
IMAGE SEGREGATION

ABSTRACT

There has been an exponential increase in the use of smartphones to store and share images. An average user has a thousand images accumulated in their smartphones. All these images are stored in a single Gallery folder. In the current age of burgeoning social media applications, unwanted images seep their way into the gallery. Consequently the phone memory reaches its maximum capacity and leads to slowing down of the smartphone. One then needs to go through all the images to remove unwanted ones.

To save time and space, we propose an automated solution that segregates images into four categories as people (friends/family or strangers) and text or memes. To accomplish this, we focus on a few select approaches such as VGG16 and OpenCV(DNN) and then elaborate on the performance of each of the models. The dataset used has been integrated from various sources such as the CelebFaces Attributes (CelebA) Dataset [1] for face images. To better depict the real world scenario, we created our own dataset by scraping images for people, text and memes from social media sites such as reddit and instagram.

Keywords VGG16 · OpenCV(DNN) · CelebFaces Dataset · Reddit · Instagram

1. Introduction

Sorting out images manually is a time-consuming and monotonous job that requires no application of mental capacity whatsoever. Automating such a task that eventually saves us time and energy is much needed with the surge in the use of several forms of technology and hoarding of unnecessary data.

The primary goal of this project is classification, which is a supervised learning method. Machine Learning provides

several classification methods that aid us in this process, namely - Linear Classifiers (Logistic Regression, Naive Bayes Classifier), Nearest Neighbours, Support Vector Machines, Decision Trees, Boosted Trees, Random Forest and Neural Networks. The Naive Bayes Classifier assumes that each of the predictors is independent and the presence or absence of any of the feature is unassociated with the other features. Logistic Regression. The Nearest Neighbour approach classifies a new point by looking at the labels of the k-points closest to it. A Decision Tree with leaf and decision nodes is developed by breaking down the data into smaller subsets where the nodes represent a classification decision. Several uncorrelated decision trees are constructed and the mode of the trees is returned as classification in a Random Forest classifier. Finally, in a Neural Network, each unit applies a function to the input passed to it and passes it on to the next layer. [2]

In this project we use Neural Networks to classify the images.

2. Research Questions

Our aim is to answer three primary questions,

1. Classification of images into 2 classes, images with people and other images.
2. Classification of the images with people into pictures of Friends/Family(High Priority) and pictures of other people (Lesser Priority).
3. Segregation of the images previously classified as other images into clusters of images with text(potentially important) and memes(junk).

We have several images in our gallery that consist of our own pictures along with pictures of us with friends/family (which are relevant), pictures of random people which are not very important, and also memes we share (irrelevant) and images of text such as notes and documents (which may possibly be crucial).

3. Methodology.

3.1 VGG16

It is a 16-layer deep convolutional neural network that classifies objects into 1000 categories. It has been pre trained on over a million images from the ImageNet dataset and has an image input size of 224x224. [3].

To classify images into people and other images, we use the VGG16 network. Initially, the dataset was too small(1000 images) and the result wasn't favourable. To overcome this, the number of images for training was increased to 10,000 images. However, this didn't lead to any improvement in the performance. Eventually, it was concluded that 16 layers might be too much and we customised the model by removing a few layers before deciding that an 8-layer deep network would suffice. We also applied data augmentation (using only flip) to get better results. This same network was used to separate the 'memes' from 'text' in the 'other' images category, thus

answering two of the research questions. The algorithm can be tested using the test set and stores the three categories: text, faces and memes in 3 different directories.

