “ validation” folder.

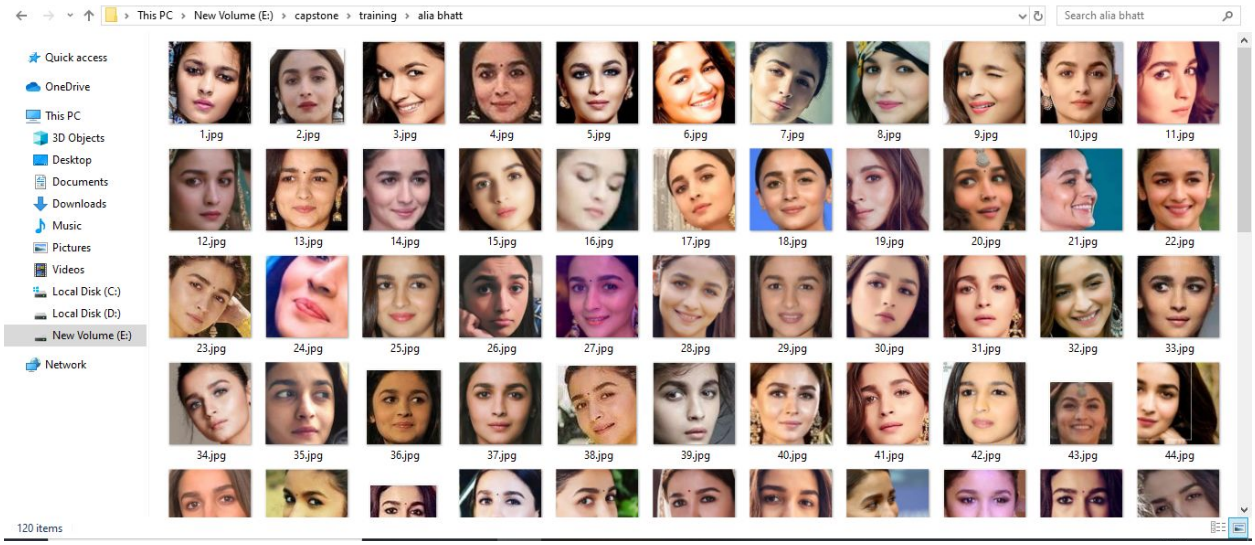
The training folder contains 1200 images and the validation folder contains 300 images.

Both folders contain the same type of hierarchy that is both folders have 10 subfolders and each subfolder consist of images of respective classes, These subfolders are named as per the class of images stored in it.

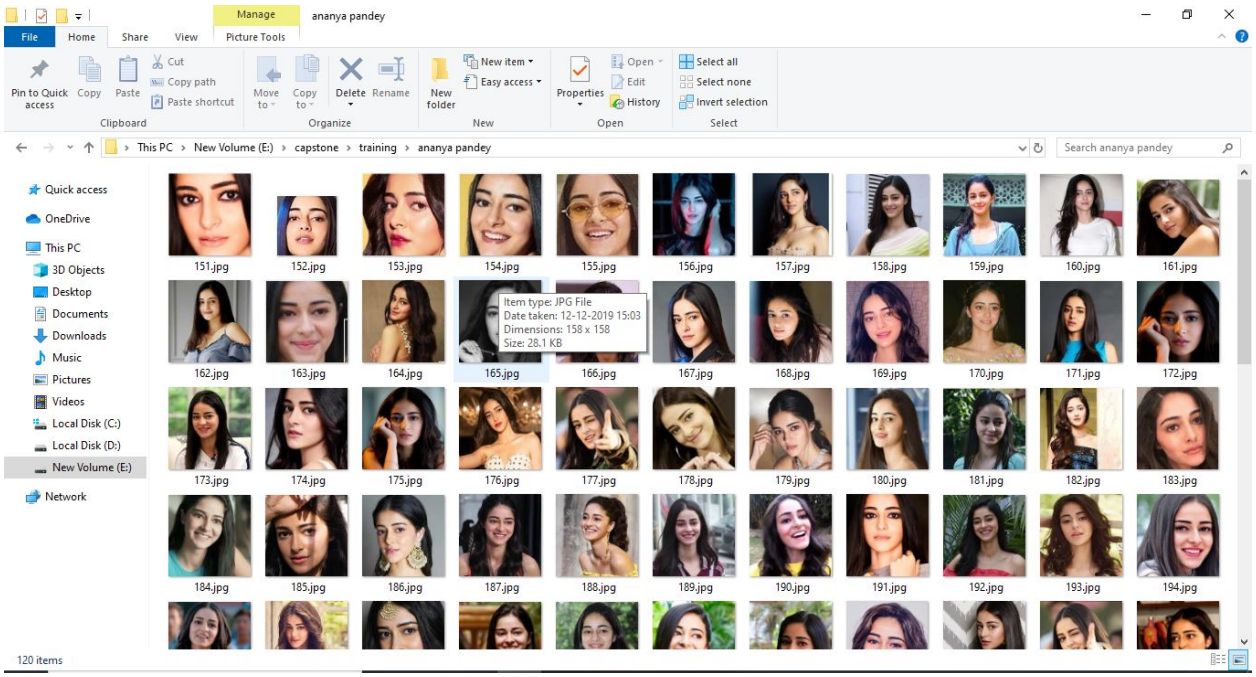
Subfolders in the “training” folder contain 120 images of respective class and the subfolders in the “validation” folder contain 30 images of the respective class.



