Face Mask Detection Report

Problem Statement

During Covid19 it become tough for the organizations and public gathering places to identify the people who are no wearing the masks. If we employee people its costs more, not much efficient moreover. its risk for the employee as well. The model should be efficient that it should identify many people at a time whether they are wearing the mask or not.

Our Goal is to build a Face Mask detection Model using Convolution Neural Networks. Our model will detect group of people at a time and says whether they are wearing mask or not. By the end of this project, we should be able to detect the people whether they are wearing the masks are not.

Dataset

For all the experiments and evaluations performed in this project, we have used the Face Mask Images dataset from Kaggle (Face Mask Detection ~12K Images Dataset | Kaggle). It will used for supervised binary classification tasks. Total number of images in the dataset is 12k that split into three folder training, testing and validation.

Exploratory Data Analysis

Exploratory data analysis is performed to gain different useful information and hidden insights from dataset. In this section different statistical techniques have been used to gain insights and then being visualized into appropriate charts and plots.

```
# Map the images from train folder with train labels to form a DataFrame
def get_all_images_from_subdirectory_to_dataframe(path):
  configfiles = [os.path.join(dirpath, f)
      for dirpath, dirnames, files in os.walk(path)
      for f in files if f.endswith('.png')]
  images_list = [(i.split("/")[-2],i.split("/")[-1], i) for i in configfiles]
  return images_list
```

From above function, we will traverse all the images from each directory (train, test and val) and store into a list with class (mask or without mask), image name and image path. After that we will show in pandas dataframe. The dataframe representing this information is shown below.

```
df_list = get_all_images_from_subdirectory_to_dataframe(train_dir)
df_train = pd.DataFrame(data=df_list, columns=['class', 'image_name', 'image_path'])

df_list = get_all_images_from_subdirectory_to_dataframe(test_dir)
df_test = pd.DataFrame(data=df_list, columns=['class', 'image_name', 'image_path'])

df_list = get_all_images_from_subdirectory_to_dataframe(val_dir)
df_val = pd.DataFrame(data=df_list, columns=['class', 'image_name', 'image_path'])
```

	class	image_name	image_path
0	WithoutMask	2083.png	/content/Face Mask Dataset/Train/WithoutMask/2...
1	WithoutMask	4529.png	/content/Face Mask Dataset/Train/WithoutMask/4...

From above analysis, we can see that our dataframe consist into three columns class, image name and image path.

Plot Random Images

This analysis is about plotting the random images from dataset. This function is shown below plot the random images from specific dataset folder (train, test and val) with class name (mask or without mask).

```
# Write a function that will select n random images and display images along with its species
def plot_random_images(df, total_image=2):
    import matplotlib.image as mpimg
    fig, axes = plt.subplots(1, total_image,figsize=(14,2))
    images_data = list(zip(df['image_path'],df['class']))
    samples = random.sample(images_data,total_image)
    for ax, (image, label) in zip(axes, samples):
        image = mpimg.imread(image)
        ax.set_axis_off()
        ax.imshow(image, cmap = 'binary')
        ax.set_title(f'{label}')
```

```
print("Traning Images")
plot_random_images(df_train, 5)
```

```
Traning Images
```

```
print("Testing Images")
plot_random_images(df_test, 5)
```

```
Testing Images
```

```
print("Validation Images")
plot_random_images(df_val, 5)
```

```
Validation Images
```

From above analysis, we plot the random images with class from each folder.

Class Distribution (Mask or Without Mask)

