A · P · U
ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION

INDIVIDUAL ASSIGNMENT 1

TECHNOLOGY PARK MALAYSIA

CT100-3-M-DL-1

DEEP LEARNING

TOPIC: FACE MASK DETECTION ANALYSIS USING MOBILENETV2 AND VGG16 MODELS

HAND IN DATA: 17 APRIL 2021

WEIGHTAGE: 60%

Table of Content

1.0 Abstract

Ever since the novel Coronavirus (COVID-19) pandemic, face masks have been adopted globally as an essential daily infection control tool to prevent cross-contamination. Unfortunately, not many people are adhering to the regulations, hastening the transmission of the virus from one person to another. Thus, we propose a method for preventing this transmission of the virus by detecting and monitoring those who are not following the guidelines and reporting them to the appropriate authorities. This motivates us to investigate face mask detection technologies to monitor the community in public places. For this project, we have successfully implemented deep learning (DL) to assure effective real-time facemask identification to address the challenge of face mask recognition in photos. We have considered the use of MobileNetV2 and VGG16 that leverage transfer learning from ImageNet. Furthermore, we have also integrated the use of OpenCV to perform continuous facial recognition from our webcam.

2.0 Background

Ever since the outbreak of COVID-19 globally, it has become mandatory for everyone to cover their faces when out in public to prevent the virus from spreading. However, this has directly posed a problem for the current facial recognition system to recognize and classify people's faces. Even with the use of the finest facial recognition algorithm, the National Institute of Standards and Technology NIST had observed an increase in the error rate for up to 50% of matching between masked and unmasked faces for the same person (Boulos, 2021). This is because when people use face masks to adhere to the standard operating procedure (SOP), it generates a failure for the facial recognition algorithm. At the same time, we are also trying to help monitor the public to make sure the community follows the SOPs, even though it is practically challenging to entirely impose face masks on the human population.

Thus, our goal for this project is to be able to detect the presence of face masks on static photos and eventually on real-time videos in hopes to overcome those two problems mentioned previously. This project's main objectives include object identification, categorization, image, object tracking, and finally analysis of the model. A two-phased CNN faces mask detector is proposed for this work. Firstly, the training of the model takes place followed by the applications of the trained model to detect faces with or without a mask on real-time videos. Moreover, to add a layer of efficiency to our mode, the MobileNetV2 and VGG16 classification models were proposed for better outputs. The reason behind this is that these models are known to pose a faster rate of processing as compared to the basic CNN architecture. Next, the image data generator (IDG) will be employed to complement our dataset, ensuring the best possible output is produced.

I believe this work will be of benefit to the authorities and the community to curb the ongoing transmission of the virus to date. This work is valuable as it is recommended to adopt in public locations like train stations, airports, shopping malls, schools, and many more to monitor the crowd in real-time.

3.0 Introduction

The spread of the Coronavirus (COVID-19) began in 2019 in Wuhan, China, and was immediately declared a global pandemic by the World Health Organization (WHO). With the rise of the COVID-19 cases everywhere, governments around the world have decided to make it mandatory for the public to wear face masks in public areas in hopes to reduce the likelihood of transmission of the virus according to research (Ullah et al., 2022). As a result, it is now critical to keep a careful eye on the general populace and to instill in them the significance of the tiniest details of basic order procedures, such as the adoption of face masks. Even still, enforcing face masks on the public is practically impossible.

In recent years, there has been an increase in the emergence of algorithms that have been demonstrated and created for the solution of complex, life-threatening issues. One such field is the image and object detection field that enables us to identify individuals with just a click of a button. With the ongoing pandemic, now is the best time to impose the mask policy on the general population. Any surveillance system deployed at any location can be linked to the face mask recognition system. Authorities or administrators can use the service to verify the person's identification. If someone enters the premises without wearing a face mask, the system sends an alarm message to the designated individual. However, the current issue that remains a puzzle to the science community is that masked face identification causes a problem in public settings as half of the face is obscured during facial recognition, causing important data to be lost. This becomes a problem, especially for businesses that have already produced and deployed the facial recognition method for a person's verification or identification. In these sorts of circumstances, the face recognition techniques have failed to cause multiple authentication applications that rely on facial recognition to suffer setbacks such as face access control and mobile payment which require facial recognition for security purposes (Mundial et al., 2020). Thus, these issues have prompted the creation of this study.

The issue with obstructed face images, particularly masks, has not been fully solved, even though it is a major element of the recognition system. As a result, one of the most important problems in developing feasible face detection and identification systems is to make good use of deep learning structures and algorithms. Therefore, we have chosen to solve this issue by implementing the convolutional networks as they can extract high-quality features and can often provide superior accuracy for image categorization in deep learning (Alzu'bi et al., 2021). Owing to its greater spatial feature extraction capability and lower processing cost, CNN is viewed to play a major role in computer vision-related recognition applications [1]. General CNN consists of connected layers, which are the input, convolutional, subsampling, fully connected and output layers (Alzu'bi et al., 2021) to regulate the degree of shift, scale, and distortion (Setiowati et al., 2017).

We have also chosen to implement the MobileNetV2, a common pre-trained architecture that has been employed successfully in past research for facial recognition tasks. Known to be a lightweight deep neural network, the MobileNetV2 primarily relies on a streamlined design (A. G. Howard et al., 2017). The MobileNetV2 architecture has been reported to have significant improvements over the MobileNetV2 architecture in terms of classification and object detection projects (Arora et al., 2021). Its design performed well with hyperparameters, and the model calculations are quicker (Xu et al., 2021).

To create new training samples from the existing dataset, we have adopted the use of the Image Data Generator. This is known as data augmentation, where it takes the original data, randomly modifies it, and then returns the altered data. It uses a set of strategies for using random fluctuations and perturbations to create new data. This has been proven to improve the model's generalizability (Arora et al., 2021). However, the question of how to create superior Convolutional neural network topologies remains unanswered. In this research, we offer a method for accurately recognizing masked faces.

The following is a breakdown of the paper's structure. Section 3 discusses the topic in-depth, exploring the aim and objectives of the project, its scope, and motivation. Section 4 reviews past work done on face mask detection, and section 5 discusses the challenges of the topic. Section 6 on the other hand explains the methodology of the project and section 7 reveals the results and analysis of the project. Lastly, we conclude the project in section 8 with further recommendations to improve the work further for future work.

