Face Mask Detection Report

Yashwanth Telukuntla(ytelukuntla@albany.edu)

Prasanth Nagisetty(pnagisetty@albany.edu)

S R K N S Bharadwaj Nidamarthy(snidamarthy@albany.edu)

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.

As we use Google Colab to build our Facemask Classification model (FmCM), we needs to download and store this dataset into Google Drive

- Connect to Google Drive from Google Colab:

```
from google.colab import drive
drive.mount('/content/drive')
```

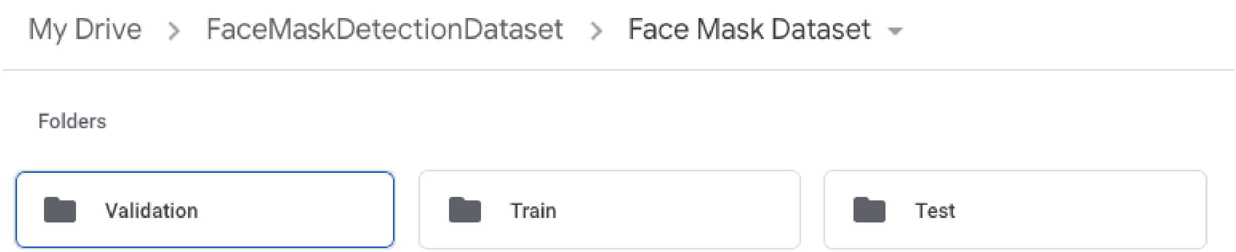
- Unzip dataset (only run one time)

```
import os
os.environ['KAGGLE_CONFIG_DIR'] =
"/content/drive/MyDrive/NewFacemaskDetection"
%cd /content/drive/MyDrive/NewFacemaskDetection/
```

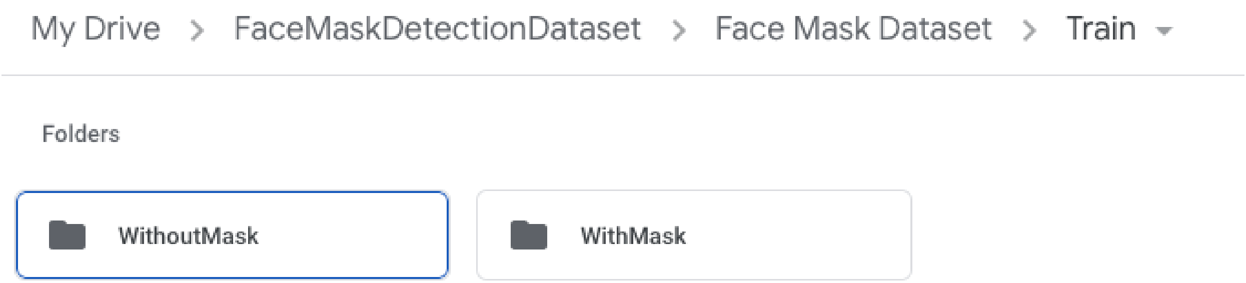
```
!unzip ./face-mask-12k-images-dataset.zip
```

>>> After unzip, the dataset folder has 3 sub-folders: Train, Test and Validation. Each sub-folder has 2 sub-sub-folders: WithMask and WithoutMask folders which contain images belonging ‘Mask’ and ‘No Mask’ categories respectively.

sub-folders

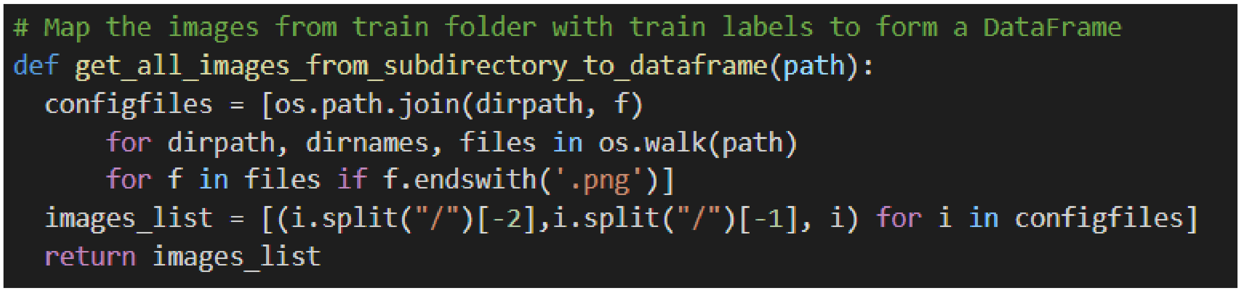


sub-sub-folders


My Drive > FaceMaskDetectionDataset > Face Mask Dataset > Train
Folders
WithoutMask
WithMask

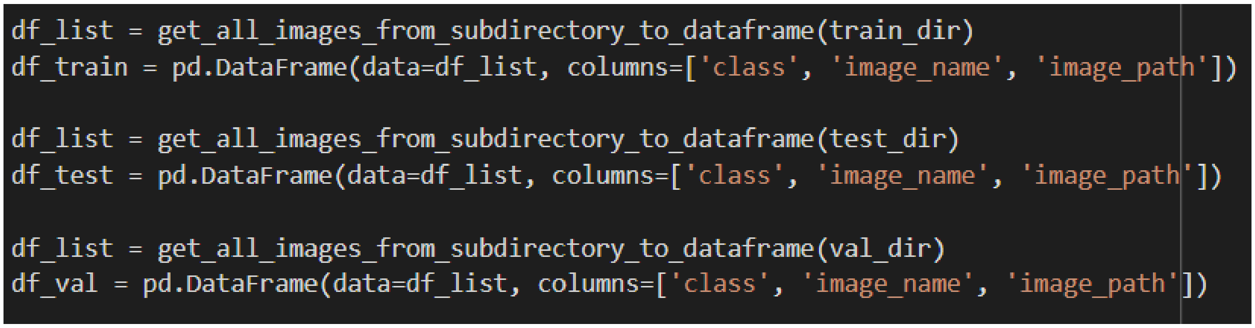
Exploratory Data Analysis

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.