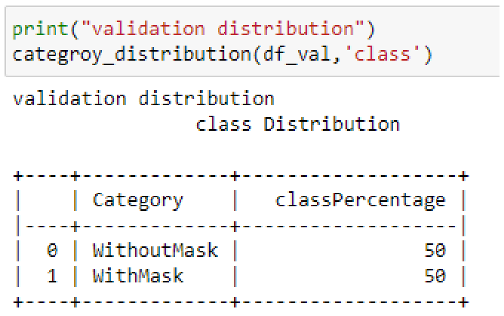
This analysis is about the distribution of class. The table representing this information is shown below.

```
print("training distribution\n")
categroy_distribution(df_train,'class')
```

```
training distribution

                      class Distribution

+----+-------------+------------------+
|    | Category    |   classPercentage |
|----+-------------+------------------|
|  0 | WithoutMask |               50 |
|  1 | WithMask    |               50 |
+----+-------------+------------------+
```

```
print("testing distribution")
categroy_distribution(df_test,'class')
```

```
testing distribution
                      class Distribution

+----+-------------+------------------+
|    | Category    |   classPercentage |
|----+-------------+------------------|
|  0 | WithoutMask |            51.31 |
|  1 | WithMask    |            48.69 |
+----+-------------+------------------+
```



```
print("validation distribution")
categroy_distribution(df_val,'class')
```

```
validation distribution
                 class Distribution

+----+-------------+-------------------+
|    | Category    |   classPercentage |
|----+-------------+-------------------|
|  0 | WithoutMask |                50 |
|  1 | WithMask    |                50 |
+----+-------------+-------------------+
```

From above table it is clear that the mask and without mask images are balanced form (50:50).

<u>Model Building</u>

In this project, we will use the three pre-trained model on dataset and choose the best model that will give a best performance metrics. These three pre-trained model are given below:

<u>VGG16</u>

VGG-16 is a convolutional neural network that is 16 layers deep. We can load a pretrained version of the network trained on more than a million images from the ImageNet database [1]. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. We will freeze all the top layers and embed the last layer according to our classes that actually two.

<u>MobileNetV2</u>

MobileNet-v2 is a convolutional neural network that is 53 layers deep. We can load a pretrained version of the network trained on more than a million images from the ImageNet database [1]. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. We will freeze all the top layers and embed the last layer according to our classes that actually two.