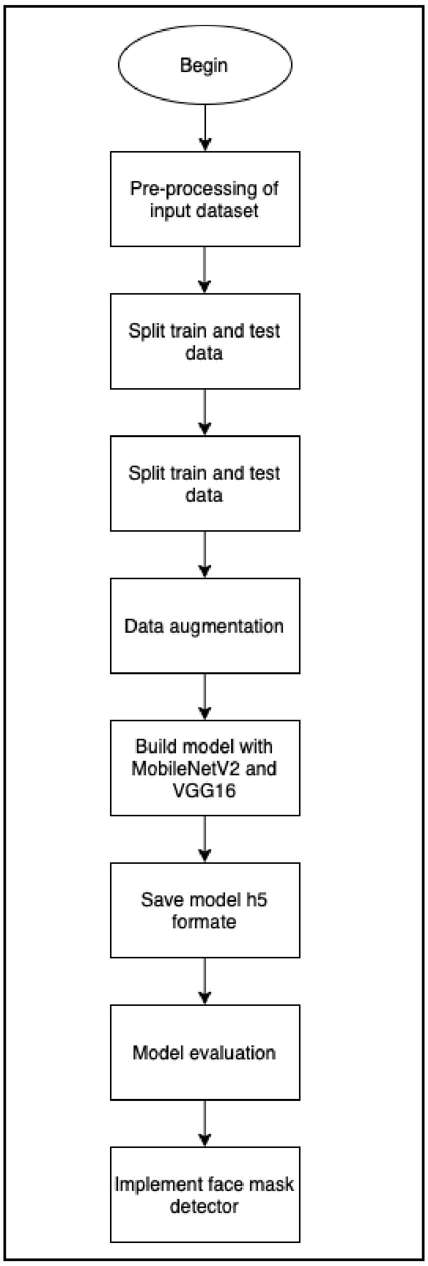

Begin
Pre-processing of input dataset
Split train and test data
Split train and test data
Data augmentation
Build model with MobileNetV2 and VGG16
Save model h5 formate
Model evaluation
Implement face mask detector

Figure 1: Workflow of the project

3.1 Aim and Objective

Aim: This project aims to address the current facial recognition challenges by creating a real-time facial recognition system that can identify whether a person is wearing a mask or not in hopes to contribute to reducing the transmission of COVID-19.

Objective:

1. To identify the barriers to the detection of face masks using the current system.
2. To introduce and provide an explanation of how deep learning using CNN and MobileNetV2 can be used to construct a new recognition system in real-time.
3. To implement required methods to pre-process the raw input images before using them for model training. This includes cleaning and deleting unnecessary data, separating the data into train and test, and applying one-hot encoding.
4. To test and make an analysis of the performances of the MobileNetV2 and VGG16 models, identify and justify which model works best to achieve our aim.

3.2 Scope

The proposed model uses a well-known deep learning method to design a classifier and collect images of people wearing and not wearing masks. We present an efficient face mask-based facial image categorization system based on a MobileNet and VGG16-based deep learning model. The packages OpenCV, Keras, and Tensorflow are used in this Python project. This model can immediately identify the mask's faces from any angle. It can generate output from any RGB image, independent of orientation. Using this technology, we hope to detect the presence of face masks on static images and real-time videos.

3.3 Motivation

With the target to gradually reduce the transmission of the virus, these models can be used to detect a mask on a person's face as masks have been proven to be able to reduce transmissions (J. Howard et al., 2021). Furthermore, with the rise of many smart cities globally, CCTV cameras can be utilized to gather real-time video footage of various public locations in the city. Facial images are taken from the camera clip, and these images are used to detect the mask on the face. With the integration of both artificial intelligence (AI) and computer vision, we have the highest possibility of implementing the mask policy within the community to screen people in hopes to prevent the transmission of the infection by determining who is adhering to the SOPs and who is not.

4.0 Literature Review

Deep learning has been an excelling technique for huge data processing has many known applications in computer vision, object, and pattern recognition, and many more. Object recognition approaches based on deep learning models have potentially become more capable than moderate models in solving difficult tasks in recent years (Yadav, 2020). Developing a real-time model capable of recognizing whether persons have worn a mask or not in public places is one example.

In (Mehedi Shamrat et al., 2021), the author demonstrated a deep learning model for face mask detection with the use of the MobileNetv2 architecture. The goal was to enable machines to perceive and comprehend in the same manner as people do, and to apply that knowledge for image and video recognition, image interpretation, and classification. To train the model, the author employed a dataset of 1845 images from various sources and 120 co-author images captured with a webcam and a mobile phone camera. CCTV footage and a Webcam were utilized to detect masks from video, and the images were in RGB format. In ensuring all the images were of the same size, the images were resized to 256 x 256 pixels. Moreover, the author also recommended the use of the Keras Image Data Generator. The author chose to split the data into 80% training and the rest for testing. Next, all the images were normalized. For the second convolution layer, the images were reduced to a size of 128 x 128 pixels and later 64 x 64 pixels for the third convolution layer. OpenCV's Caffemodel for real-time face mask detection. In the first hidden layer, the Relu activation feature was used followed by Max pooling. The image was then flattened. As for function mapping, similar architecture was used but replaced with average pooling instead. The operation was executed with a pool size of 128 hidden layers. Finally, the MobileNetV2 architecture was employed and a learning rate of 0.01 was selected to further improve accuracy. Following the training and testing phase, they evaluated the models' effectiveness using precision, recall, f1-score, and accuracy. The accuracy of the MobileNetV2 architecture was 99.72 % for training and 99.82 % for validation.

For the same work, (Loey et al., 2021) experimented on three different datasets. The Real-World Masked Face Dataset (RMFD) was the first dataset, comprising over 5000 masked and 90,000 unmasked images. The second was the Simulated Masked Face Dataset (SMFD) with over 1570 images. The Labelled Faces in the Wild (LFW) dataset was the third dataset used for their study with 13,000 images from all over the world. Two key components make up the model, which is the deep transfer learning ResNet50 as a feature extractor and the implementation of traditional machine learning such as decision trees, SVM, and ensemble. To increase model performance, the last layer of ResNet-50 was substituted with the three classic machine learning algorithms mentioned. The datasets were divided into three sections: 70% for training, 10% for validation, and 20% for testing. According to the findings, the SVM classifier attained the maximum accuracy achievable while consuming the least amount of time during the training procedure. In RMFD, the SVM classifier has a testing accuracy of 99.64%. It achieved 99.49% testing accuracy in SMFD and 100% testing accuracy in LFW.

A study by (Rahman et al., 2020), gathered 858 images of people wearing masks and 681 images of people who were not wearing masks. During the pre-processing steps, the images were converted to grayscale as RGB color images contained irrelevant information for mask detection. The grayscale images were reported to store only 8 bits per pixel and supplied enough information for classification. The images were then resized into 64 x 64 pixels to keep the images uniform. Next, they were normalized, where the pixels' value ranges from 0 to 1. 80% of each class's images were used for training, while the remaining images were used for testing. Three pairs of convolution layers were followed by one max pooling layer in the architecture. The information is then reshaped into a vector and sent into the dense network via a flattened layer. Three pairs of dense and dropout layers were used. The designed architecture was trained for 100 epochs. On the test data, the trained model had 98.7% accuracy and an AUC of 0.985.

Singh et al., (2021) investigated face mask identification techniques using two prominent object detection models, namely YOLOv3 and faster R-CNN. The approach splits the image into sections, forecasts bounding boxes and probabilities for each region, and then applies a single neural network to the entire image (Redmon et al., 2016). The authors created a small custom dataset by hand, labeled it properly, and utilized transfer learning to complete the assignment. Each image's face was labeled with bounding boxes. The authors employed the ResNet-101-FPN architecture as the foundation for training the Faster-RCNN model for 50 epochs. The YOLOv3 has 53 convolutional layers that were trained on ImageNet. Moreover, an addition of 53 layers was added on top of it, producing a YOLOv3 106-layer architecture. For the first 81 layers, the author had down sampled the images, while having 32 strides. The 1x 1 detection kernel is used to make one detection, yielding a detection feature map of 13 x 13 x 255. From layer 79, the feature map was passed through some layers before being upsampled by 2 to a dimension of 26 x 26. Next, the 94th layer then performs the second detection providing a feature map of 26 x 26 x 255. The final 106th layer has a size of 52 x 52 x 255.