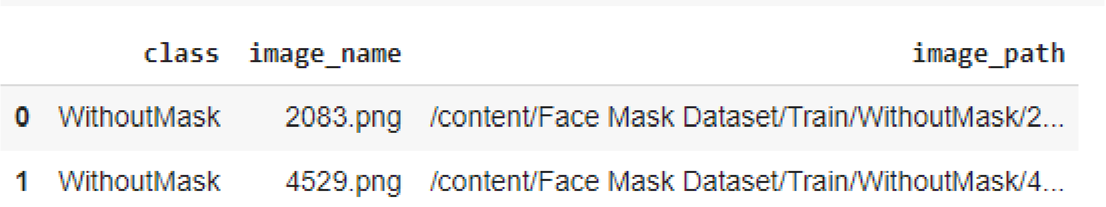


```
# Map the images from train folder with train labels to form a DataFrame
def get_all_images_from_subdirectory_to_dataframe(path):
  configfiles = [os.path.join(dirpath, f)
      for dirpath, dirnames, files in os.walk(path)
      for f in files if f.endswith('.png')]
  images_list = [(i.split("/")[-2],i.split("/")[-1],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

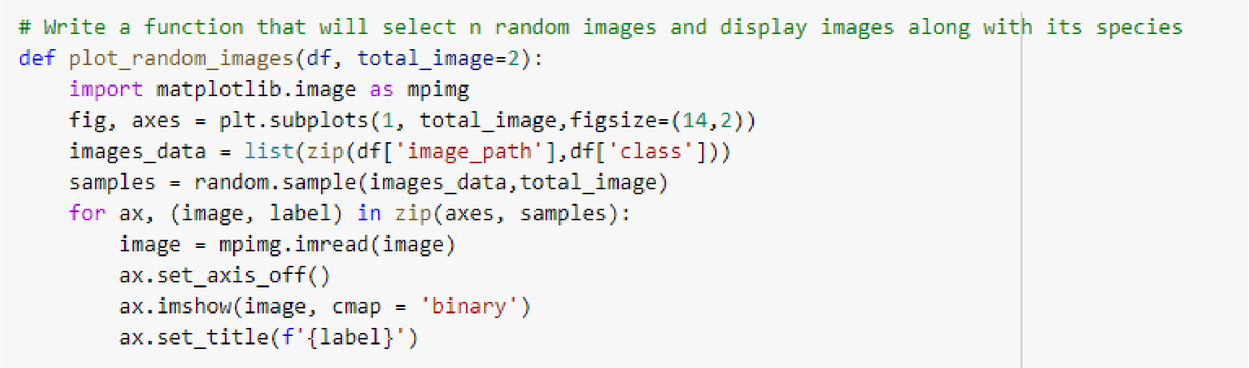


```
         class  image_name                                     image_path
0  WithoutMask    2083.png  /content/Face Mask Dataset/Train/WithoutMask/2...
1  WithoutMask    4529.png  /content/Face Mask Dataset/Train/WithoutMask/4...
```

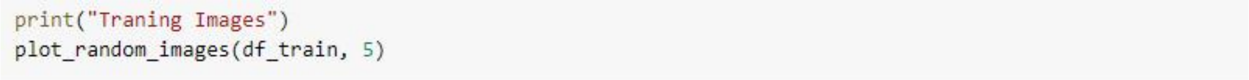
From above analysis, we can see that our dataframe consist into three columns class, image name and image path.

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).

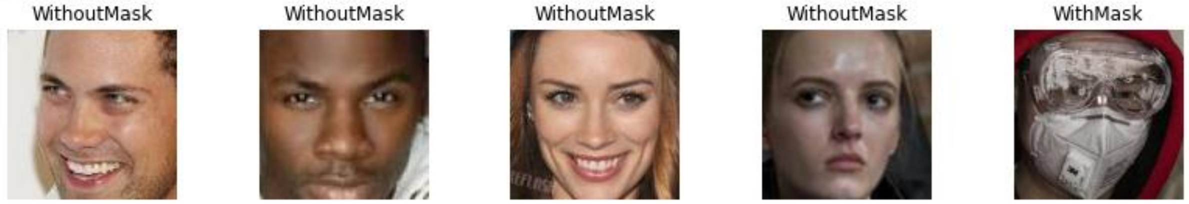


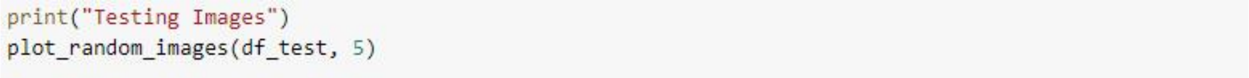
```python
# Write a function that will select n random images and display images along with its species
def plot_random_images(df, total_image=2):
    import matplotlib.image as mpimg
    fig, axes = plt.subplots(1, total_image,figsize=(14,2))
    images_data = list(zip(df['image_path'],df['class']))
    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}')
```



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

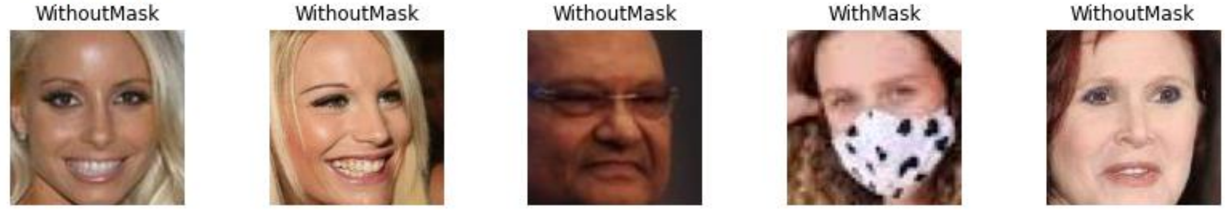
```
Traning Images
```

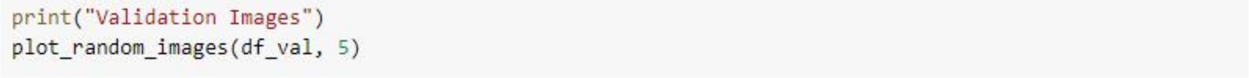




```python
print("Testing Images")
plot_random_images(df_test, 5)
```

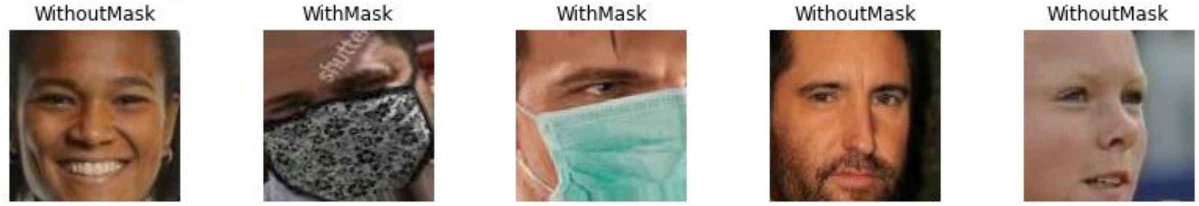
```
Testing Images
```





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

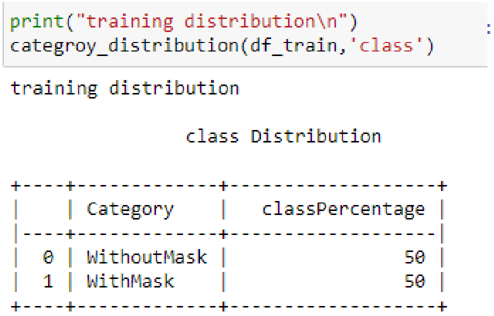
```
Validation Images
```



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

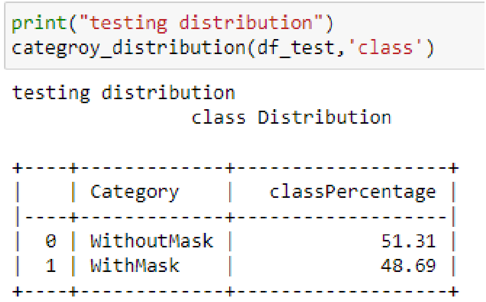
Class Distribution (Mask or Without Mask)

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



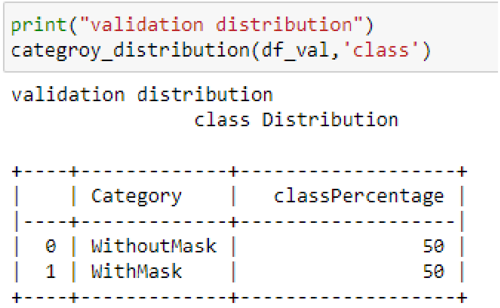
```
print("training distribution\n")
categroy_distribution(df_train,'class')
```