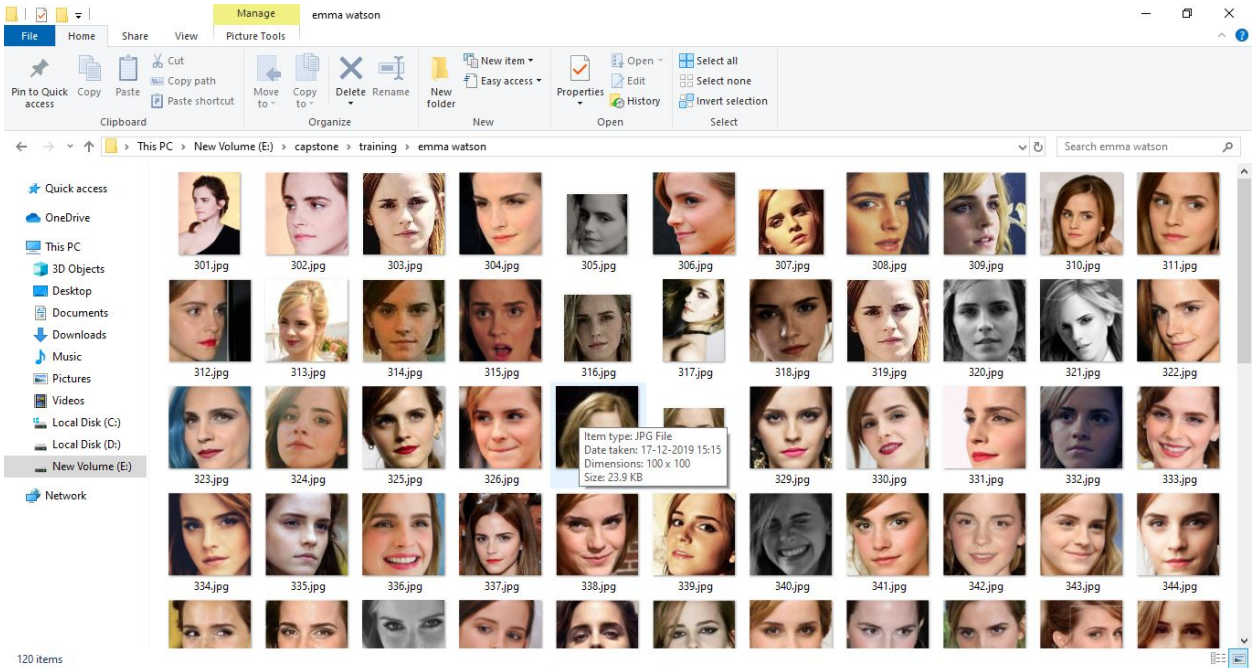
Class 1:- Alia Bhatt



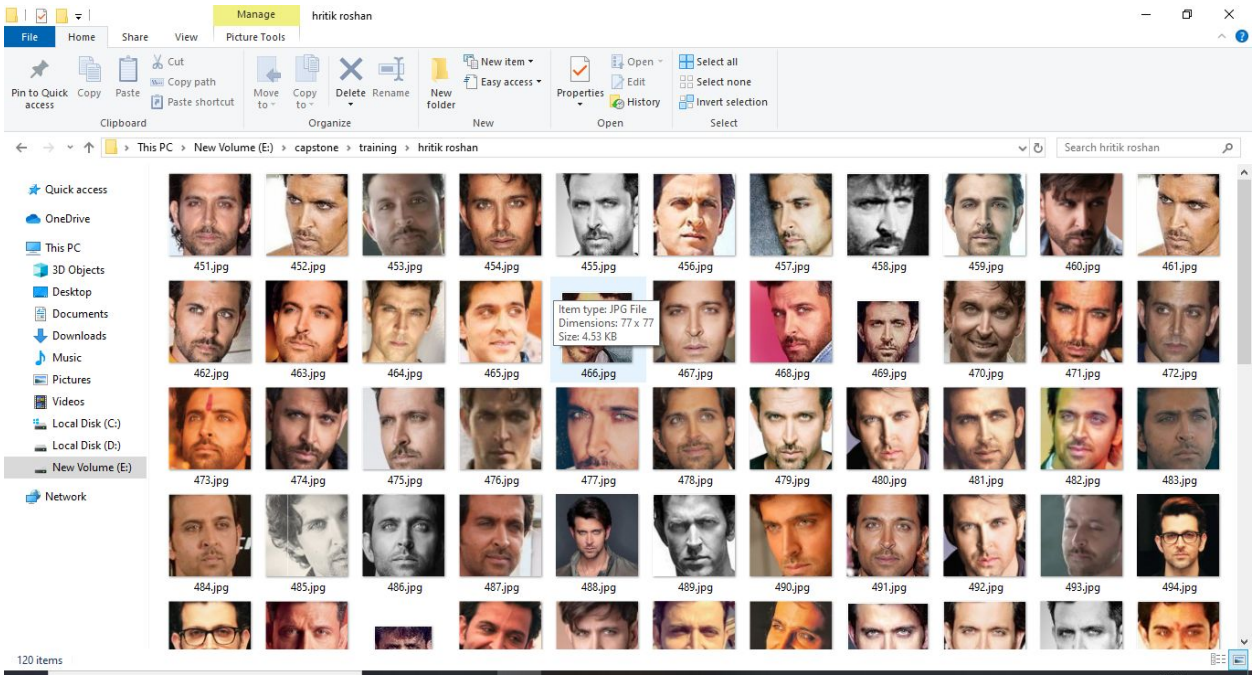
Class 2:- Ananya Pandey



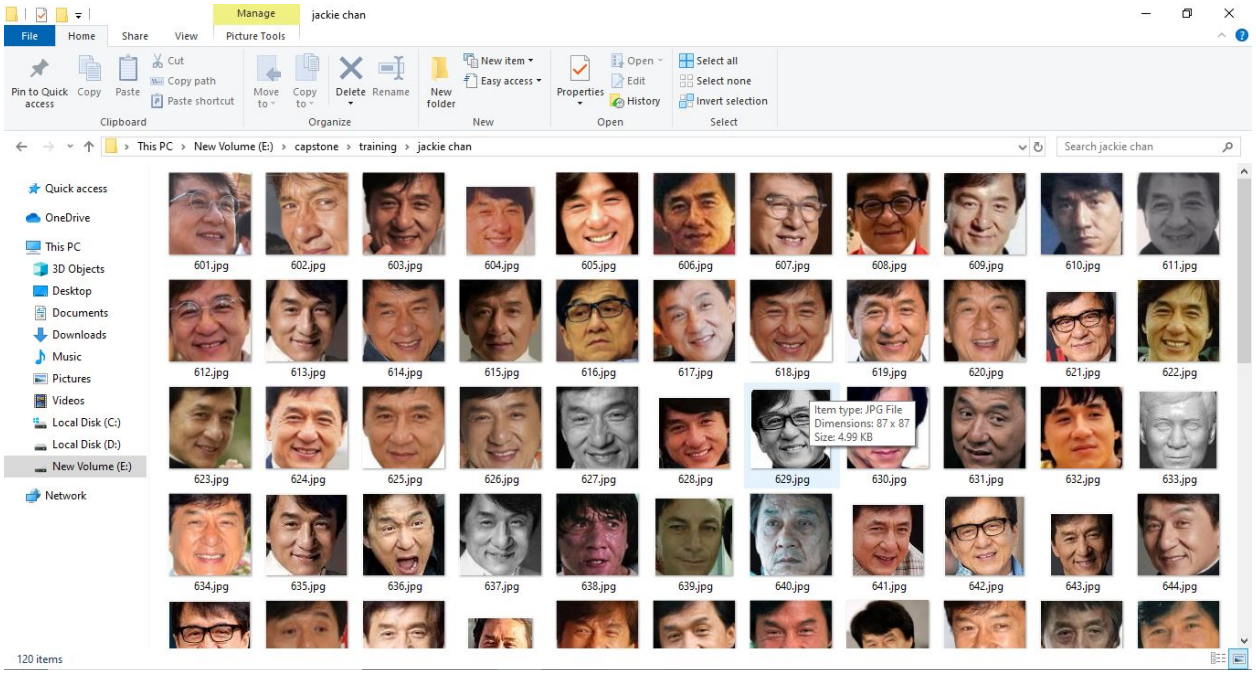
Class 3:- Emma Watson



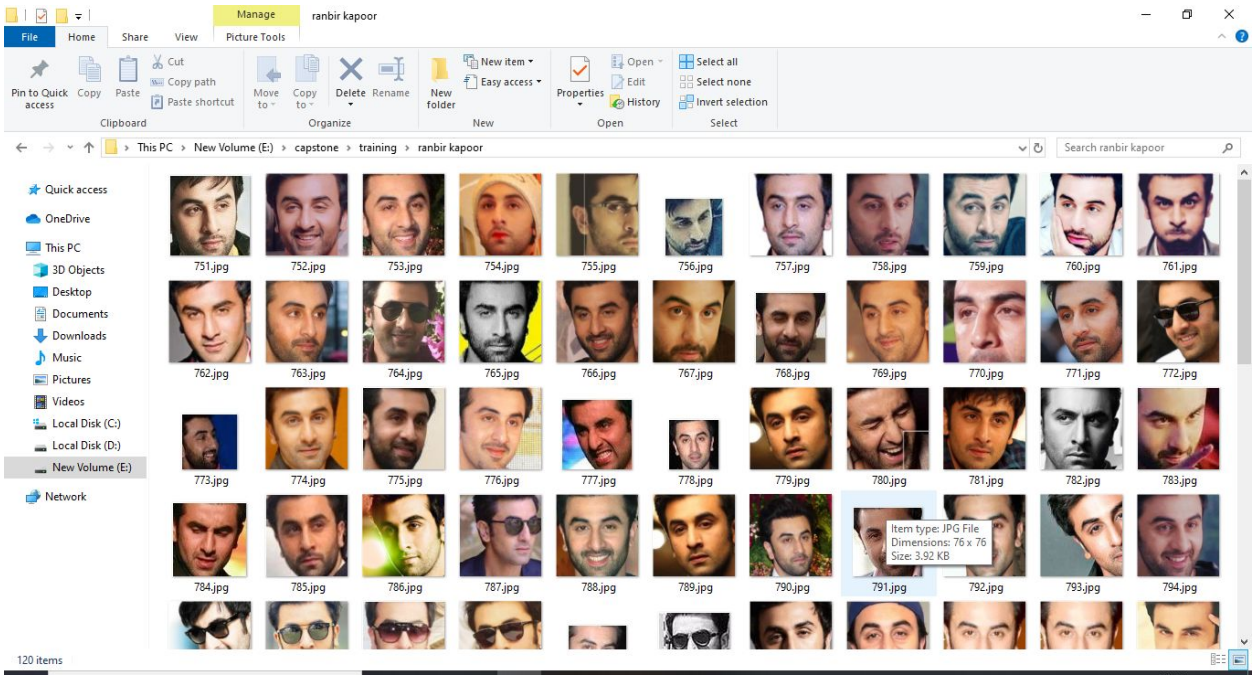
Class 4:- Hritik Roshan



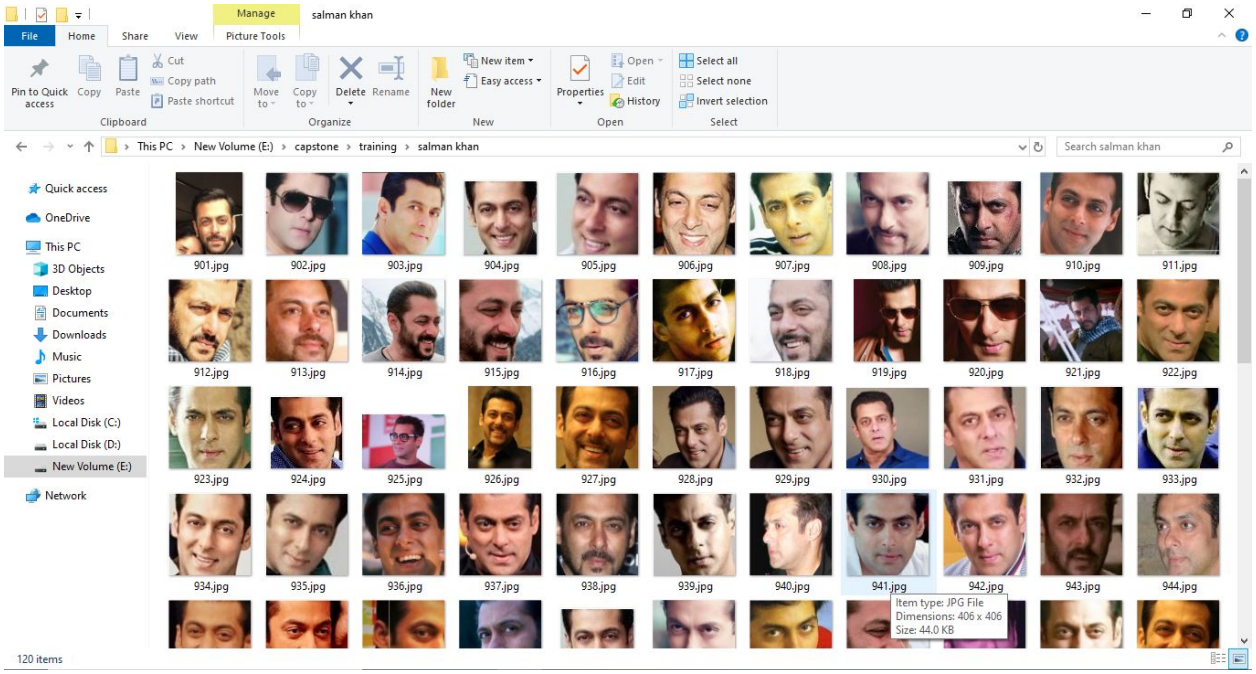
Class 5:- Jackie Chan



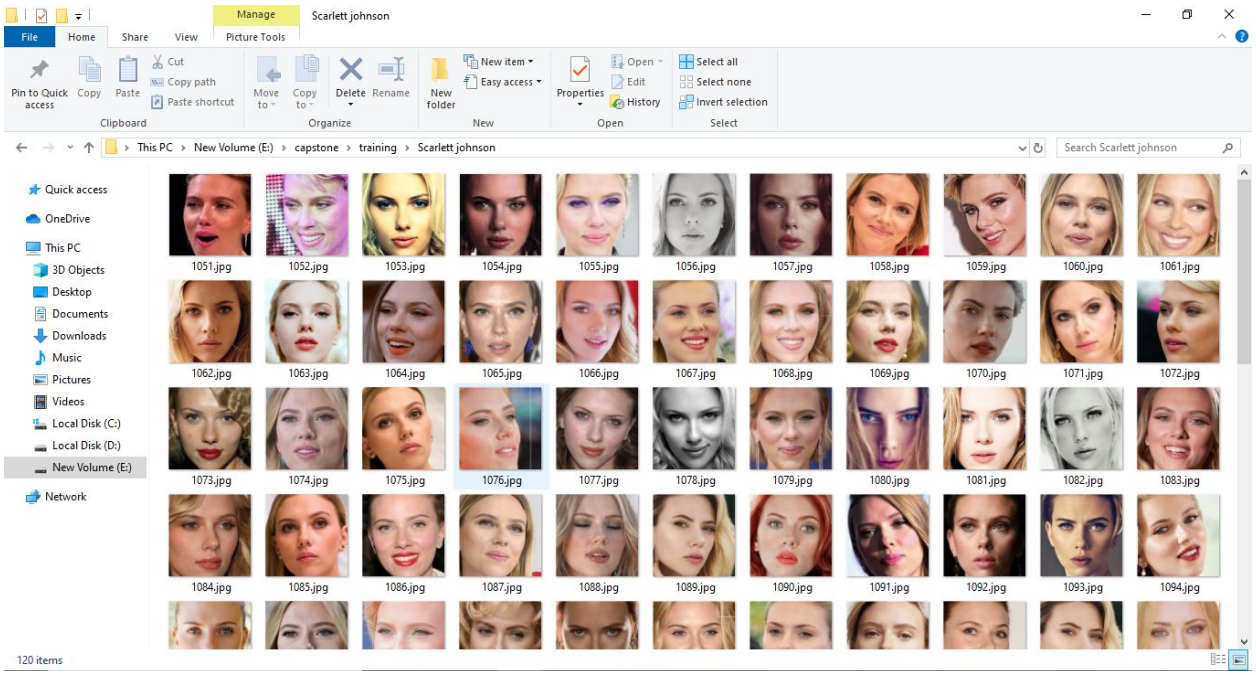
Class 6:-Ranbir Kapoor



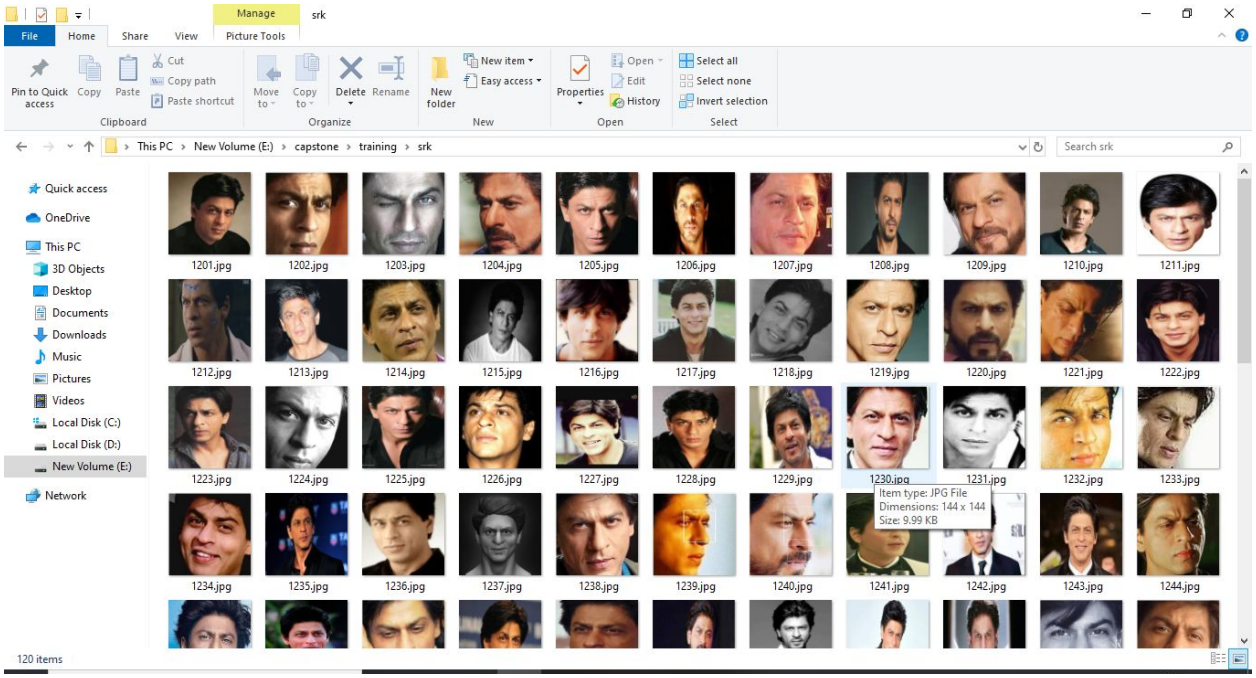
Class 7:- Salman Khan



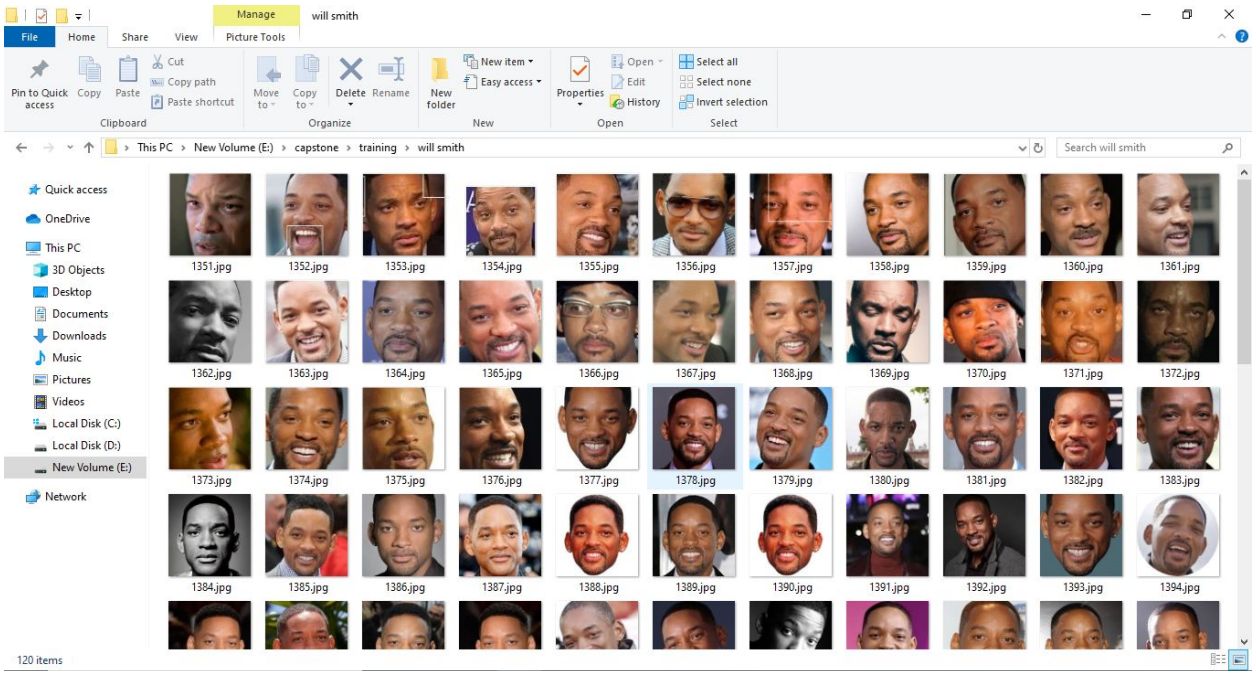
Class 8:- Scarlett johnson



Class 9:- Shahrukh Khan



Class 10:- Will Smith



2.2) Algorithms and technique

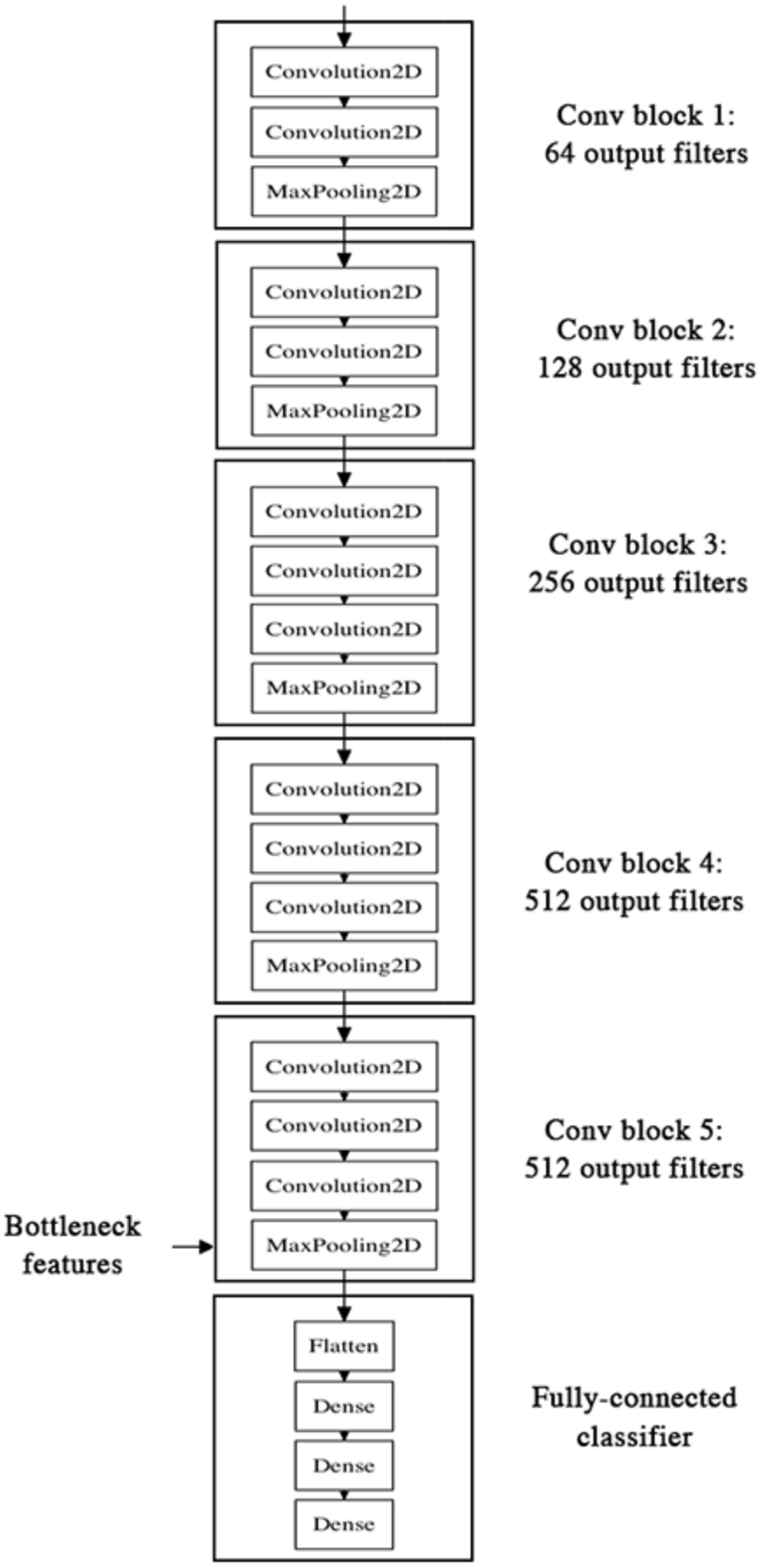