(Sandesara et al., 2020) proposed a model, which is a stack of 2-D convolutional layers with Relu activations and Max Pooling, with Gradient Descent as the training method and binary cross-entropy as the loss function. The authors used a combination of two datasets, which are the RMFD and a Kaggle dataset to train their algorithm. Thus, the images were resized to 150 x 150 pixels before training. During the training phase, they employed the dropout function to disregard specific images from the dataset to improve accuracy. The pre-processed images were fed into the 32-filter Conv-2D layer. They used Max Pooling after each layer in their model to ensure the best possible accuracy. The additional Conv2D layers employed were 64,128 and 256 filters, respectively. In making sure the correct size of images entering the dense layer, the images were flattened before being passed into the output layer. A dense layer for the output with 100 neurons was implemented with an addition of a sigmoid activation function. Furthermore, the author used the Adam optimizer to train their model, with binary cross-entropy as the loss function. Overall, they achieved a 95% testing accuracy and a 97% training accuracy.

Boulos (2021) proposed a CNN model that can distinguish between persons who are wearing a face mask, those who are not wearing a face mask, and those who are wearing a face mask wrongly. The dataset used is publicly available on Kaggle with over 7553 images. People who wear face masks were labeled 0, while those who do not wear face masks were labeled 1. Images were transformed to grayscale instead of color channels and scaled to 100 x 100 pixels to train the model faster. The entire dataset was divided into a 90:10 ratio, with 90% of the data going into the train and 10% going into the test. To make sure that overfitting did not occur, the dataset was shuffled. Three convolutional layers were used in this paper to attain the best model performance with kernel size and stride as two other parameters used for each convolutional layer. The kernel size was set to 5x5 while the stride dimensions were set to 1. ReLU was implemented after every convolutional step. A 2 x 2 max-pooling was deployed to reduce the three-dimensional size of the convoluted feature and the Softmax activation function was implemented after the fully connected layers. The author also flatted the images. A learning rate of 0.001 was applied. The Adam optimizer was also used, and 30 epochs were set to attain the best accuracy. The author reported an accuracy of 97.1% using the model.

Moyo & Yuefeng, (2022) used a hybrid model combining deep and traditional machine learning using Python, OpenCV, TensorFlow, and Keras. They compared three machine learning models to select the most appropriate method with the highest level of accuracy. The author created a very simple and rudimentary Convolutional Neural Network (CNN) model and the Haar Cascade Classifier was used to detect the videocam input. The dataset used has 1315 images. RGB color images were first transformed into grayscale images during the pre-processing step. They then rescaled the input images to (150x150) pixels to keep the architecture's input visuals uniform. The images are then normalized, with pixel values ranging from 0 to 1. 80% of the images from each class were utilized for training, with the remaining images being used for assessment. In the architecture, three pairs of convolution layers were followed by one max pooling layer. The maximum pooling layer has a window size of 2x2, and the convolution layer has 100 kernels with a 3x3 window size. The output of the convolution layers was flattened and converted into a 1-D array. There were two dense layers and one dropout layer following that. The suggested architecture was trained for 10 epochs because longer training leads to overfitting of the training data. 95% accuracy was achieved by the model.

Citation	Dateset	Model	Pre-processing	Result
Mehedi Shamrat et al., 2021	Source: CCTC footage, webcam and phone camera	MobileNetV2	Images were resized to 256 x 256 pixels Split the data into 80% training, 20% testing Data Augmentation	Accuracy training: 99.72% Accuracy testing: 99.82%
	1845 images 120 co-author images			
Loey et al., 2021	Real-World Masked Face Dataset (RMFD)	ResNet50	Split data into 70% for training, 10% for validation, and 20% for testing.	
	Labelled Faces in the Wild (LFW)	Decision tree		
	Simulated Masked Face Dataset (SMFD)	SVM		Best accuracy testing: 99.64% on RMFD datatset, 99.49% on SMFD and 100% on LFW
		Ensemble		
Rahman et al., 2020	Source not mentioned	Basic CNN	The images were converted to grayscale from RGB	Accuracy testing: 98.70%
	858 images of people wearing masks		The images were then resized into 64 x 64 pixels Split the data into 80% training, 20% testing	

Singh et al., 2021	Not mentioned	YOLOv3 faster R-CNN	Not mentioned	Not mentioned
Sandesara et al., 2020	RMFD	2-D convolutional layers	Images were resized to 150 x 150 pixels	Accuracy training: 97% Accuracy testing: 95%
	Kaggle dataset			
Boulos, 2021	Kaggle	CNN	Images were transformed to grayscale instead of color channels Images scaled to 100 x 100 pixels Split the data into 90% training, 10% testing	Accuracy testing: 97.1%
Moyo & Yuefeng, 2022	Not mentioned	CNN	RGB color images were first transformed into grayscale images Rescaled the input images to (150x150) pixels Split the data into 80% training, 20% testing	Accuracy testing: 95%
		Haar Cascade Classifier		
		Feature selection		

5.0 Description of the problem

Due to the general increase in accuracy of the facial recognition system, they are now widely adopted in business settings. Unfortunately, when the world was down with the COVID-19 pandemic, governments everywhere declared the wearing of face masks in public places mandatory. However, the adaptation of face masks recently has posed a new challenge to the current commercial facial recognition system used especially for inspections as people can only be recognized by their eyes and brows now that their mouths and noses are often covered. As mentioned previously, in the research conducted by the National Institute of Standards and Technology (NIST), the algorithm accuracy when detecting masked faces has decreased significantly. These images have reduced the failure rate of even the best models to approximately 5%, whereas many alternative competent models have failed 20% to 50% of the time. This condition is known as the 'failure to enroll or template (FTE) in technical terms. Face detection models normally function by measuring a face's attributes — such as its size and distance from one another — and comparing them to data from another image. An FTE indicates that the algorithm was unable to gather enough information from a face to generate an accurate comparison.

Thus, we propose the use of the Convolutional Neural Network (CNN) as a feature extractor as it is the most significant out of the many deep learning methods. To construct distinct layers, the model uses Convolution Neural Network Layers (CNN) as its backbone architecture. Additionally, libraries like OpenCV, Keras, and Tensorflow are employed. MobileNet and the VGG16 transfer learning models were also implemented for this work.

6.0 Methodology

6.1 Description of Dataset

The collection of data is the first step in developing the Face Mask Recognition model. The dataset chosen was acquired from Github (link: https://github.com/balajisrinivas/Face-Mask-Detection/tree/master/dataset). Inside the dataset folder, there are two folders called the ‘mask’ and ‘without mask’. The ‘mask’ folder contains 1915 images while the ‘without mask’folder contains 1918 images.