```
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')
```

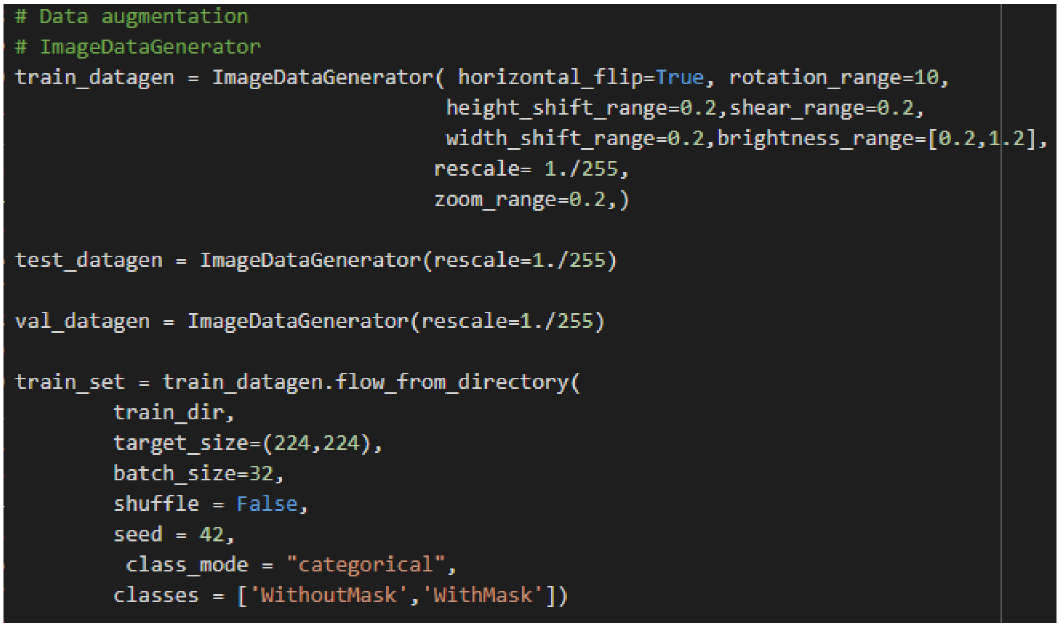
```
validation distribution
                        class Distribution

+----+------------+------------------+
|    | 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).

Data Augmentation

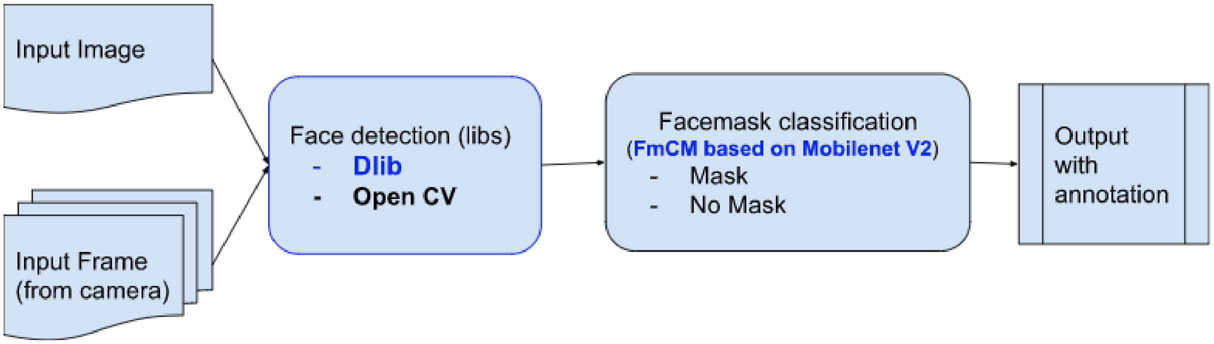
Data Augmentation in play. A convolutional neural network that can robustly classify objects even if its placed in different orientations is said to have the property called invariance. More specifically, a CNN can be invariant to translation, viewpoint, size or illumination (Or a combination of the 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
# ImageDataGenerator
train_datagen = ImageDataGenerator( horizontal_flip=True, rotation_range=10,
                                   height_shift_range=0.2,shear_range=0.2,
                                   width_shift_range=0.2,brightness_range=[0.2,1.2],
                                  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'])
```

FACEMASK DETECTION PROCESS



1. **Face detection overview**: There are many libs that support face detection
    a. **OpenCV** (Haar-Cascade or **Deep neural network**)

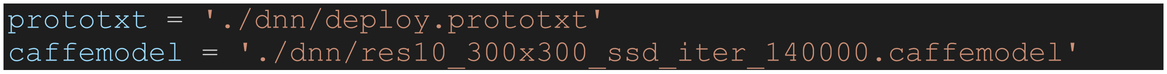
    >>> **Using Haar-Cascade:** use Haar cascade algorithm to detect objects in images. We can use a pre-trained cascade (xml file) to detect frontal faces. From experimental results, this method is a lightweight face detector: model size is around 930KB, fast detection. However, it returns many ‘false positive’ detections, i.e. return ‘face’ that is actually not a face. This method is recommended to use on devices with resource limitation such as phones, tablets.

>>>>> usage: face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

>>> **Deep neural network:** We use Open CV's deep neural network model to detect face. The model includes a **.prototxt** file which defines the model architecture (layers of DNN) and a **.caffemodel** which contains the pre-trained weights of the DNN. It is based on Single-Shot-Multibox detector and uses ResNet-10 Architecture as backbone. The model was trained using images available from the web, but the source is not disclosed.

>>>>> usage: dnn_net = cv2.dnn.readNetFromCaffe(facenet_dnn.prototxt, facenet_dnn.caffemodel)

>>>>> reference: https://github.com/opencv/opencv/blob/3.4.0/samples/dnn/resnet_ssd_face_python.py