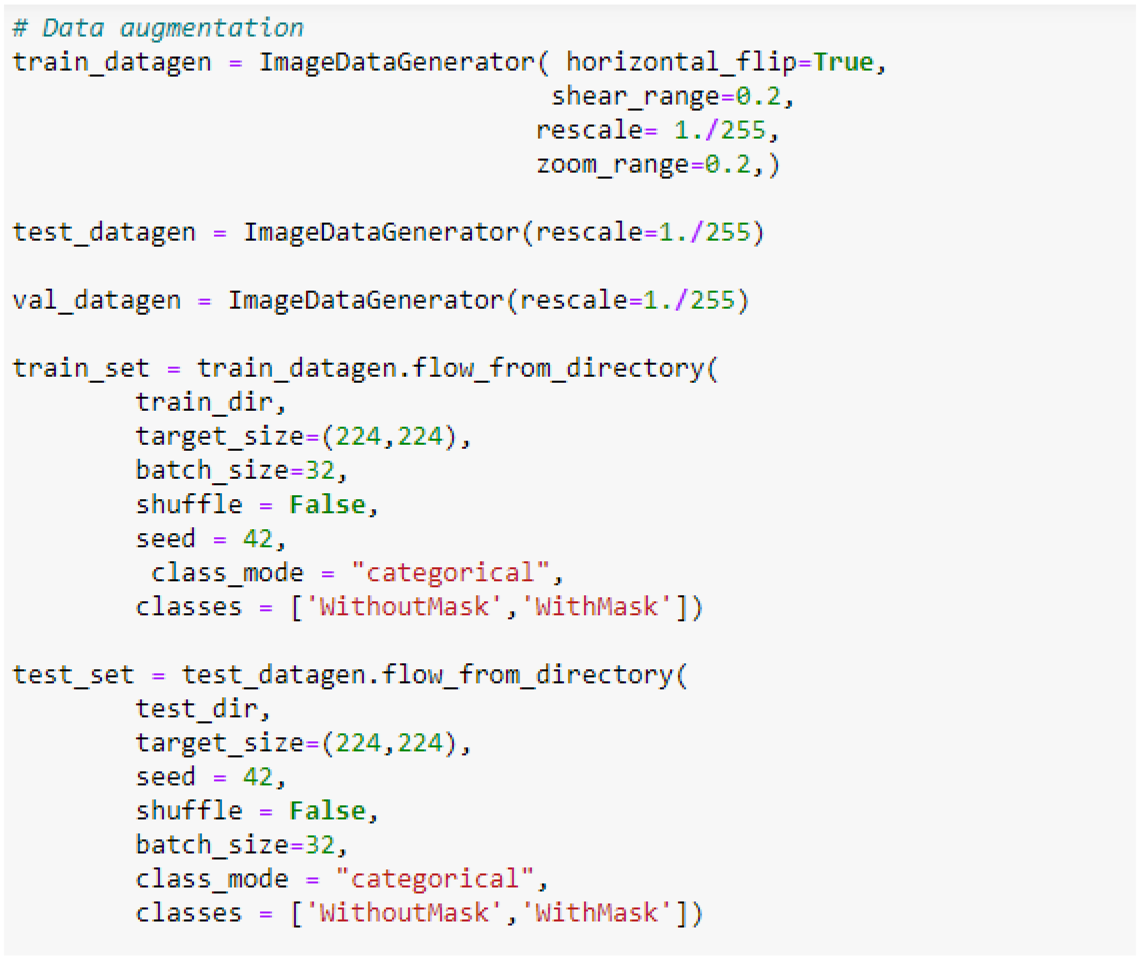
<u>VGG19</u>

VGG-19 same as VGG-16. VGG-19 is a convolutional neural network that is 19 layers deep. We can load a pretrained version of the network trained on more than a million images from the ImageNet database [1]. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. We will freeze all the top layers and embed the last layer according to our classes that actually two.

<u>Data Augmentation</u>

Data Augmentation in play. A convolutional neural network that can robustly classify objects even if its placed in different orientations is said to have the property called invariance. More specifically, a CNN can be invariant to translation, viewpoint, size or illumination (Or a combination of the above).

We have a directory in which two folder mask and without mask. Therefore, we will use the tensorflow provide flow from directory method. The flow_from_directory() method takes a path of a directory and generates batches of augmented data. The directory structure is very important when you are using flow_from_directory() method. The flow_from_directory() assumes: The root directory contains at least two folders one for train and one for the test.



```
# Data augmentation
train_datagen = ImageDataGenerator( horizontal_flip=True,
                                   shear_range=0.2,
                                  rescale= 1./255,
                                  zoom_range=0.2,)

test_datagen = ImageDataGenerator(rescale=1./255)

val_datagen = ImageDataGenerator(rescale=1./255)

train_set = train_datagen.flow_from_directory(
        train_dir,
        target_size=(224,224),
        batch_size=32,
        shuffle = False,
        seed = 42,
         class_mode = "categorical",
        classes = ['WithoutMask','WithMask'])

test_set = test_datagen.flow_from_directory(
        test_dir,
        target_size=(224,224),
        seed = 42,
        shuffle = False,
        batch_size=32,
        class_mode = "categorical",
        classes = ['WithoutMask','WithMask'])
```

Evaluation Metrics

State of the art evaluation metrics for supervised binary classification problems are given below:

- **Confusion Matrix**

In machine learning algorithm, confusion matrix is a performance measurement. A confusion matrix is a summarized table of the number of correct and incorrect predictions (or actual and predicted values).

	Actual Positive	*Actual Negative*
Predicted Positive	➢ **TP**	➢ **FP**
Predicted Negative	➢ **FN**	➢ **TN**

$$\begin{gathered} where\ TP = True\ Positive \\ FP = False\ Positive \\ TN = True\ Negative \\ FN = False\ Negetive \end{gathered}$$

True Positive (TP): where the model correctly predicts the positive class.
True Negative (TN): where the model correctly predicts the negative class.
False Positive (FP): where the model incorrectly predicts the positive class.
False Negative (FN): where the model incorrectly predicts the negative class.

- **Accuracy**

Accuracy represents the correctly predict both the positives and negatives out of all the predictions.

Measure to evaluate how accurate model's performance is:

$$\frac{TP + TN}{TP + FP + FN + FP}$$

- **Precision**

Precision represents the correctly predict the positives out of all the positive prediction it made.

Measure to evaluate how accurate model's performance is:

$$\frac{TP}{TP + FP}$$

- **Recall**

Recall represents the correctly predict the positives out of actual positives.