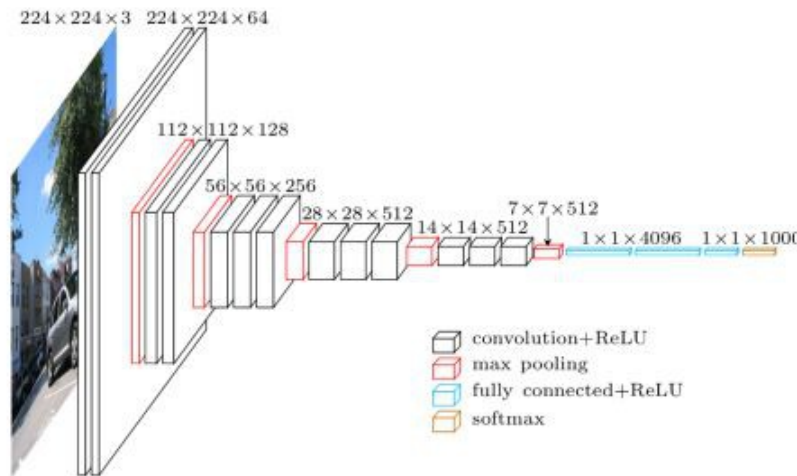


Fig - 5.1 The architecture of VGG16

3.2 OpenCV (DNN module)

“OpenCV (Open Source Computer Vision) is a library with functions that mainly aiming real-time computer vision”[4]. The DNN module in this library comes pre-trained along with the bundle. It supports several frameworks such as Caffe, Tensorflow, Torch, Darknet, etc [5].

To classify the images containing the people, two possible approaches can be used, i.e., OpenCV or Dlib. With the OpenCV - DNN module, it is easier to handle small sized faces in a fast, accurate and effective manner. This also has the advantage that it can detect faces at various angles and positions.

The Dlib library - while being the faster method on the cpu, is not as strong in detecting smaller-sized faces (< 70x70). However, similar to OpenCV, it can detect faces at various angles and positions.

Due to this trade-off, we prefer using the OpenCV library to classify images into those with higher priority such as friends or family and those with lesser priority such as those of random strangers. This answers our second research question.

The model responsible for quantifying each face in an image is from the OpenFace project, a python and torch implementation of face recognition with deep learning.

4. Dataset

4.1 Dataset Version 1

We intend to use the project to solve the modern day problem of saving memory and time, by segregating images into

folders. One can then easily remove the low priority files over the higher priority ones in order to manage memory. In our case we are assuming that memes and pictures of other people are of lesser priority than that of the user.

We categorise images into 3 subsections:

1. Human
2. Memes
3. Text

Initially we worked on a different dataset than our actual final dataset. We decided to use 1000 images as a set of Training images for our model and 100 images as a set of testing images in the model. The estimated size for this dataset is 1.5MB. The first dataset that we worked with is enlisted as follows:

1. Faces of People: We used GeorgiaTech database to collect images of people [6]. It consists of images of 50 people. The shots of each person are taken from a different angle. The resolution of each image is 640 by 480 pixels. Each image is a jpeg image of 15 colours. Each face is approximately of size 150 by 150 pixels. All images are clicked with varied light settings and facial expressions. The entire dataset is of 128 MB. We used 1000 images from this dataset.
2. Memes: We used a python scraper [7] to scrape manga memes from Google. This dataset consisted of 1000 images of manga memes. The dataset consists of mostly animated memes of varied sizes, pixel values and colours.
3. Text: Harvard dataset was used to collect Text images for our dataset [8]. The dataset is called IAMHandwriting database. We had to create an account on the website in order to get the database. The database consisted of images with handwriting of different people. Each image was of size 300dpi, with 256 grey levels. We downloaded 1000 images manually.

However after the first run of our algorithm we soon found out that we needed more images as data to feed to the network, since the network was showing approximately 50% accuracy for Text and Memes datasets, and 34% accuracy for Faces, Memes and Text datasets as described above. We then came to the conclusion that the network is not really learning anything and is just making the best guess of an image being in a particular class because the network does not have enough data. We therefore decided to use another dataset: Dataset Version 2.

4.2 Dataset Version 2

Due to less number of images used to feed our network, we decided on coming up with a new dataset. VGG-16 requires a dataset of 1,00,000 images to train it in the ideal manner. Due to lack of computing power and resources we shrunk the network and modified the original architecture of VGG -16. Upon researching, we came to a conclusion that we need 15,000 images for our network. Since, we were following a new approach and using a new dataset, the right path would be to use real world images for low priority classification and not a pre formed dataset. Most of the unwanted images (in our case lower priority images) seep into the gallery through social media. Therefore, we

decided to make our own dataset that has images scraped from Social Media.

Following are the social media platforms that we used to scrape images from:

1. Instagram
2. Reddit

For higher priority images, (in our case images with human faces in them) we decided to use a ready made dataset of human faces, since it was readily available.