```
prototxt = './dnn/deploy.prototxt'
caffemodel = './dnn/res10_300x300_ssd_iter_140000.caffemodel'
```

b. **Dlib:** is used widely in image synthesis applications such as: face reconstruction, style transfer. **Dlib** this library offers two methods for face detection:

>>> **HOG and Linear SVM**: This method is based on histogram of oriented gradients and linear support vector machine. It provides a very fast recognition result of frontal faces but has limited recognition on side-profile faces This method is suitable for the case in which we can get the direct view of the face (ID systems). This method is better than Haar-cascade (OpenCV) but need more computational power.

>>>>>usage: dlib.get_frontal_face_detector()

>>> **Max-Margin Mod Object Detection (MNMOD) CNN** face detector: this method has capability of capturing face(s) at angles. However, this model needs GPU or will be super slow in CPU.

>>>>>usage: dlib.cnn_face_detection_model_v1(CNN model)

>>>> CNN model: mmod_human_face_detector.dat

2. **FACEMASK Classification model (FmCM)**: we use the output of face detection process (Dlib) as the input of our FmCM. Face detection module

(Dlib/OpenCV) return a list of coordinates of face(s). With Open CV DNN, it also returns the confidence, we only accept confidence >0.5. The return data includes (x1,y1) of the Left-Bottom point and (x2,y2) of the Top-Right point. We then need to get the region of interest (ROI) data from the image/frame. The ROI then will be reshaped into **(1,224,224,3)** as the required input of FmCM.

1. **Transform the output of Dlib CNN/OpenCV DNN into input for FmCM**

>>> Dlib CNN ⇒ FmCM:

```
        coordinate_list = dlib_detector(gray)
        for i, face in enumerate(coordinate_list):
            x1,y1,x2,y2 = face.rect.left(), face.rect.top(),face.rect.right(),face.rect.bottom()
            ROI = frame[y1:y2, x1:x2]
            ROI = imutils.resize(ROI, height=224 )
            ROI = np.reshape(ROI, (1, 224, 224,3))

            cv.rectangle(frame,(x1,y1), (x2, y2), (0,255,0), 2)
            cv.putText(frame,title, (x1,y1), cv.FONT_HERSHEY_COMPLEX,1,(244,250,250),2)
```

>>> OpenCV DNN ⇒ FmCM:

```
    for i in range(detections.shape[2]):
        confidence = detections[0, 0, i, 2]
        if confidence > confThreshold:  # if confidence <0.5, ignore
            xLeftBottom = int(detections[0, 0, i, 3] * cols)
            yLeftBottom = int(detections[0, 0, i, 4] * rows)
            xRightTop  = int(detections[0, 0, i, 5] * cols)
            yRightTop  = int(detections[0, 0, i, 6] * rows)
            x1,x2,y1,y2= xLeftBottom,xRightTop,yLeftBottom,yRightTop
            ROI = frame[y1:y2, x1:x2]
            ROI = imutils.resize(ROI, width = 224 )
            ROI = np.reshape(ROI, (1, 224, 224,3))
```

b. **Predict (classify): MASK/ NO MASK**

>>> the return model includes two values [no_mask_value, mask_value], we use np.argmax to select the index of the larger (max) value ⇒ returns 0 or 1. 0 means 'NO MASK' and 1 means 'MASK': **mask_label = {0:'NO MASK',1:'MASK'}**

```
             #LOAD FmCM model
             json_file = open("FaceMaskDetection_model_file_MobileNet.json", 'r')
             loaded_model_json = json_file.read()
             json_file.close()
             loaded_model = model_from_json(loaded_model_json)
             # load weights into new model
             loaded_model.load_weights("FaceMaskDetection_model_file_MobileNet.h5")
             print("Loaded model from disk")
             mask_label = {0:'NO MASK',1:'MASK'}
            #PREDICT
            face_mask = np.reshape(ROI,[1,224,224,3])
            face_mask = face_mask/255.0
            label = np.argmax(loaded_model.predict(face_mask, verbose=0))
            title= mask_label[label]
```

c. Add result of FmCM back to the original image/video

```
          labelSize, baseLine = cv.getTextSize(title, cv.FONT_HERSHEY_SIMPLEX, 0.5, 1)
          cv.rectangle(frame, (x1, y1), (x2, y2),(0, 255, 0))
          cv.rectangle(frame, (x1, y1 - labelSize[1]),(x1 + labelSize[0], y1 +
baseLine),(255, 255, 255), cv.FILLED)
          cv.putText(frame, label, (x1, y1), cv.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0))
```

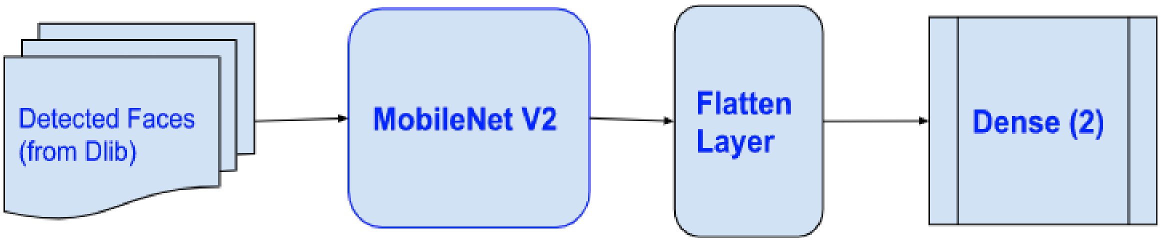
FACEMASK DETECTION MODEL

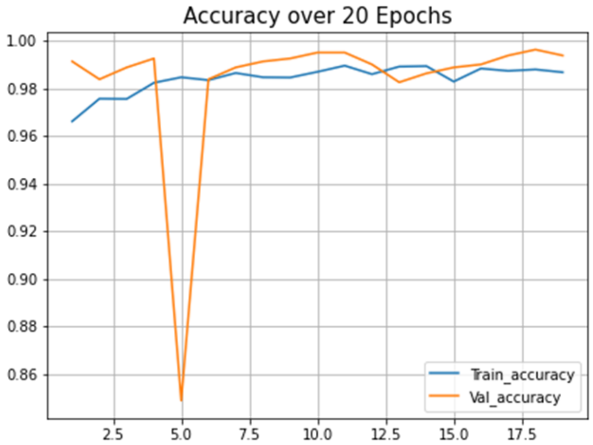
The original model

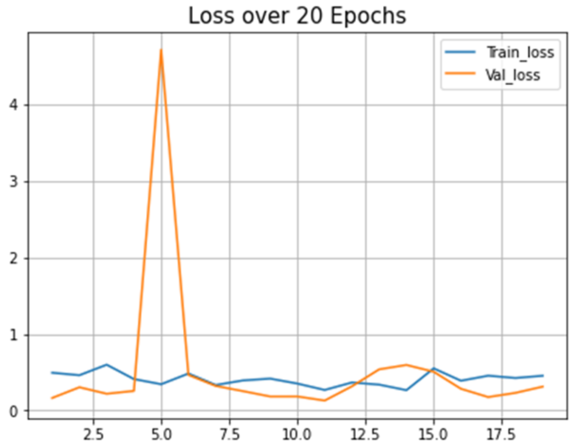
Model we used earlier.