Our problem is image classification problem which uses deep learning as the solution, we are using CNN(Convolution Neural Network) to solve this problem effectively.

VGG Facenet

The pre-trained model has its own benefits Using a pre-trained model doesn't drain your computational power and uses fewer data to solve the problem.

Vgg facenet was developed especially for detecting faces.

The training of a model using the VGGFace2 dataset and softmax loss. The dataset contains 3.31 million images of 9131 subjects (identities), with an average of 362.6 images for each subject. Images are downloaded from Google Image Search and have large variations in pose, age, illumination, ethnicity and profession (e.g. actors, athletes, politicians). The VGGFace2 consists of a training set and a validation set. Here only the training part of the dataset is used. Identities overlap with LFW and have not been removed.


Convolution2D
Convolution2D
MaxPooling2D
Conv block 1:
64 output filters
Convolution2D
Convolution2D
MaxPooling2D
Conv block 2:
128 output filters
Convolution2D
Convolution2D
Convolution2D
MaxPooling2D
Conv block 3:
256 output filters
Convolution2D
Convolution2D
Convolution2D
MaxPooling2D
Conv block 4:
512 output filters
Convolution2D
Convolution2D
Convolution2D
MaxPooling2D
Conv block 5:
512 output filters
Bottleneck
features
Flatten
Dense
Dense
Dense
Fully-connected
classifier

2.3) Benchmark model

Amazon Rekogintion is one of the common usee API for face recognition also there are Google face recognition API and Microsoft's API to detect faces this API provides state of art level performance by giving accuracy almost close to hundred percent

	Kairos	Amazon	Microsoft	Google	Face++	SenseTime
Face Detection	✔	✔	✔	✔	✔	✔
Face Recognition	✔	✔	✔	X	✔	✔
Face Verification	✔	✔	✔	X	✔	✔
Age & Gender	✔	✔	✔	X	✔	✔
Ethnicity	✔	X	X	X	✔	✔
Multi-Face Tracking	✔	✔	✔	✔	✔	✔
On-Premise	✔	X	✔	X	✔	✔
Cloud	✔	✔	✔	✔	✔	✔

3) Methodology

3.1) Data preprocessing

The images are collected from various different resources that's why the resolution, size, and properties are different but our Neural Network works on the same size of images so all the images need prepossessing.

The preprocessing consist of the various stage to give desirable data, the stages are

3.1.1)Detecting face in the pictures: model works better if we just train it on the exact face of the person rather than training it on the whole image, we reduce the unwanted part present in the image such as background, body and giving more emphasis on the face.

We used a face haar cascade to detect the face present in the image this Algorithm gives the co-ordinates of the space in which the face is present.