The datasets are elaborated as follows:

1. Human: For human faces, we used celebrity dataset from kaggle. The dataset consists of 2,02,599 images of various celebrities consisting of 10,177 people. However we used only 5000 images from this dataset.
2. Memes: For memes, we used an instagram scraper [9]. that scrapes memes from meme pages.
 - a. Fukkard
 - b. Capdt
 - c. Meme_Raja
 - d. Meme_baasha

The training accuracy that we achieved using this dataset was 60%-70%. This time the model did perform better on 2 sets. We needed an even better accuracy as the output of our first CNN architecture was to be fed to our face recognition model. We therefore decided on increasing the dataset further.

4.3 Dataset Version 3

Version 3 of the dataset is an extended category of version 2, with 10,000 images of each category, including text images, which resulted in a dataset of 30,000 images.

3. Text: For text images we used an instagram scraper that scrapes images with quotes from multiple pages. The pages that we scraped images from are enlisted as below:
 - a. extreme_quotes
 - b. feelingsbyquotes
 - c. quotes_n_much_more
 - d. quotesbyweheartit
 - e. scoopwhoopwordgasm
 - f. thegoodquote
 - g. wordporm

On pre-processing and cleaning this data we received an approximate 95% accuracy.

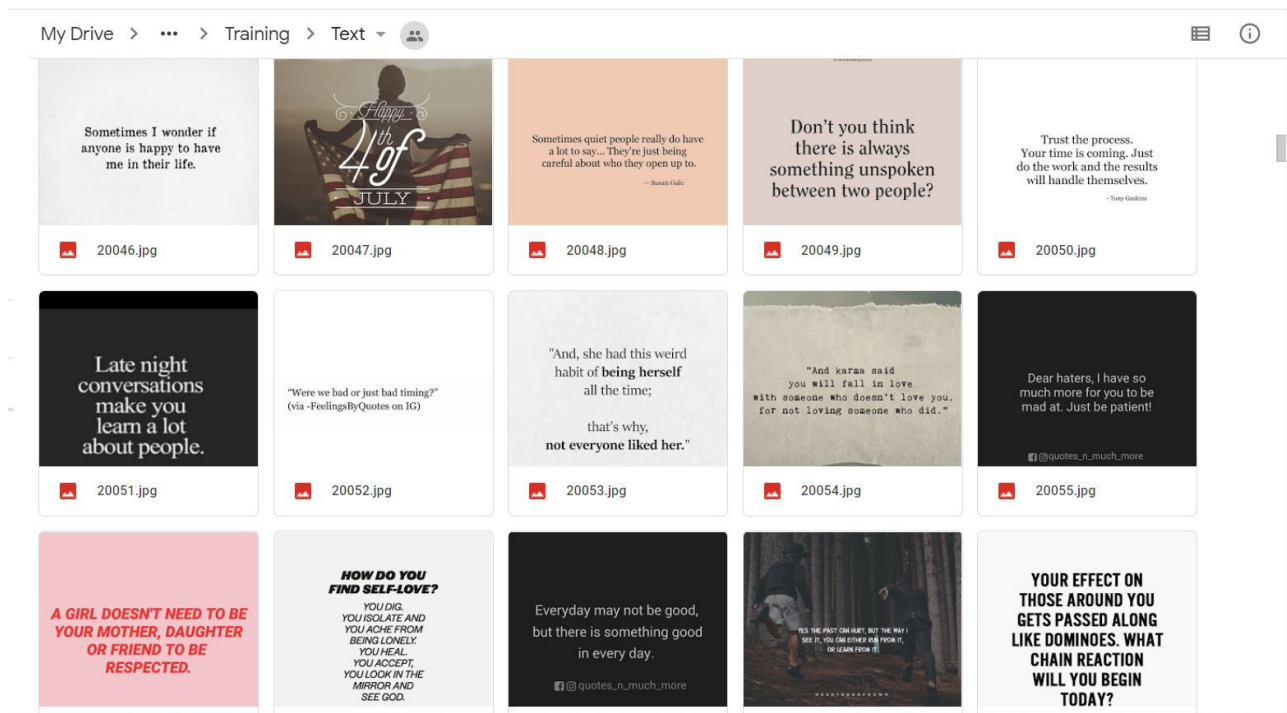
4.4 Data Preprocessing

For data preprocessing we performed the following steps:

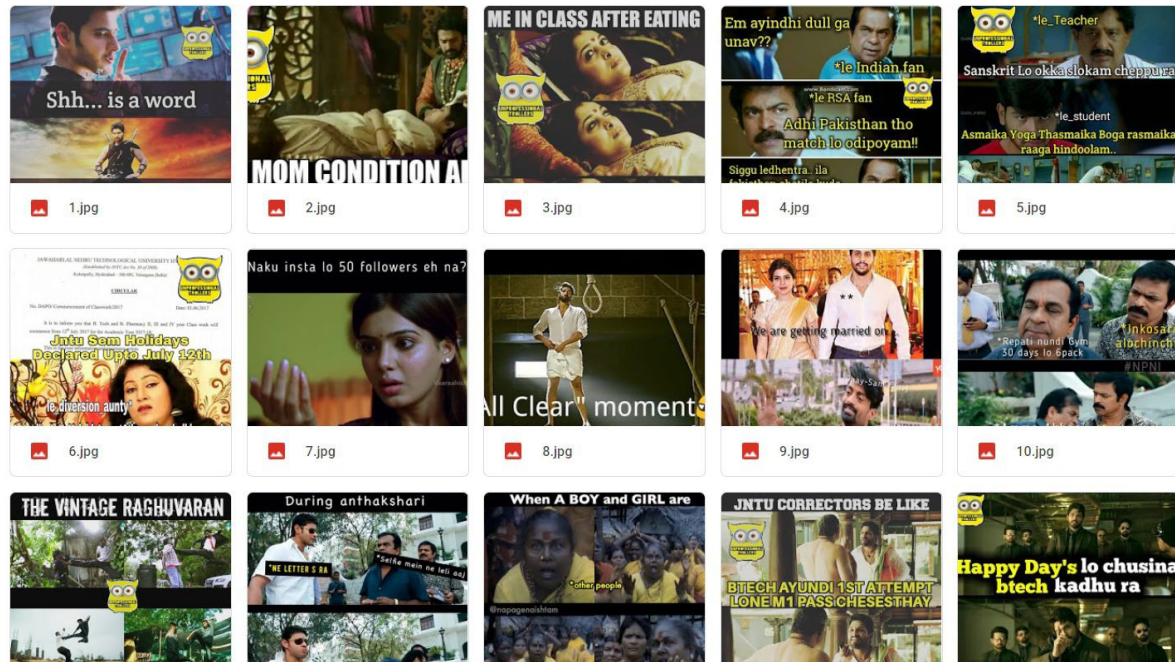
1. Removed videos and gifs from the data.
2. Removed copy of images that were present in multiple pages.
3. For the memes dataset: We removed images of only humans in the dataset.
4. Renamed all the images using a python script.
5. Zipped and Unzipped all images using a python script.