```
model = Sequential()
model.add(pretrained_model)
model.add(Flatten())
model.add(Dense(2,activation='softmax'))
```

>>> **Problem with this model:** Training problem of FmCM after 10 epochs, training_loss keeps above 0.4 and does not reduce.



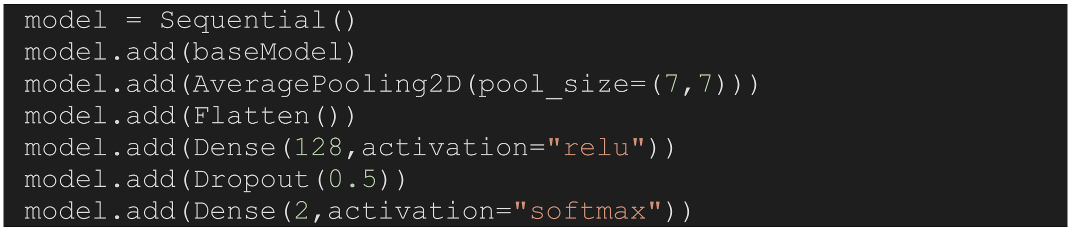




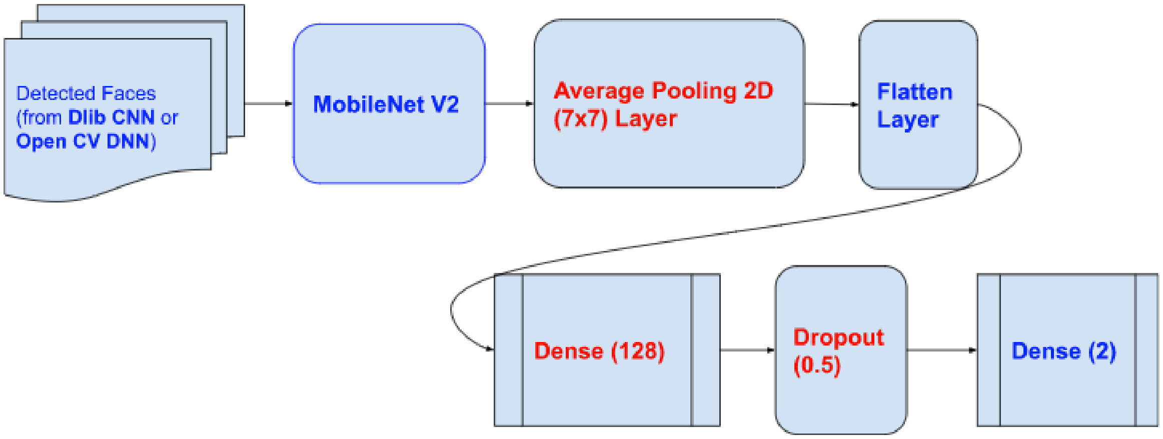
>>> **Solution**: adding Average Pooling 2D and dropout layers (described in the below section)

The new FaceMask Detection model:

- added an **Average Pooling2D** ⇒ after a CNN network, it's recommended to use an Average pooling layer to downsample the input (i.e.. output of CNN)
- add a **dense(128)** and a **dropout(0.5)** layer : The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting. Inputs not set to 0 are scaled up by 1/(1 - rate) such that the sum over all inputs is unchanged.



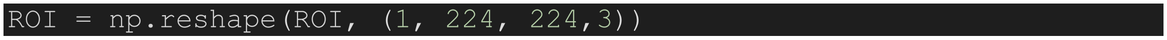
```
model = Sequential()
model.add(baseModel)
model.add(AveragePooling2D(pool_size=(7,7)))
model.add(Flatten())
model.add(Dense(128,activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(2,activation="softmax"))
```



Detected Faces (from Dlib CNN or Open CV DNN)
MobileNet V2
Average Pooling 2D (7x7) Layer
Flatten Layer
Dense (128)
Dropout (0.5)
Dense (2)

Model explanation

>>>Detected faces module (left most module): Input image/ frame captured from Dlib ⇒ location = (x1,y1, x2,y2)
(Bottom left point x1,y1 and top right point x2,y2) ⇒ get ROI (region of interest) of the detected face ⇒ resize to 224* 224 and reshape to **(1,224,224,3)** as the input of MobileNet V2.

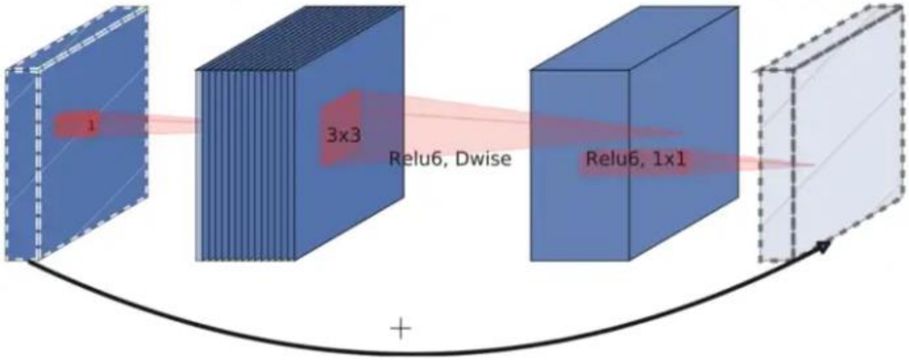


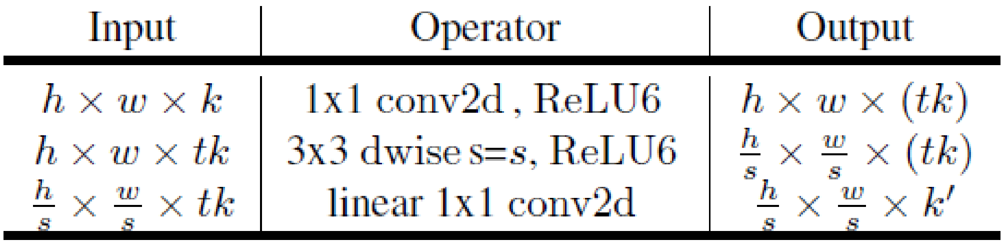
```
ROI = np.reshape(ROI, (1, 224, 224,3))
```

>>>MobileNet V2 module: In general, MobileNet V2 has 3 blocks as below

- **The first block** is a Convolution 1x1 with ReLU6 activation (ReLU6: x< 0 → x= 0, >=0 → in range [0,6]) which has input **shape (32, 224, 224,3)**
    - **3 = RGB (8 bits each),**

  - **The first number (32) is the batch size.** When we print out the summary of the model, this number is **None).**
  - The output of the first block is expanded with expansion factor t. And t=6 for all main experiments ⇒ **output = (32, 224, 224,3*6)**
- **The second block** is a depthwise convolution with 3*3 dwise and s= 2 and use ReLU6 activation. Input= output of the first block **= (32, 224, 224,3)**. The output of this block i = **(32, 112, 112,3*6)**
- **The third block** is also a Convolution 1x1 with linear activation (not ReLU6). It has **input = output of the 2nd block = (32, 112, 112,3*6** The output of this block is **(32, 7, 7,1280). Actually, there are many layers in the second block which expand/reduce the number of channels and the final is 1280.**





Input	Operator	Output
$h \times w \times k$	1x1 conv2d , ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3x3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

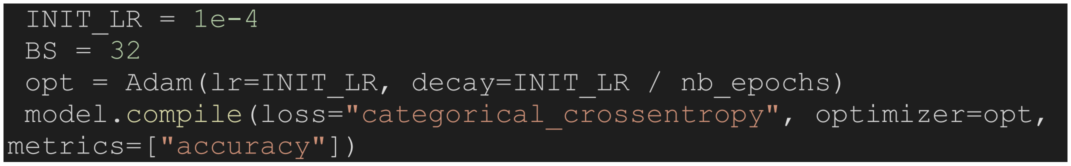
>>>Average Pooling 2D (7x7) module: downsample the output of MobileNet V2 with 7*7 pooling size. Input shape **(32, 7, 7,1280),** Output shape**: (32, 1, 1,1280)**

>>>Flatten module: Input shape **(32, 1, 1,1280),** Output shape**: (32,1280)**

>>>Dense(128) module: Input shape **(32,1280),** Output shape**: (32,128)**
>>>Dropout(0.5): Input shape **(32,128),** Output shape**: (32,128)**
>>>Dense(2): Input shape **(32,128),** Output shape**: (32,2)**

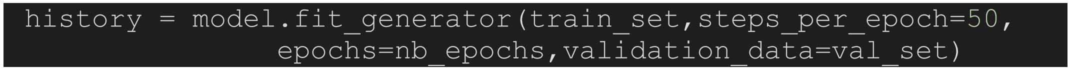
Config parameters:

- Config Adam optimizer: initial learning rate (INIT_LR = 0.0001, decay = INIT_LR/ number_of_epochs)