6.2 Deep Learning

As technology advances, machine learning has slowly begun to progress into deep learning. It is a type of artificial neural network that uses several hidden layers to interpret the data. In deep learning, the Convolutional Neural Network has been frequently used to interpret image data. Transfer learning, on the other hand, is a model that has already been trained with other data previously. In this project, we have used the MobileNetV2 and the VGG16 models to carry out the aim.

6.3 Convolutional Neural Network (CNN)

The use of CNN for face recognition has been the most widely used approach in this field ever since the invention of deep learning. Its advantages include lowering memory needs and, as a result, lowering the number of parameters that must be taught. Input data is fed into the CNN from the first layer, which usually extracts simple features such as horizontal or diagonal edges. Next, the output is then passed onto the next layer to detect more complicated features such as corners. Each layer contains a convolution kernel to extract the most important data properties. Convolution, Pooling or Sub Sampling, Non-Linearity, and Classification are the four fundamental processes of CNN. Jupyter notebook was used to program the CNN architectures, which is written in the Python 3.9 programming language. Both the Keras library and TensorFlow backend were applied.

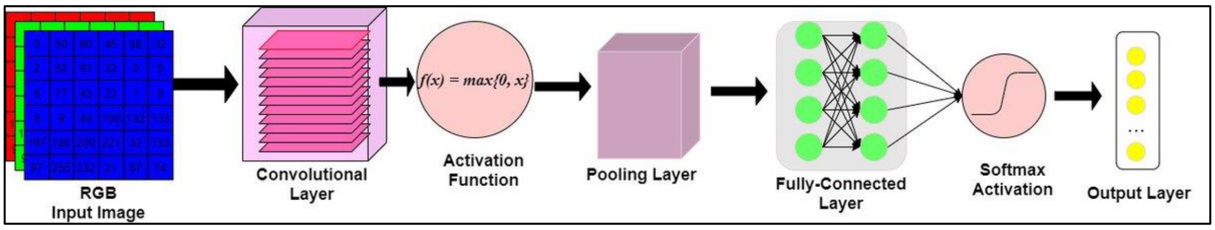


Figure 2: CNN architecture (Face Recognition Using Transfer Learning with VGG16 | by Shikhar Srivastava | Medium, n.d.)

6.3.1 Convolution

Convolution's main goal is to extract features from the input image. By learning image attributes with small squares of input data, convolution preserves the spatial connection between pixels.

6.3.2 Pooling

Pooling functions to lower the size of the input image while still retaining vital information on the images.

6.3.3 Non-linearity

To incorporate non-linearity in CNN, the activation function ReLU (Rectified Linear Unit) was used. By conducting an element-wise process, it converts all negative value pixels in a picture to zero. The activation function helps the network learn complicated patterns in the data.

6.4 MobileNetV2

MobileNetV2 is known as a classification model designed by Google, which allows for real-time classification of images, requiring small computing power. This approach applies ImageNet transfer learning to the dataset. The MobileNetV2 framework has an inverted residual structure, with narrow bottleneck layers serving as the residual blocks' input and output. In the expansion layer, lightweight convolutions are also utilized to filter features. Non-linearities in the narrow layers are subsequently eliminated. Overall, MobileNetV2's architecture includes a fully convolutional layer with 32 filters, followed by 19 residual bottleneck layers. MobileNetV2 is a convolutional neural network design that aims to be mobile-friendly.

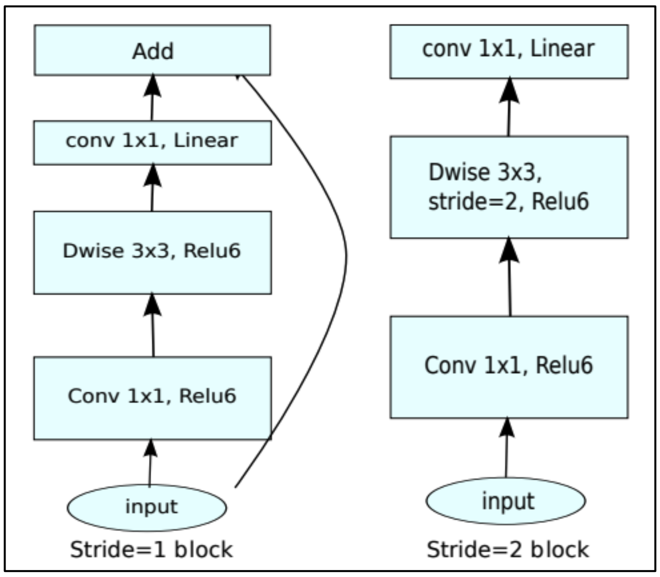


Figure 3: Architecture of the MobileNetV2 (Sandler et al., 2018)

6.5 VGG16

VGG16 is a convolutional neural network that was trained on a portion of the ImageNet dataset, which contains more than 14 million images divided into 22,000 categories (Simonyan & Zisserman, 2015). Three fundamental components make up the VGG16 model, which are the convolution, pooling, and fully connected layers. In the convolution layer, the filters are used to extract features from images. In this section, the kernel size and stride are the most essential aspects. Next, pooling is applied to lower the spatial scale of a network to lessen the number of parameters and computations needed. The fully connected layers on the other hand represent the fully connected connections of the previous layers.

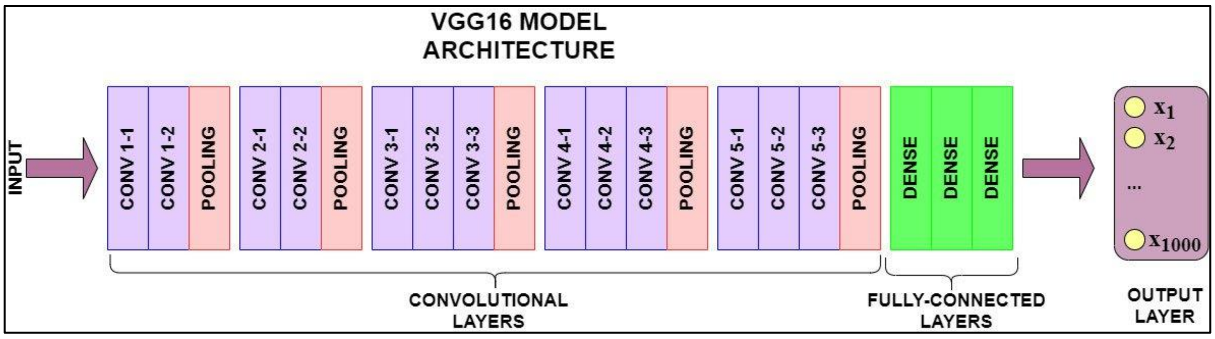


Figure 4: Architecture of the VGG16 model (*Face Recognition Using Transfer Learning with VGG16 | by Shikhar Srivastava | Medium*, n.d.)

6.6 Pre-processing

The pre-processing stage occurs before the data is trained and tested. It is carried out to clean up the raw input images before feeding them into a neural network model. The pre-processing consists of four steps: scaling the image size, turning the image to an array, pre-processing input with MobileNetV2, and performing hot encoding on labels.