4.5 Examples

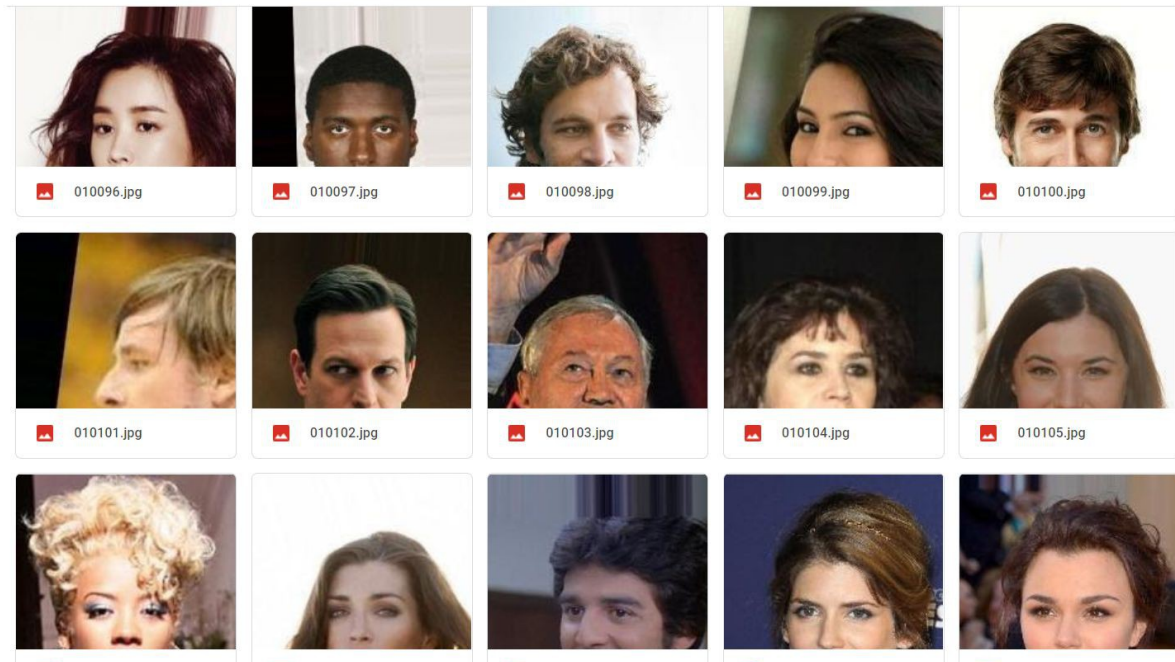
4.5.1 Text Dataset



4.5.2 Meme Dataset



4.5.3 Faces Dataset



5. Results

The Results for our model with training data:

```
+ Code + Text Connect Editing
[ ] WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:197: The name tf.ConfigProto is deprecated. Please use tf.compat.v1.ConfigProto ins
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:207: The name tf.global_variables is deprecated. Please use tf.compat.v1.global_var
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:216: The name tf.is_variable_initialized is deprecated. Please use tf.compat.v1.is_
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:223: The name tf.variables_initializer is deprecated. Please use tf.compat.v1.varia

50/50 [=====] - 788s 16s/step - loss: 0.7060 - acc: 0.6800 - val_loss: 0.3091 - val_acc: 0.8969

Epoch 00001: val_acc improved from -inf to 0.89687, saving model to vgg16_1.h5
Epoch 2/100
50/50 [=====] - 780s 16s/step - loss: 0.2286 - acc: 0.9319 - val_loss: 0.1916 - val_acc: 0.9250

Epoch 00002: val_acc improved from 0.89687 to 0.92500, saving model to vgg16_1.h5
Epoch 3/100
50/50 [=====] - 785s 16s/step - loss: 0.1877 - acc: 0.9431 - val_loss: 0.3453 - val_acc: 0.9062

Epoch 00003: val_acc did not improve from 0.92500
Epoch 4/100
50/50 [=====] - 803s 16s/step - loss: 0.1708 - acc: 0.9513 - val_loss: 0.2279 - val_acc: 0.9344

Epoch 00004: val_acc improved from 0.92500 to 0.93437, saving model to vgg16_1.h5
Epoch 5/100
50/50 [=====] - 772s 15s/step - loss: 0.1033 - acc: 0.9688 - val_loss: 0.1076 - val_acc: 0.9625

Epoch 00005: val_acc improved from 0.93437 to 0.96250, saving model to vgg16_1.h5
Epoch 6/100
50/50 [=====] - 776s 16s/step - loss: 0.1233 - acc: 0.9681 - val_loss: 0.2005 - val_acc: 0.9500

Epoch 00006: val_acc did not improve from 0.96250
Epoch 7/100
50/50 [=====] - 762s 15s/step - loss: 0.1042 - acc: 0.9694 - val_loss: 0.1253 - val_acc: 0.9563

Epoch 00007: val_acc did not improve from 0.96250
Epoch 8/100
50/50 [=====] - 762s 15s/step - loss: 0.1124 - acc: 0.9688 - val_loss: 0.1662 - val_acc: 0.9344

Epoch 00008: val_acc did not improve from 0.96250
Epoch 00008: early stopping
```

Human Recognition Accuracy -

Accuracy : 99%

Images : 500

495 of 500 images were correctly classified.

Accuracy : 86.67%

Images : 15

13 of 15 images were correctly classified.