```
 INIT_LR = 1e-4
 BS = 32
 opt = Adam(lr=INIT_LR, decay=INIT_LR / nb_epochs)
 model.compile(loss="categorical_crossentropy", optimizer=opt,
metrics=["accuracy"])
```

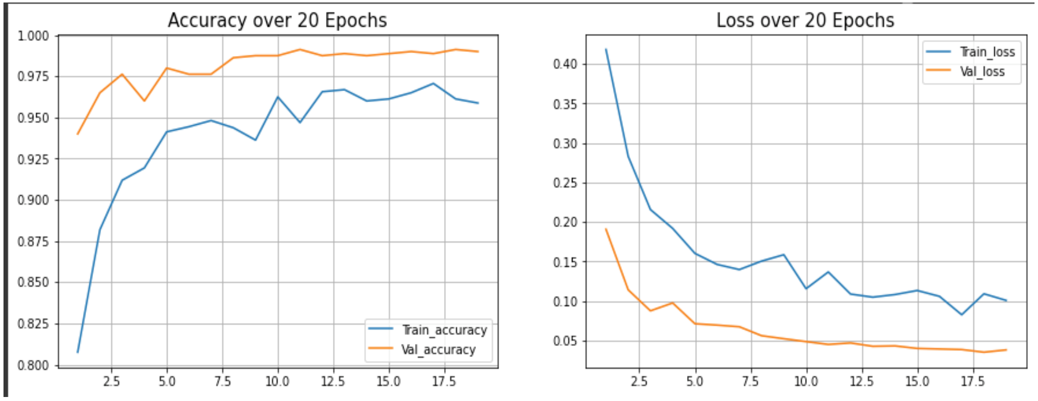
- Config **steps_per_epoch** in the fit model An epoch is completed when the dataset is once passed completely through the model. The number of steps that are requires for the completion of an epoch is ceil(dataset size/batch size). At each step, the network takes in the number of batch size samples, and the weights update constantly based on mean loss. So, at each step weights updates on its own. The steps per epoch simply indicate how many times the batch of the dataset has been fed to the network in each epoch



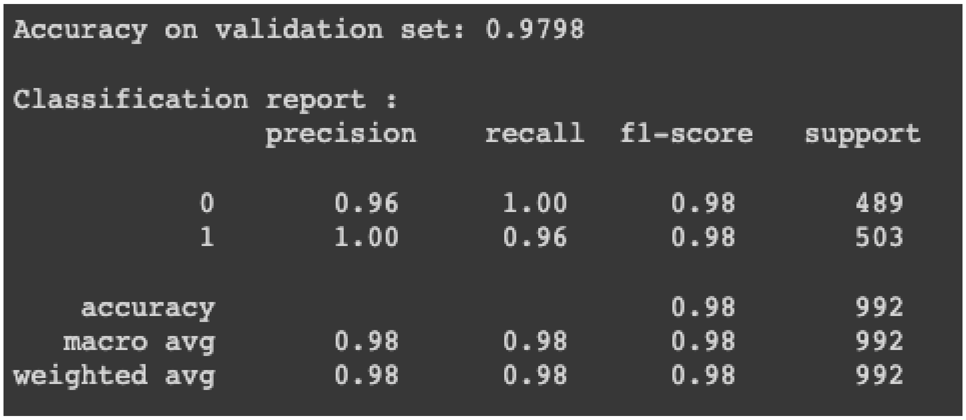
```
 history = model.fit_generator(train_set,steps_per_epoch=50,
                     epochs=nb_epochs,validation_data=val_set)
```

  - Result **steps_per_epoch = 50**

>>>Accuracy and Loss over 20 epochs



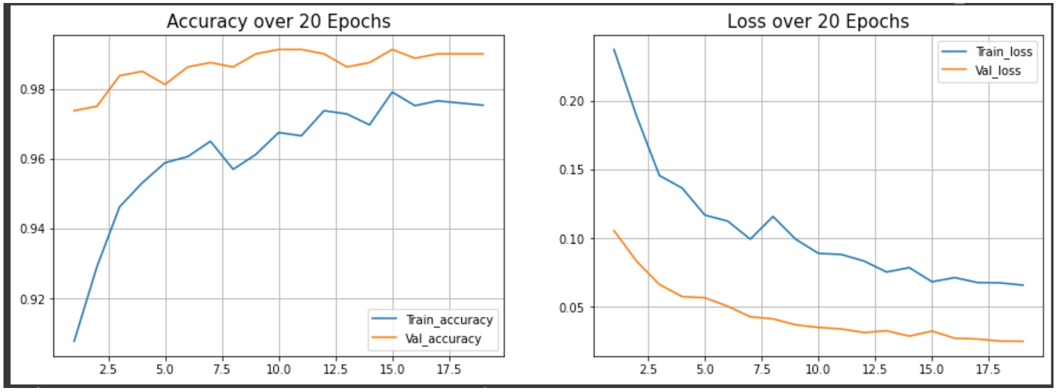
>>>Test reports



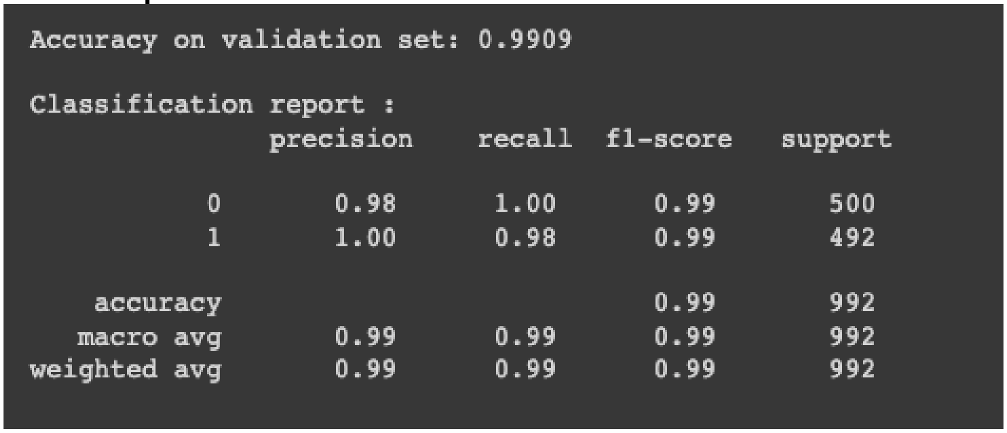
```
Accuracy on validation set: 0.9798