3.1.2) Cropping the pictures: Using the co-ordinates the face is cropped in the images. It is possible that a single picture can contain more than one face, a manual selection is done on the cropped images to keep the quality of images intact.

3.1.3) Resizing the pictures: All the selected cropped images are then resized to 224*224 resolution as it is the optimal resolution present in images.

3.1.4)Renaming the pictures: After resizing the images the were renamed to numbers to give every image a unique id and maintaining the data.

4)Implementation

Approach 1

Initially, we develop a Multi-Layer Perceptron Neural Network using Keras and TensorFlow at the background

We started by creating a sequential model and then adding a layer on layer.
We added a Convolution layer as an input layer with input shape 224*224*3 as 224*224 represents the resolution of input images and 3 represents the channels of the images with an activation function relu.
There are multiple hidden layers such as pooling, Dense and Flatten layer.
The output layer has 10 unit which represents the number of classes and a softmax function is used on the output layer as this function provide the probability of each class.
Data augmentation
In order to make the most of our few training examples, we will "augment" them via a number of random transformations, so that our model would never see twice the exact same picture. This helps prevent overfitting and helps the model generalize better.

The architecture of the network is given below

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 222, 222, 32)	896
activation_1 (Activation)	(None, 222, 222, 32)	0
max_pooling2d_1 (MaxPooling2	(None, 111, 111, 32)	0
conv2d_2 (Conv2D)	(None, 109, 109, 32)	9248
activation_2 (Activation)	(None, 109, 109, 32)	0
max_pooling2d_2 (MaxPooling2	(None, 54, 54, 32)	0
conv2d_3 (Conv2D)	(None, 52, 52, 64)	18496
activation_3 (Activation)	(None, 52, 52, 64)	0
max_pooling2d_3 (MaxPooling2	(None, 26, 26, 64)	0
flatten_1 (Flatten)	(None, 43264)	0
dense_1 (Dense)	(None, 64)	2768960
activation_4 (Activation)	(None, 64)	0
dropout_1 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 10)	650
activation_5 (Activation)	(None, 10)	0

Total params: 2,798,250
Trainable params: 2,798,250
Non-trainable params: 0

For compiling the loss is categorical cross-entropy because it is a multi-class classification problem and the optimizer is used as RMSPROP.

Result for this was:

```
Epoch 1/50
60/60 [==============================] - 35s 581ms/step - loss: 2.0727 -
accuracy: 0.2900 - val_loss: 1.8005 - val_accuracy: 0.3600
Epoch 2/50
60/60 [==============================] - 35s 584ms/step - loss: 1.8662 -
accuracy: 0.3558 - val_loss: 1.5477 - val_accuracy: 0.4300
Epoch 3/50
60/60 [==============================] - 35s 579ms/step - loss: 1.7141 -
accuracy: 0.4067 - val_loss: 1.4589 - val_accuracy: 0.5233
Epoch 4/50
60/60 [==============================] - 35s 586ms/step - loss: 1.5985 -
accuracy: 0.4408 - val_loss: 1.0134 - val_accuracy: 0.5300
.
.
.
.

Epoch 44/50
60/60 [==============================] - 35s 583ms/step - loss: 0.5932 -
accuracy: 0.7975 - val_loss: 1.0624 - val_accuracy: 0.7767
Epoch 45/50
60/60 [==============================] - 35s 581ms/step - loss: 0.6594 -
accuracy: 0.7767 - val_loss: 1.8512 - val_accuracy: 0.7567
Epoch 46/50
60/60 [==============================] - 35s 583ms/step - loss: 0.6306 -
accuracy: 0.8025 - val_loss: 1.0127 - val_accuracy: 0.7700
Epoch 47/50
60/60 [==============================] - 35s 586ms/step - loss: 0.6265 -
accuracy: 0.7933 - val_loss: 1.0373 - val_accuracy: 0.7600
Epoch 48/50
60/60 [==============================] - 35s 582ms/step - loss: 0.6727 -
accuracy: 0.7567 - val_loss: 0.5545 - val_accuracy: 0.7233
Epoch 49/50
60/60 [==============================] - 35s 583ms/step - loss: 0.5540 -
accuracy: 0.8317 - val_loss: 0.9239 - val_accuracy: 0.7967
Epoch 50/50
```

```
60/60 [==============================] - 35s 581ms/step - loss: 0.5894 -
accuracy: 0.8142 - val_loss: 0.3673 - val_accuracy: 0.7400
```

The model doesn't perform well this is due to many factors first we don't have enough data to train model and second the model is not able to capture the features of the images, so I used a different approach.

Approach 2

Here we are using transfer learning for the classification as the result was not acceptable previously.

Transfer learning is an approach of Deep Learning where we use a pre-trained model to classify our images, These pre-trained models are trained on millions of images and the weights are stored for further use.

We are using the VGG Facenet model which is developed by Visual Geometry Group (VGG) at the University of Oxford.

Here we develop a layer on top of Facenet architecture and train the architecture on our data by keeping the weights of Facenet intact this gives exceptional result on training data and on the validation data

Data Augmentation is also performed in this process where every image creates similar images like original but not the same.

The Architecture of the model is:

Model: "sequential_3"

Layer (type)	Output Shape	Param #
flatten_3 (Flatten)	(None, 25088)	0
dense_5 (Dense)	(None, 256)	6422784
dropout_3 (Dropout)	(None, 256)	0
dense_6 (Dense)	(None, 10)	2570

Total params: 6,425,354
Trainable params: 6,425,354
Non-trainable params: 0

4) Training

Here we will train our model.

We will start with 50 epochs which is the number of times the model will cycle through the data.

The model will improve on each cycle until it reaches a certain point.

We will also start with low batch size, as having a large batch size can reduce the generalization

The ability of the model.