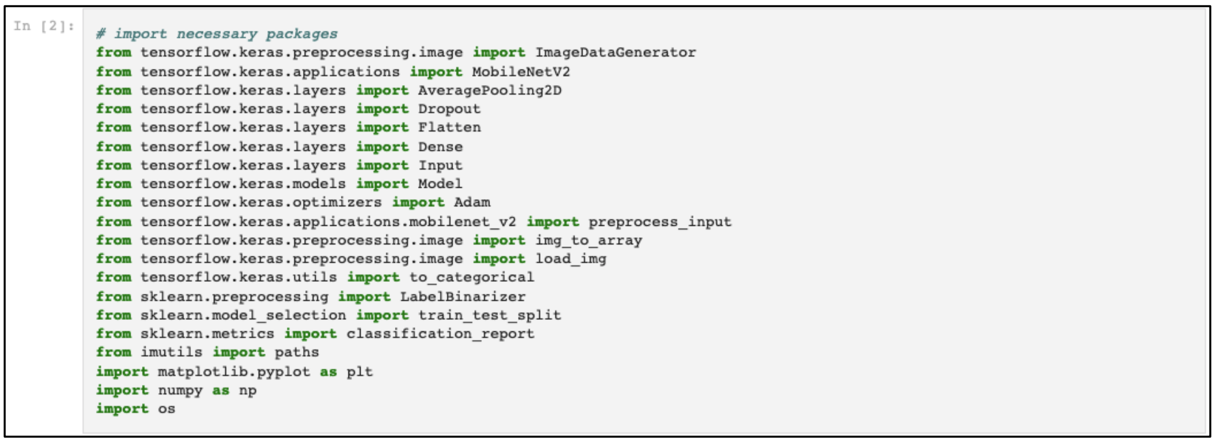
The input is rescaled to 224 × 224 pixels before being passed to the preprocess input function in tensorflow.keras.mobilenet, which is responsible for converting the image to the format required by the model. Finally, data and labels are converted to NumPy arrays using the floot32 datatype. To convert categorical data to numerical data, one hot encoding was conducted on labels.

In our project, 80%of the data was used for training, and 20% was for testing. We also employed data augmentation to increase the diversity of our dataset. Geometric transformations, flipping, width weight compensation, and rotation are some modes performed.

6.7 Model Creation

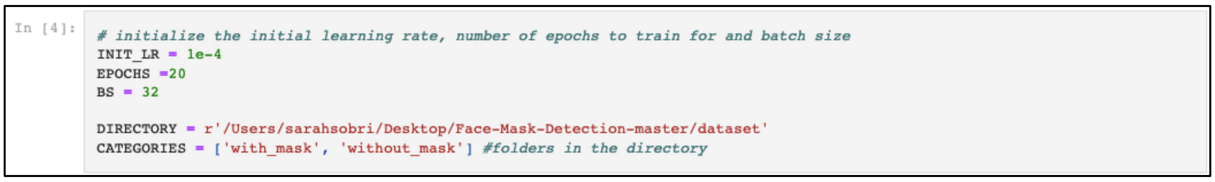
Six processes were carried out to create the model, which were constructing the image generator for augmentation, creating the base model with MobileNetV2 and VGG16, adding model parameters, compiling the model, training the model, and saving the model for future predictions. The CNN model comprises two convolutional layers, which were followed by the activation function ReLU and Average Pooling. Moreover, dropout was also included in the architecture to avoid over-fitting the model. The final step is to join the fully connected layers. In addition, in the dense layer, the SoftMax function is implemented to calculate the probability of each class. Finally, we connected the loss function and the optimizer to our model. During the learning process, the loss function is utilized to discover errors of deviations. The Adam optimizer on the other hand is a procedure that compares the prediction and the loss function to optimize the input weights, and an accuracy metric was used to evaluate the performance of the model.

6.8 Model Training



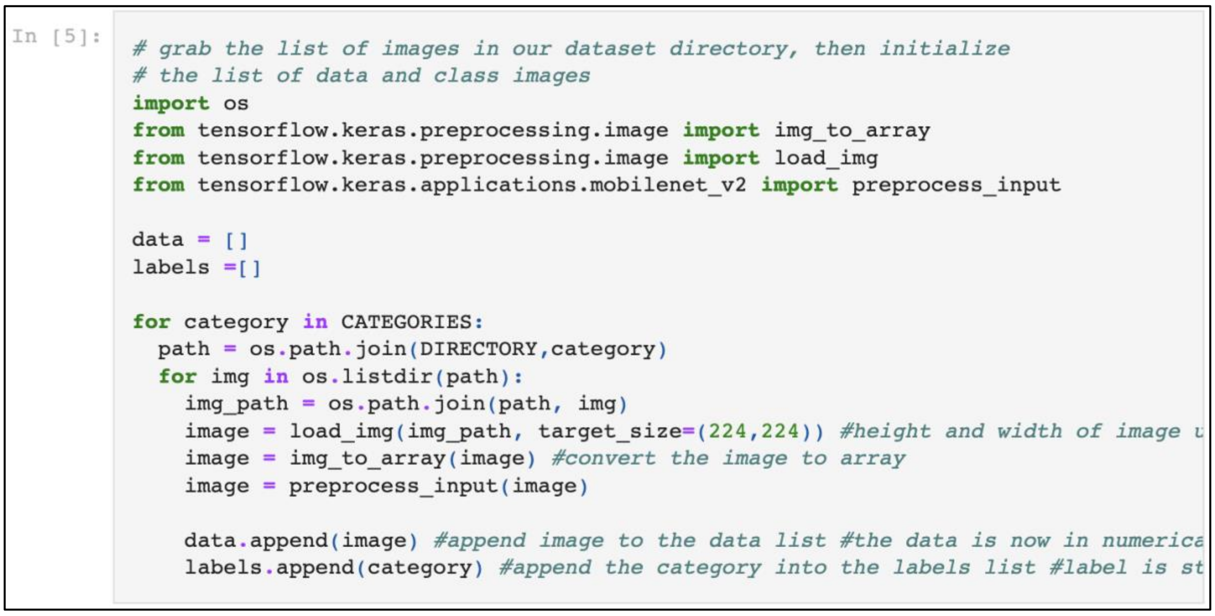
```
In [2]:
# import necessary packages
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import os
```

Comments: Before starting the project, we imported some important packages. Using the tensor.flow.keras import, we have augmented the data, load the MobileNetV2 model, construct a new fully-connected head, conduct some pre-processing procedures and finally load the image data. The 'imutils' path was used to search and list images in the data. We have also used the 'matplotlib' to plot our training and testing curves at the end of the project. The function load_img converts an image to PIL format. The function img_to_array transforms a PIL image to a Numpy array. ImageDataGenerator takes the actual data, modifies it randomly, and only outputs the new, modified data. These functions are used to enrich data. Using the to_categorical function, we were able to convert a class vector with integers into a NumPy array with binary values.