Classification report :
              precision    recall  f1-score   support

           0       0.96      1.00      0.98       489
           1       1.00      0.96      0.98       503

    accuracy                           0.98       992
   macro avg       0.98      0.98      0.98       992
weighted avg       0.98      0.98      0.98       992
```

- Result **steps_per_epoch= 100**

>>>Accuracy and loss over 20 epochs



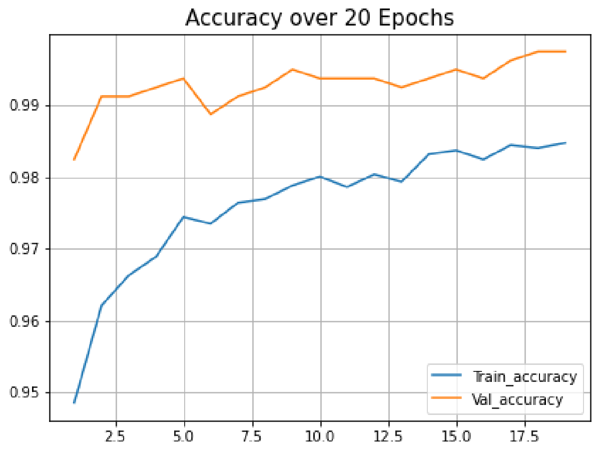
>>>test report

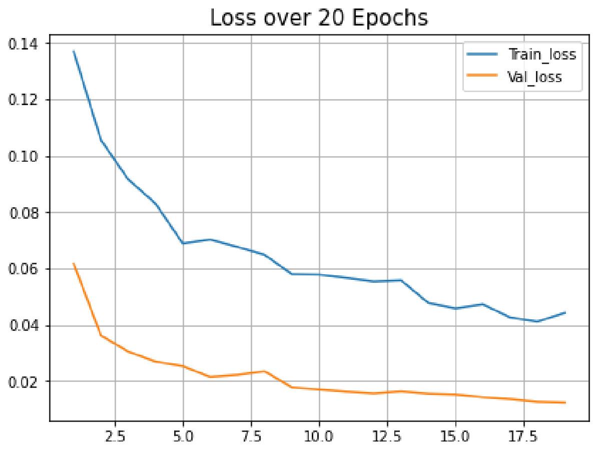


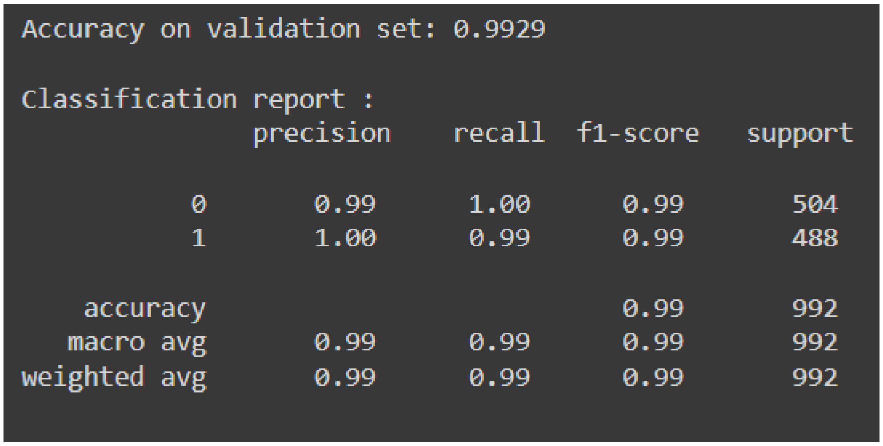
```
Accuracy on validation set: 0.9909

Classification report :
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       500
           1       1.00      0.98      0.99       492

    accuracy                           0.99       992
   macro avg       0.99      0.99      0.99       992
weighted avg       0.99      0.99      0.99       992
```

- Result **steps_per_epoch=300**

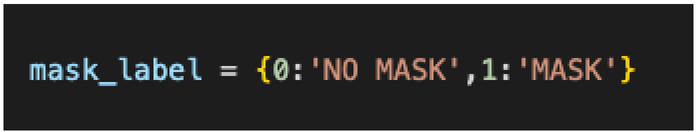
>>>Accuracy and loss over 20 epochs



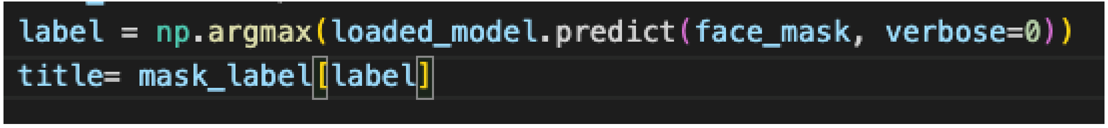




```
Accuracy on validation set: 0.9929

Classification report :
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       504
           1       1.00      0.99      0.99       488

    accuracy                           0.99       992
   macro avg       0.99      0.99      0.99       992
weighted avg       0.99      0.99      0.99       992
```