Measure to evaluate how accurate model's performance is:

$$\frac{TP}{TP + FN}$$

- **F_1**

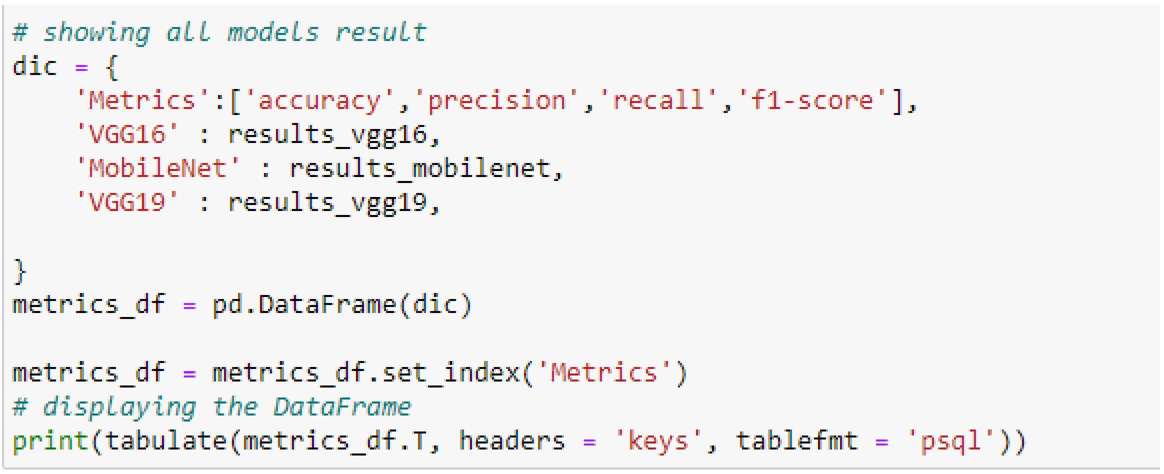
F1 is a combination of both precision and recall.

Provides information of both sides TN and TP

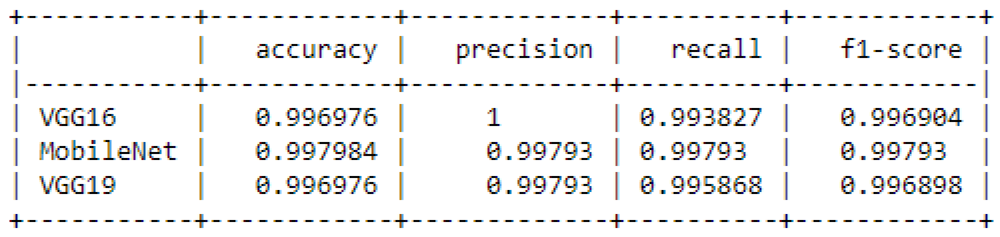
$$2 * \frac{Precision * Recall}{Precision + Recall}$$

<u>Results</u>

This section delves into the performance of all deep learning models on our dataset and conducts a systematic comparative analysis to determine which model is the best. Table depicts the result of all models on testing data. It shows that all models performing well on unseen data. Therefore, we will use the MobileNetV2 mode because its not too large model as compare to VGG-16 and 19.



```python
# showing all models result
dic = {
    'Metrics':['accuracy','precision','recall','f1-score'],
    'VGG16' : results_vgg16,
    'MobileNet' : results_mobilenet,
    'VGG19' : results_vgg19,

}
metrics_df = pd.DataFrame(dic)

metrics_df = metrics_df.set_index('Metrics')
# displaying the DataFrame
print(tabulate(metrics_df.T, headers = 'keys', tablefmt = 'psql'))
```



```
+-----------+------------+-------------+----------+------------+
|           |   accuracy |   precision |   recall |   f1-score |
|-----------+------------+-------------+----------+------------|
| VGG16     |   0.996976 |     1       | 0.993827 |   0.996904 |
| MobileNet |   0.997984 |     0.99793 | 0.99793  |   0.99793  |
| VGG19     |   0.996976 |     0.99793 | 0.995868 |   0.996898 |
+-----------+------------+-------------+----------+------------+
```