Epoch 47/50
1200/1200 [==============================] - 10s 8ms/step - loss: 5.0664e-08 - accuracy: 1.0000 - val_loss: 0.0571 - val_accuracy: 0.9900
Epoch 48/50
1200/1200 [==============================] - 10s 8ms/step - loss: 2.1154e-06 - accuracy: 1.0000 - val_loss: 0.0453 - val_accuracy: 0.9900
Epoch 49/50
1200/1200 [==============================] - 10s 8ms/step - loss: 1.2232e-06 - accuracy: 1.0000 - val_loss: 0.0527 - val_accuracy: 0.9867
Epoch 50/50
1200/1200 [==============================] - 10s 8ms/step - loss: 5.0524e-06 - accuracy: 1.0000 - val_loss: 0.0533 - val_accuracy: 0.9900

5)Test the model

Here we will check the performance of our new model

```
score = model.evaluate(train_data,train_labels,verbose=0)
print("Training Accuracy",score[1])

score = model.evaluate(validation_data,validation_labels,verbose=0)
print("Training Accuracy",score[1])
```

```
Training Accuracy 1.0
Training Accuracy 0.9900000095367432
```

6)Result

6.1)Model Evaluation and Validation

The model gives a good result than the previous one. The model completes the task of identifying the images of celebrities successfully.
Models work pretty well on unseen data as well.

6.2)Justification

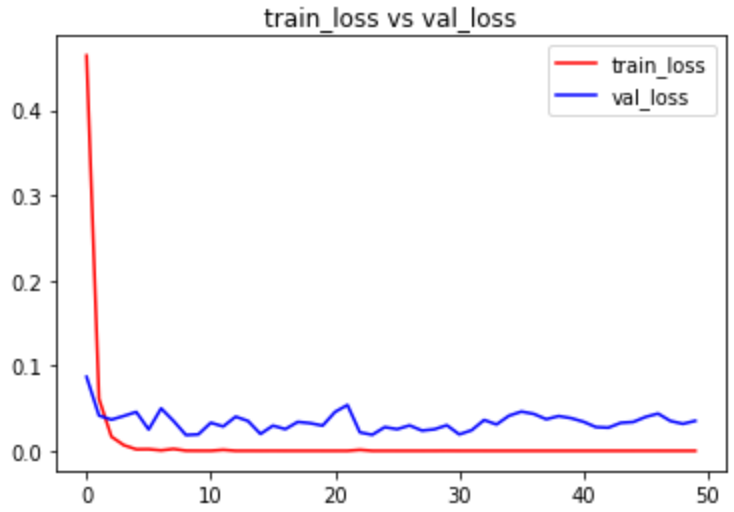
The model works better than the benchmark model where the benchmark model got the testing accuracy as 98% and Our models give an accuracy of 99%

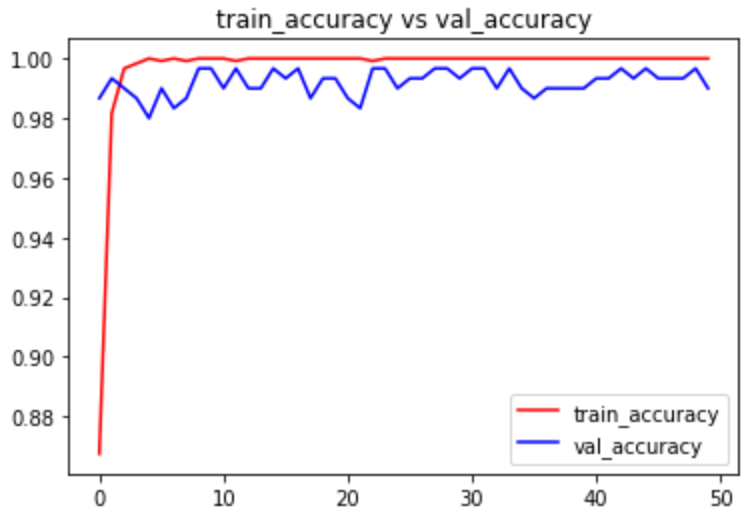
7)Conclusion

7.1)Visualizing the result

```
# visualizing losses and accuracy
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
%matplotlib inline

train_loss=hist.history['loss']
val_loss=hist.history['val_loss']
```

```
train_acc=hist.history['accuracy']
val_acc=hist.history['val_accuracy']

epochs = range(len(train_acc))

plt.plot(epochs,train_loss,'r', label='train_loss')
plt.plot(epochs,val_loss,'b', label='val_loss')
plt.title('train_loss vs val_loss')
plt.legend()
plt.figure()

plt.plot(epochs,train_acc,'r', label='train_accuracy')
plt.plot(epochs,val_acc,'b', label='val_accuracy')
plt.title('train_accuracy vs val_accuracy')
plt.legend()
plt.figure()
```





7.2)Reflection

The whole project has two approaches to solve the problem First we tried to solve the problem with a simple deep learning model Multilayer perceptron but its end up giving very poor result due to lack of sufficient data
Than we Tried solving the problem by transfer learning where we built our model on top of the VGG pre-trained model this solved our problem and gave a very good performance.
The most difficult part was to create the dataset huge amount of time was drained in just creating a quality dataset once the dataset got ready the getting the state of art result was the challenge, I tried to solve this by data augmentation by using the first simple model it gave a better result than previous but not up to the mark.

7.3)Improvement

Improvement can be done in terms of generating new datasets, as I generated my dataset using a haar cascade algorithm to crop the face but MTCNN works better than this.
I tried various other pre-trained models like Imagenet, Resnet but I think the deep net can give a similar result or better result because it is also trained on the face data.

8)References

- https://github.com/davidsandberg/facenet/wiki/Training-using-the-VGGFace2-dataset
- https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
- https://machinelearningmastery.com/how-to-develop-a-face-recognition-system-using-facenet-in-keras-and-an-svm-classifier/