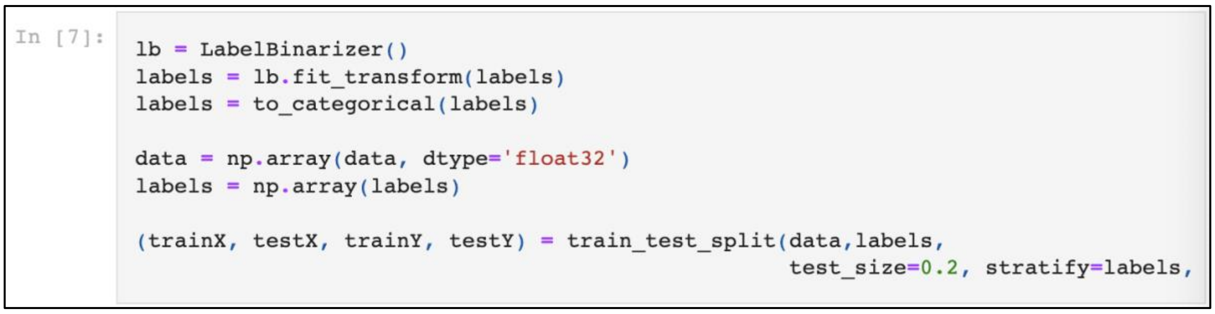
```
In [4]:
# initialize the initial learning rate, number of epochs to train for and batch size
INIT_LR = 1e-4
EPOCHS =20
BS = 32

DIRECTORY = r'/Users/sarahsobri/Desktop/Face-Mask-Detection-master/dataset'
CATEGORIES = ['with_mask', 'without_mask'] #folders in the directory
```

Comments: This block was used to centralize the deep learning hyperparameters. The initial learning rate was set to 1e-4, with 20 epochs and 32 batches.



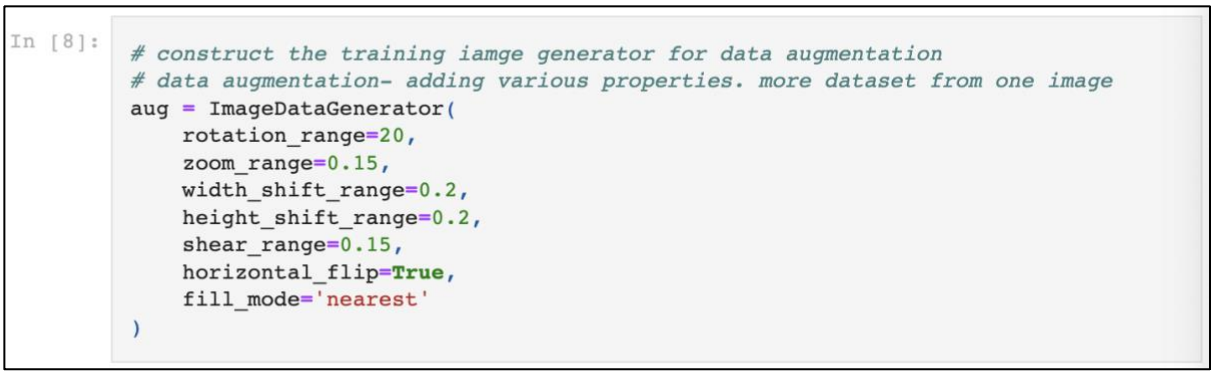
```
In [5]:  # grab the list of images in our dataset directory, then initialize
         # the list of data and class images
         import os
         from tensorflow.keras.preprocessing.image import img_to_array
         from tensorflow.keras.preprocessing.image import load_img
         from tensorflow.keras.applications.mobilenet_v2 import preprocess_input

         data = []
         labels =[]

         for category in CATEGORIES:
           path = os.path.join(DIRECTORY,category)
           for img in os.listdir(path):
             img_path = os.path.join(path, img)
             image = load_img(img_path, target_size=(224,224)) #height and width of image u
             image = img_to_array(image) #convert the image to array
             image = preprocess_input(image)

             data.append(image) #append image to the data list #the data is now in numerica
             labels.append(category) #append the category into the labels list #label is st
```

Comments: We begin by initializing the dataset with data and labels. After that, the categories are looped over and the images are loaded and pre-processed. Resizing to 224 x 224 pixels, converting to array format with the img_to_array function, and adjusting the pixel intensities in the input image to the range [-1,1] with the pre-processed input function are all phases in the pre-processing process. The pre-processed image and category are then appended to the data and labels lists, respectively.

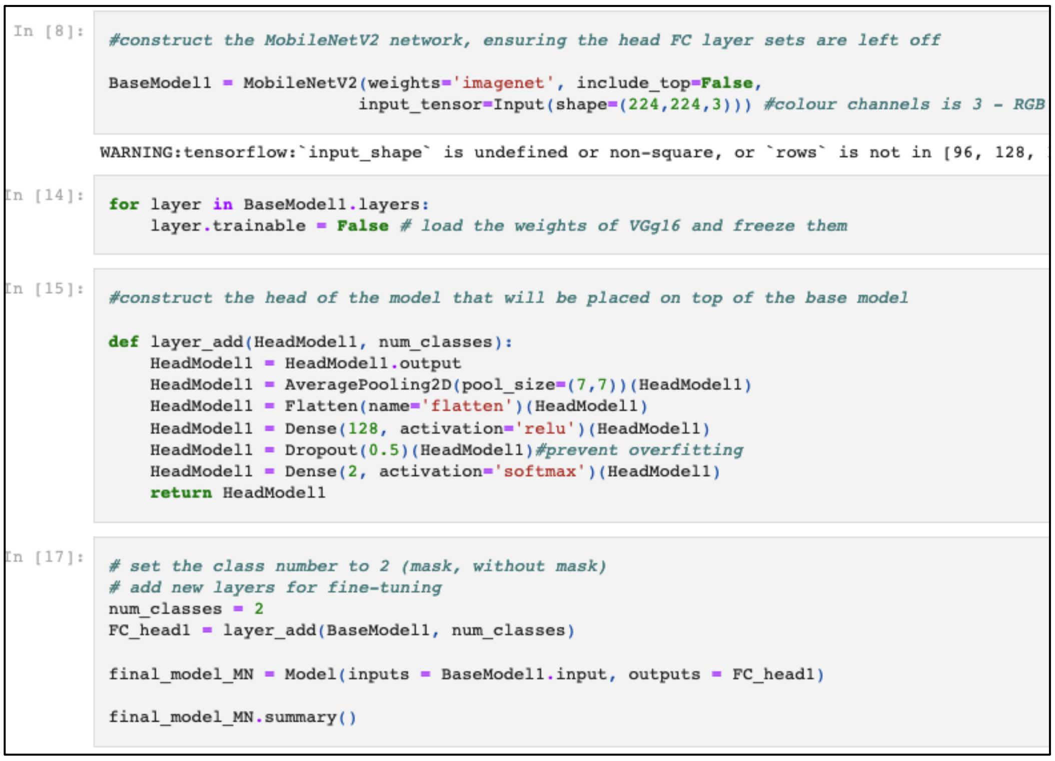


```
In [7]:  lb = LabelBinarizer()
         labels = lb.fit_transform(labels)
         labels = to_categorical(labels)

         data = np.array(data, dtype='float32')
         labels = np.array(labels)

         (trainX, testX, trainY, testY) = train_test_split(data,labels,
                                                  test_size=0.2, stratify=labels,
```