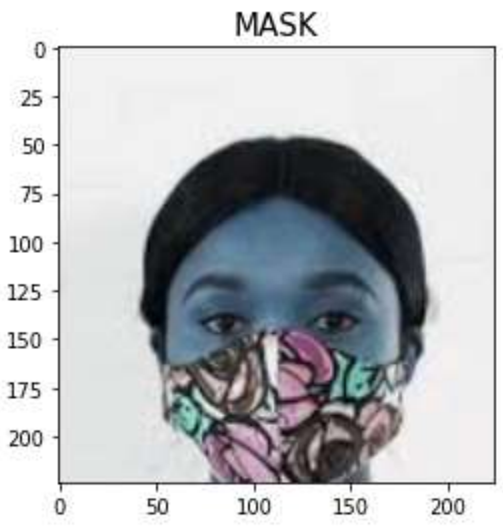
<u>Save & Load Model</u>

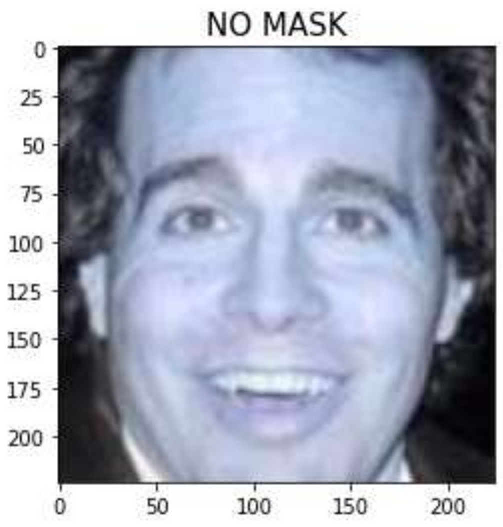
This section we will save the model weights. The weights are saved directly from the model using the save_weights() function and later loaded using the symmetrical load_weights() function. The model is then converted to JSON format and written to model. json in the local directory.

<u>Model Prediction</u>

Now, we will predict the the images from dataset through model that we have already trained and stored. From below we can see that our model predicting correctly.

```python
def image_prediction(path):
    mask_label = {0:'NO MASK',1:'MASK'}
    sample_mask_img = cv2.imread(path)
    img = cv2.resize(sample_mask_img,(224,224))
    sample_mask_img = np.reshape(img,[1,224,224,3])
    sample_mask_img = sample_mask_img/255.0
    label = np.argmax(loaded_model.predict(sample_mask_img, verbose=0))
    plt.imshow(img)
    plt.title(mask_label[label], fontsize=15)
    plt.show()
```