Meme Recognition Accuracy -

Accuracy : 96%

Images : 500

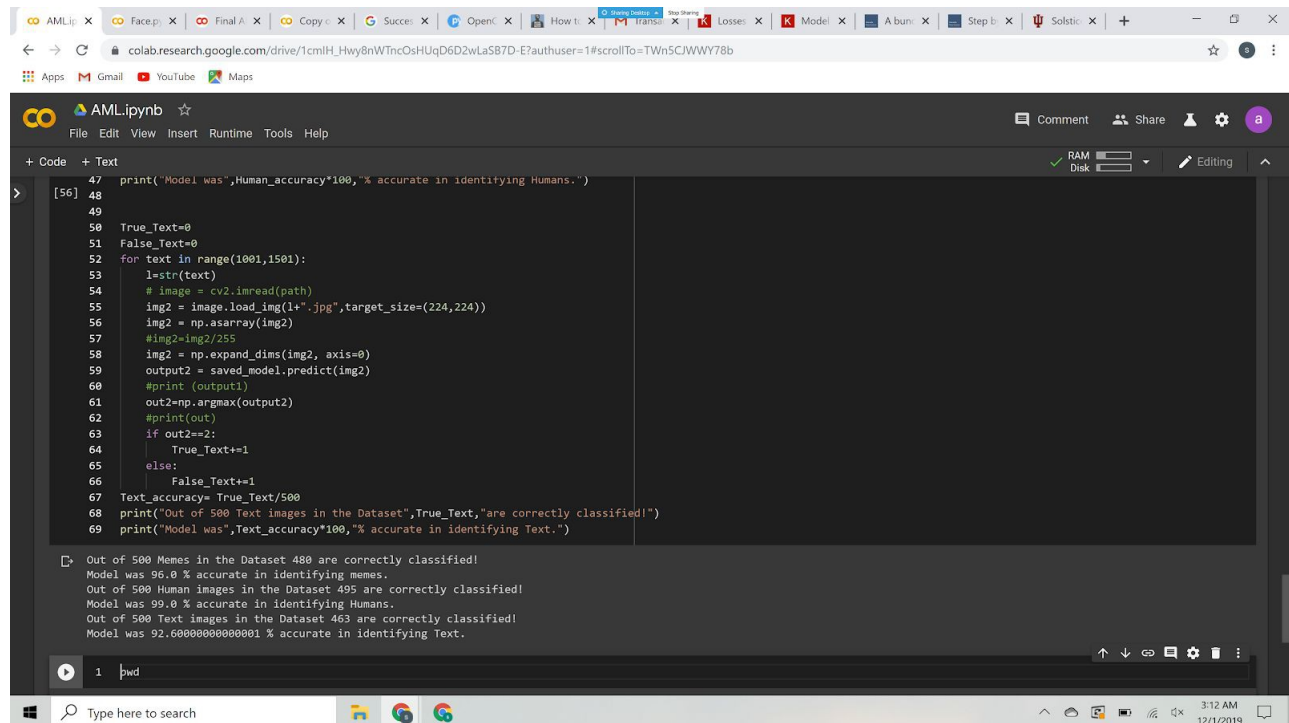
480 of 500 images were correctly classified.

Text Recognition Accuracy -

Accuracy : 92.6%

Images : 500

463 of 500 images were correctly classified.



The screenshot shows a Google Colab notebook titled 'AMLipynb'. The code in the cell iterates through 1500 images, classifying them as 'Human' or 'Text' based on a model's accuracy. The output shows that 489 out of 500 human images and 463 out of 500 text images were correctly classified, resulting in an overall accuracy of 95.8%.

```
47 print('Model was',Human_accuracy*100,"% accurate in identifying Humans.")
48
49
50 True_Text=0
51 False_Text=0
52 for text in range(1001,1501):
53     l=str(text)
54     # image = cv2.imread(path)
55     img2 = image.load_img(l+".jpg",target_size=(224,224))
56     img2 = np.asarray(img2)
57     #img2=img2/255
58     img2 = np.expand_dims(img2, axis=0)
59     output2 = saved_model.predict(img2)
60     #print (output1)
61     out2=np.argmax(output2)
62     #print(out)
63     if out2==2:
64         True_Text+=1
65     else:
66         False_Text+=1
67 Text_accuracy= True_Text/500
68 print("Out of 500 Text images in the Dataset",True_Text,"are correctly classified!")
69 print("Model was",Text_accuracy*100,"% accurate in identifying Text.")
```

Out of 500 Memes in the Dataset 489 are correctly classified!
Model was 95.8 % accurate in identifying memes.
Out of 500 Human images in the Dataset 495 are correctly classified!
Model was 99.0 % accurate in identifying Humans.
Out of 500 Text images in the Dataset 463 are correctly classified!
Model was 92.60000000000001 % accurate in identifying Text.