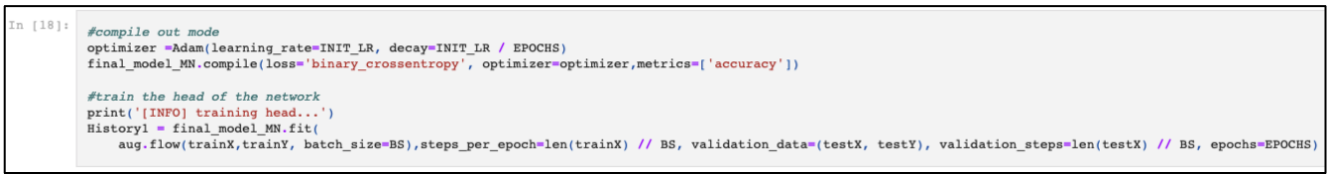
Comments: This code one-hot encode our class labels. The np.array method is then used to guarantee that our training data is in NumPy array format. The last line divides the data into training and testing data, with % of the data being train data and the remaining 20% being test data.


```
In [8]:  # construct the training iamge generator for data augmentation
         # data augmentation- adding various properties. more dataset from one image
         aug = ImageDataGenerator(
             rotation_range=20,
             zoom_range=0.15,
             width_shift_range=0.2,
             height_shift_range=0.2,
             shear_range=0.15,
             horizontal_flip=True,
             fill_mode='nearest'
         )
```

Comments: To increase generalization, we used mutation on our images during training. The randomized rotation, zoom, shear, shift, and flip parameters are defined during data augmentation. Data augmentation refers to a set of strategies for creating 'new' training samples from existing ones while maintaining the data's class labels.


```
In [8]:  #construct the MobileNetV2 network, ensuring the head FC layer sets are left off

         BaseModel1 = MobileNetV2(weights='imagenet', include_top=False,
                                  input_tensor=Input(shape=(224,224,3))) #colour channels is 3 - RGB

        WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [96, 128,

In [14]: for layer in BaseModel1.layers:
             layer.trainable = False # load the weights of VGg16 and freeze them

In [15]: #construct the head of the model that will be placed on top of the base model

         def layer_add(HeadModel1, num_classes):
             HeadModel1 = HeadModel1.output
             HeadModel1 = AveragePooling2D(pool_size=(7,7))(HeadModel1)
             HeadModel1 = Flatten(name='flatten')(HeadModel1)
             HeadModel1 = Dense(128, activation='relu')(HeadModel1)
             HeadModel1 = Dropout(0.5)(HeadModel1)#prevent overfitting
             HeadModel1 = Dense(2, activation='softmax')(HeadModel1)
             return HeadModel1

In [17]: # set the class number to 2 (mask, without mask)
         # add new layers for fine-tuning
         num_classes = 2
         FC_head1 = layer_add(BaseModel1, num_classes)

         final_model_MN = Model(inputs = BaseModel1.input, outputs = FC_head1)

         final_model_MN.summary()
```

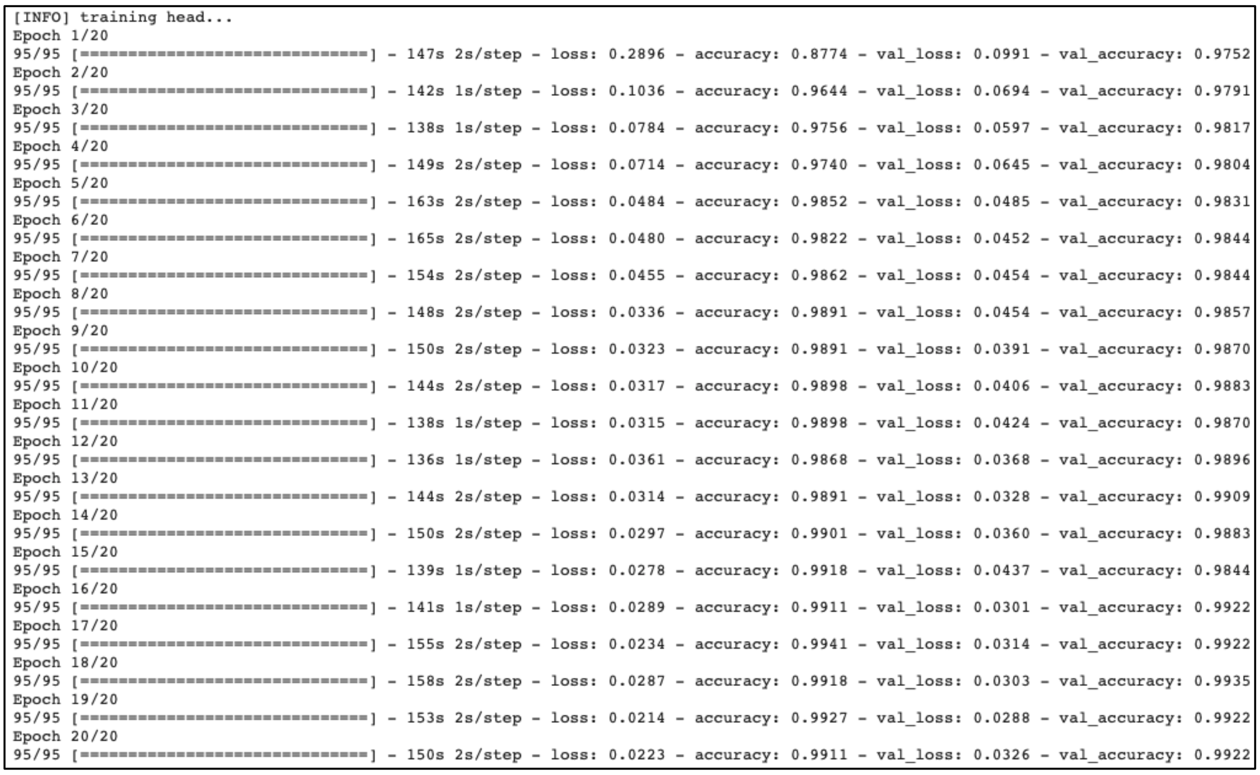
Comments: We load MobileNet with pre-trained imagenet weights, but not the network's head. The second block of codes is when we freeze the layer and not updating them with backpropagation. Next, we design a new model to be added on top of the output from the base model. We add those layers for the purpose of fine-tuning. The ‘final_mode_MN’ is out final model that will be used for training.