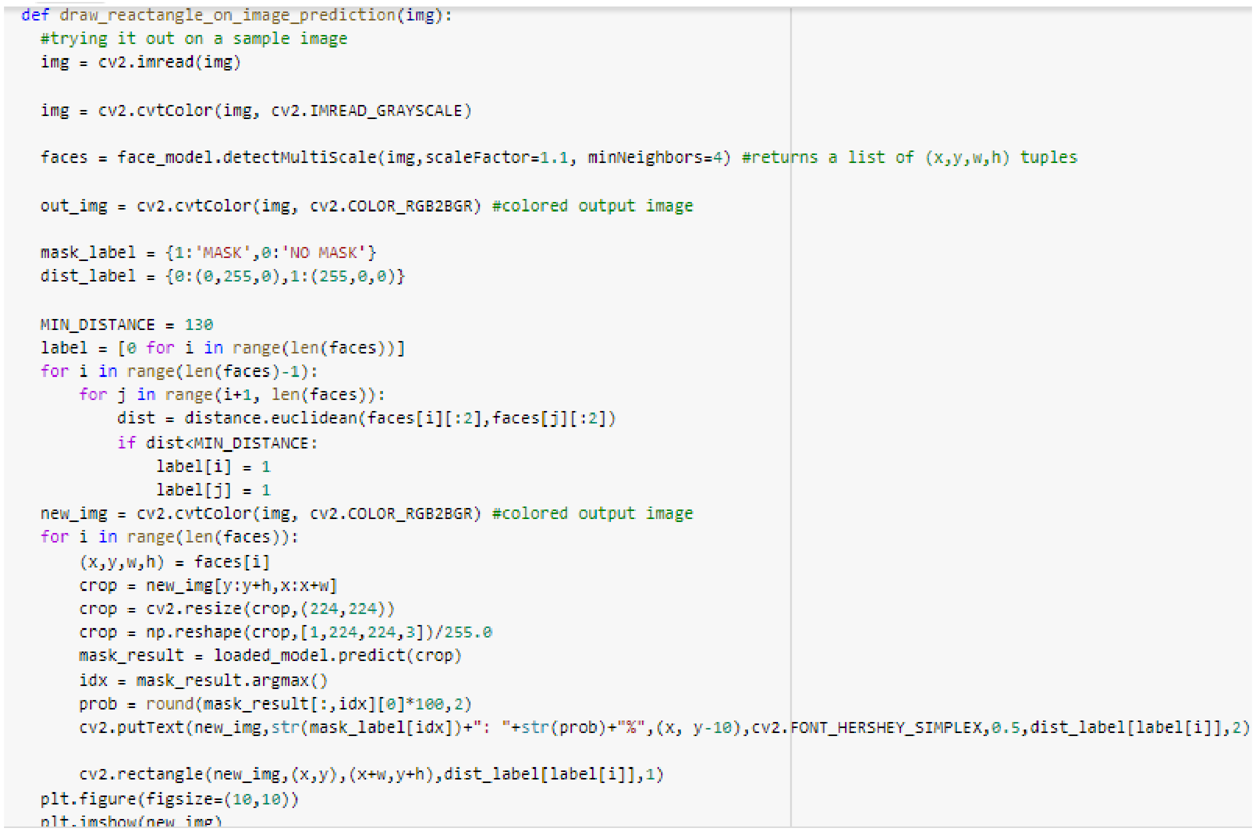




Detect & Predict Multi-Face from images

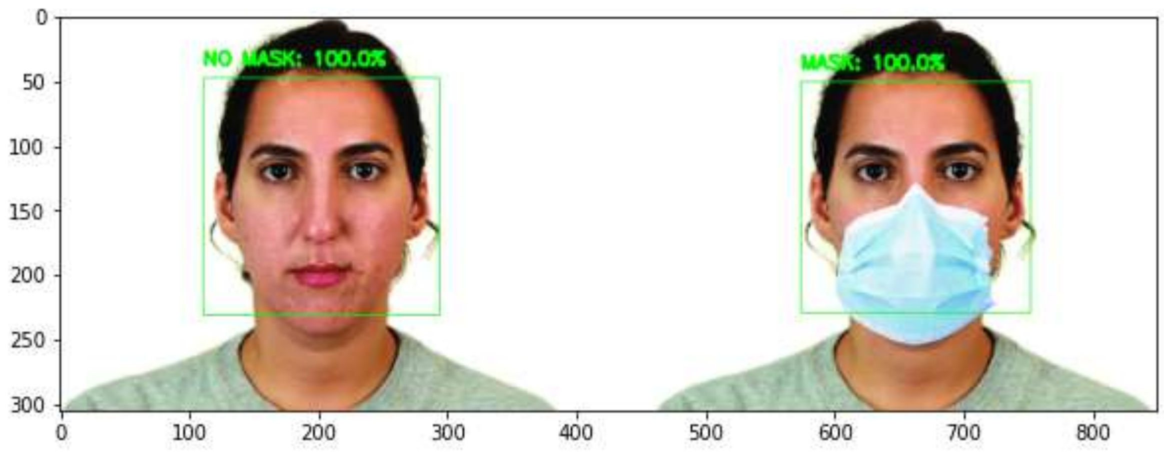
Now, we will use the harcasscade for detect the multi face from images and predict the each face through our model MobileNetV2. We will also draw a text and rectangle on face with the help of opencv library.