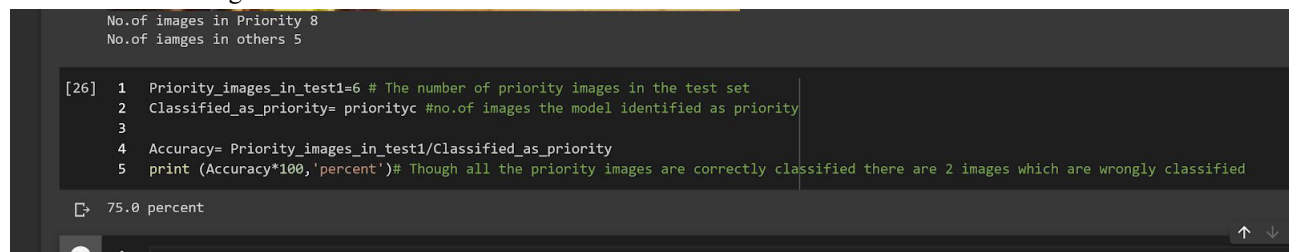
Face Recognition Accuracy -

Accuracy : 75%

Model detects all the priority images correctly with a few false detections.

True Positive: 6 images

False Positive: 2 images



The code cell calculates the accuracy of the model in identifying priority images. It shows that 6 out of 8 priority images were correctly classified, resulting in a 75.0 percent accuracy.

```
No.of images in Priority 8
No.of images in others 5

[26] 1 Priority_images_in_test1=6 # The number of priority images in the test set
      2 Classified_as_priority= priorityc #no.of images the model identified as priority
      3
      4 Accuracy= Priority_images_in_test1/Classified_as_priority
      5 print (Accuracy*100,'percent')# Though all the priority images are correctly classified there are 2 images which are wrongly classified
```

75.0 percent

Overall Accuracy -

Overall : 95.8%

Images : 1500

1438 of 1500 images were correctly classified.

6. References

- [1] Kaggle.com. (2019). *CelebFaces Attributes (CelebA) Dataset*. [online] Available at: <https://www.kaggle.com/jessicali9530/celeba-dataset>.
- [2] Medium. (2019). *Types of classification algorithms in Machine Learning*. [online] Available at: <https://medium.com/@Mandysidana/machine-learning-types-of-classification-9497bd4f2e14>.
- [3] Mathworks.com. (2019). *Pretrained VGG-16 convolutional neural network - MATLAB vgg16*. [online] Available at: <https://www.mathworks.com/help/deeplearning/ref/vgg16.html>.

[4] Medium. (2019). *Face detection with OpenCV and Deep Learning from image-part 1*. [online] Available at: <https://becominghuman.ai/face-detection-with-opencv-and-deep-learning-90b84735f421>.

[5] GitHub. (2019). *opencv/opencv*. [online] Available at: <https://github.com/opencv/opencv/wiki/Deep-Learning-in-OpenCV>.

[6] computervisiononline.com. (2019). *Georgia Tech Face Database | Computer Vision Online*. [online] Available at: <https://computervisiononline.com/dataset/1105138700> [Accessed 2 Dec. 2019].

[7] GitHub. (2019). *hardikvasa/google-images-download*. [online] Available at: <https://github.com/hardikvasa/google-images-download>.

[8] Fki.inf.unibe.ch. (2019). *IAM Handwriting Database — Computer Vision and Artificial Intelligence*. [online] Available at: <http://www.fki.inf.unibe.ch/databases/iam-handwriting-database> [Accessed 2 Dec. 2019].

[9] GitHub. (2019). *rarcega/instagram-scraper*. [online] Available at: <https://github.com/rarcega/instagram-scraper> [Accessed 2 Dec. 2019].