```
mask_label = {0:'NO MASK',1:'MASK'}
```



```
label = np.argmax(loaded_model.predict(face_mask, verbose=0))
title= mask_label[label]
```

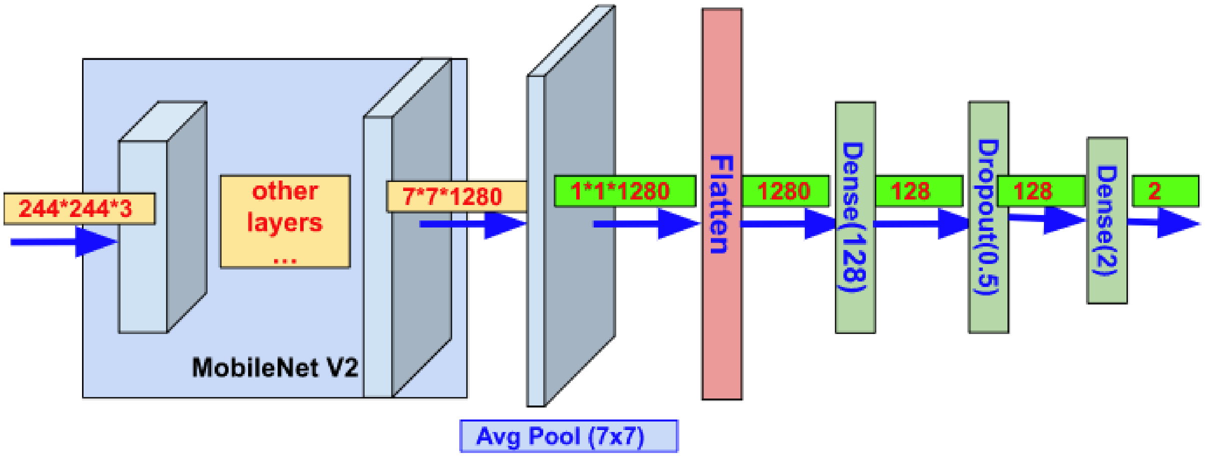
In case we get 0.5 and 0.5 the model will choose No mask as it is the first index

4. **Remaining problem**: side profile facemask detection problem
>>> Can only detect facemask with angle <60 degrees. The main reason is the face detection module (Dlib CNN / Open CV DNN) cannot detect faces which pose with angle > 60 degrees.
>>>Dlib CNN(Slow):

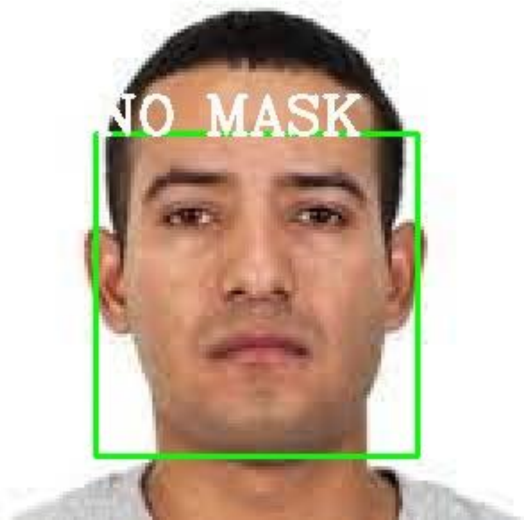
- Face with angles: **can be detected using Dlib CNN.** However, **Dlib**(both HoG+LinearSVM and CNN) cannot detect a face which is covered by a mask ⇒ our model does not get input (faces)

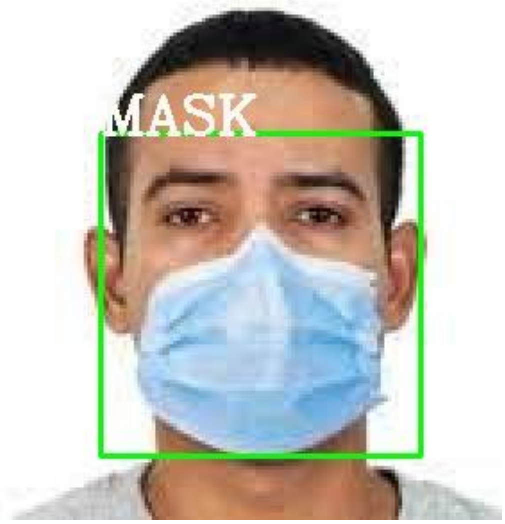
>>>Open CV DNN(fast):

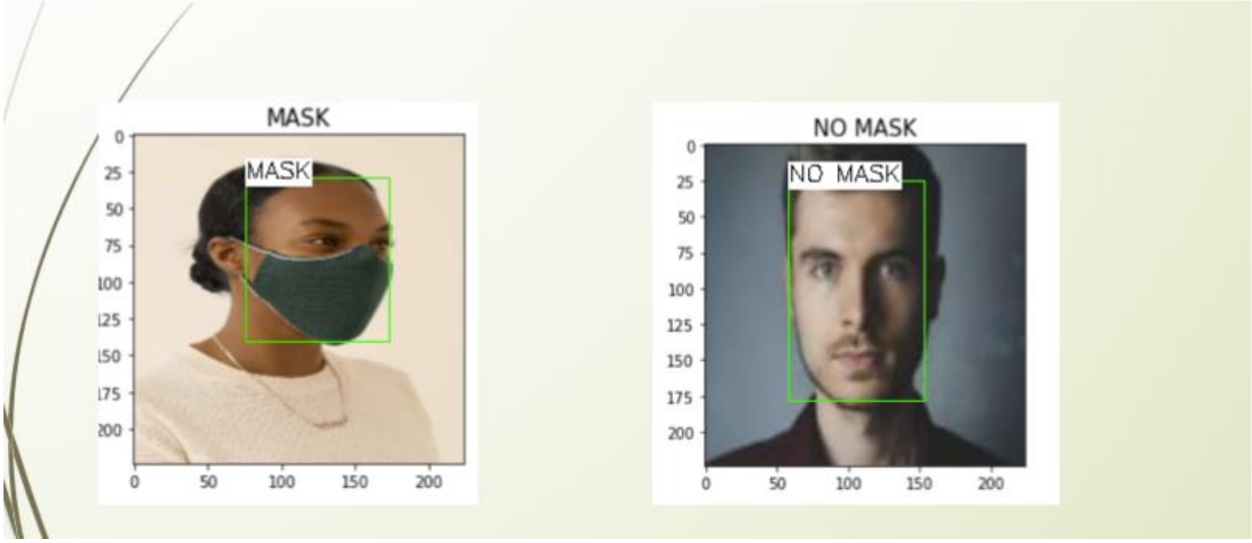
- Recognize frontal faces with/without mask
- Facemask with angle: still cannot recognize face
- Face with angle (no mask): sometime detected as ‘Mask’



Output






MASK
MASK
0
25
50
75
100
125
150
175
200
0
50
100
150
200
NO MASK
NO MASK
0
25
50
75
100
125
150
175
200
0
50
100
150
200

NO MASK