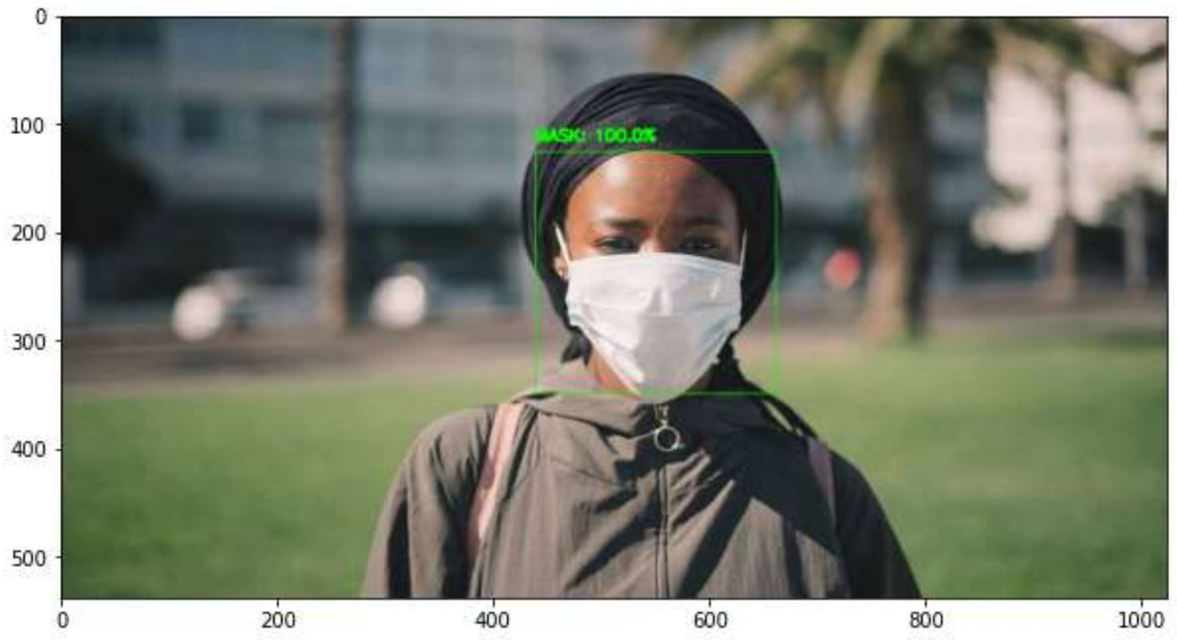
Haar Cascade is an Object Detection Algorithm used to identify faces in an image or a real time video. The algorithm uses edge or line detection features proposed by Viola and Jones in their research paper “Rapid Object Detection using a Boosted Cascade of Simple Features” published in 2001.



```
def draw_reactangle_on_image_prediction(img):
  #trying it out on a sample image
  img = cv2.imread(img)

  img = cv2.cvtColor(img, cv2.IMREAD_GRAYSCALE)

  faces = face_model.detectMultiScale(img,scaleFactor=1.1, minNeighbors=4) #returns a list of (x,y,w,h) tuples

  out_img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR) #colored output image

  mask_label = {1:'MASK',0:'NO MASK'}
  dist_label = {0:(0,255,0),1:(255,0,0)}

  MIN_DISTANCE = 130
  label = [0 for i in range(len(faces))]
  for i in range(len(faces)-1):
      for j in range(i+1, len(faces)):
          dist = distance.euclidean(faces[i][:2],faces[j][:2])
          if dist<MIN_DISTANCE:
              label[i] = 1
              label[j] = 1
  new_img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR) #colored output image
  for i in range(len(faces)):
      (x,y,w,h) = faces[i]
      crop = new_img[y:y+h,x:x+w]
      crop = cv2.resize(crop,(224,224))
      crop = np.reshape(crop,[1,224,224,3])/255.0
      mask_result = loaded_model.predict(crop)
      idx = mask_result.argmax()
      prob = round(mask_result[:,idx][0]*100,2)
      cv2.putText(new_img,str(mask_label[idx])+": "+str(prob)+"%",(x, y-10),cv2.FONT_HERSHEY_SIMPLEX,0.5,dist_label[label[i]],2)

      cv2.rectangle(new_img,(x,y),(x+w,y+h),dist_label[label[i]],1)
  plt.figure(figsize=(10,10))
  plt.imshow(new_img)
```

In this above function, we will read the images and then detect the faces from haarcascade and predict the each face through our trained model MobileNetV2. From below we can see the results of face detection and classify the mask or without mask faces.




MASK: 100.0%
0
100
200
300
400
500
0
200
400
600
800
1000