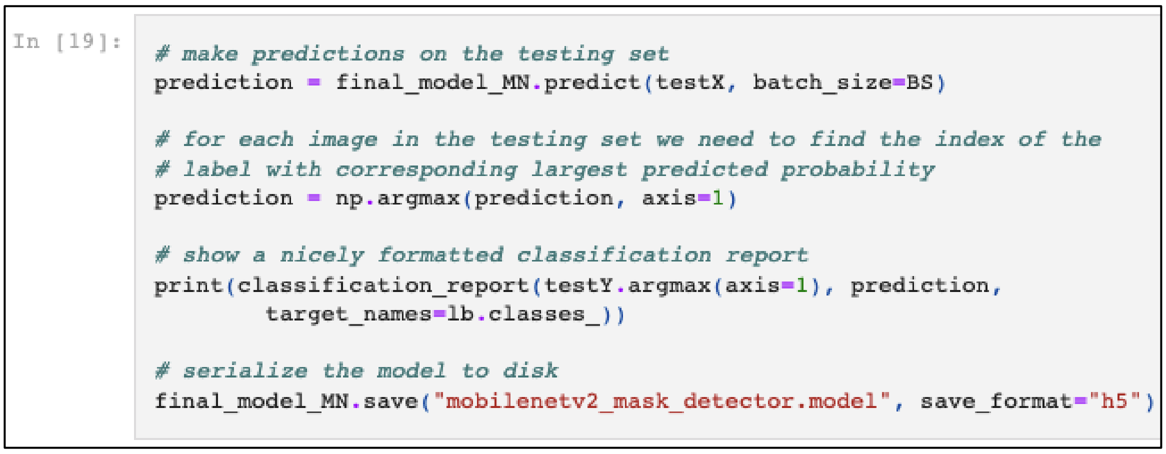
```
In [18]:  #compile out mode
          optimizer =Adam(learning_rate=INIT_LR, decay=INIT_LR / EPOCHS)
          final_model_MN.compile(loss='binary_crossentropy', optimizer=optimizer,metrics=['accuracy'])

          #train the head of the network
          print('[INFO] training head...')
          History1 = final_model_MN.fit(
              aug.flow(trainX,trainY, batch_size=BS),steps_per_epoch=len(trainX) // BS, validation_data=(testX, testY), validation_steps=len(testX) // BS, epochs=EPOCHS)
```

Comments: The first code relates to the network's foundation layers being frozen. Fine-tuning is a recommended approach for establishing a baseline model while conserving time. Next, we use the 'Adam' optimizers, a learning rate decay schedule, and binary cross-entropy to assemble our model. Because we have a two-class output, binary cross-entropy is chosen. Data augmentation provides batches of modified picture data to our model by using our 'aug' object.

Output:

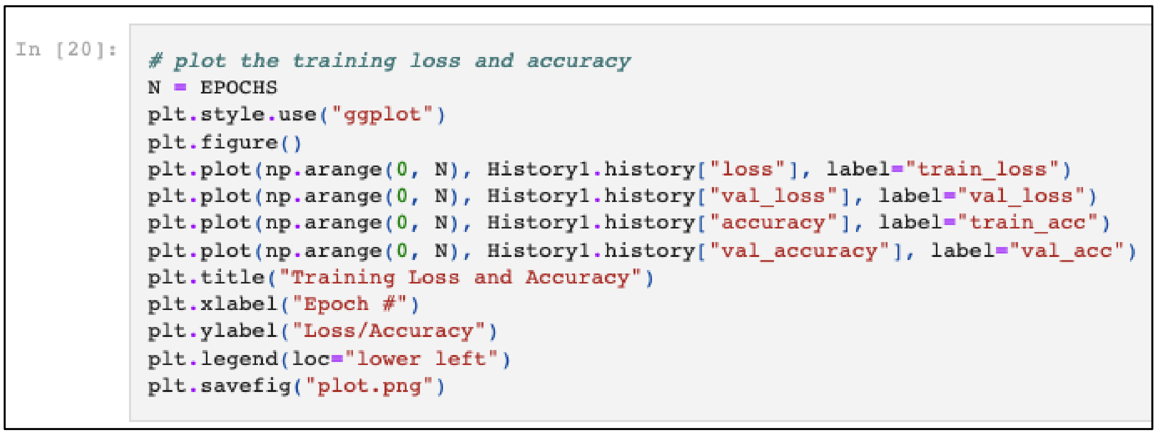


```
[INFO] training head...
Epoch 1/20
95/95 [==============================] - 147s 2s/step - loss: 0.2896 - accuracy: 0.8774 - val_loss: 0.0991 - val_accuracy: 0.9752
Epoch 2/20
95/95 [==============================] - 142s 1s/step - loss: 0.1036 - accuracy: 0.9644 - val_loss: 0.0694 - val_accuracy: 0.9791
Epoch 3/20
95/95 [==============================] - 138s 1s/step - loss: 0.0784 - accuracy: 0.9756 - val_loss: 0.0597 - val_accuracy: 0.9817
Epoch 4/20
95/95 [==============================] - 149s 2s/step - loss: 0.0714 - accuracy: 0.9740 - val_loss: 0.0645 - val_accuracy: 0.9804
Epoch 5/20
95/95 [==============================] - 163s 2s/step - loss: 0.0484 - accuracy: 0.9852 - val_loss: 0.0485 - val_accuracy: 0.9831
Epoch 6/20
95/95 [==============================] - 165s 2s/step - loss: 0.0480 - accuracy: 0.9822 - val_loss: 0.0452 - val_accuracy: 0.9844
Epoch 7/20
95/95 [==============================] - 154s 2s/step - loss: 0.0455 - accuracy: 0.9862 - val_loss: 0.0454 - val_accuracy: 0.9844
Epoch 8/20
95/95 [==============================] - 148s 2s/step - loss: 0.0336 - accuracy: 0.9891 - val_loss: 0.0454 - val_accuracy: 0.9857
Epoch 9/20
95/95 [==============================] - 150s 2s/step - loss: 0.0323 - accuracy: 0.9891 - val_loss: 0.0391 - val_accuracy: 0.9870
Epoch 10/20
95/95 [==============================] - 144s 2s/step - loss: 0.0317 - accuracy: 0.9898 - val_loss: 0.0406 - val_accuracy: 0.9883
Epoch 11/20
95/95 [==============================] - 138s 1s/step - loss: 0.0315 - accuracy: 0.9898 - val_loss: 0.0424 - val_accuracy: 0.9870
Epoch 12/20
95/95 [==============================] - 136s 1s/step - loss: 0.0361 - accuracy: 0.9868 - val_loss: 0.0368 - val_accuracy: 0.9896
Epoch 13/20
95/95 [==============================] - 144s 2s/step - loss: 0.0314 - accuracy: 0.9891 - val_loss: 0.0328 - val_accuracy: 0.9909
Epoch 14/20
95/95 [==============================] - 150s 2s/step - loss: 0.0297 - accuracy: 0.9901 - val_loss: 0.0360 - val_accuracy: 0.9883
Epoch 15/20
95/95 [==============================] - 139s 1s/step - loss: 0.0278 - accuracy: 0.9918 - val_loss: 0.0437 - val_accuracy: 0.9844
Epoch 16/20
95/95 [==============================] - 141s 1s/step - loss: 0.0289 - accuracy: 0.9911 - val_loss: 0.0301 - val_accuracy: 0.9922
Epoch 17/20
95/95 [==============================] - 155s 2s/step - loss: 0.0234 - accuracy: 0.9941 - val_loss: 0.0314 - val_accuracy: 0.9922
Epoch 18/20
95/95 [==============================] - 158s 2s/step - loss: 0.0287 - accuracy: 0.9918 - val_loss: 0.0303 - val_accuracy: 0.9935
Epoch 19/20
95/95 [==============================] - 153s 2s/step - loss: 0.0214 - accuracy: 0.9927 - val_loss: 0.0288 - val_accuracy: 0.9922
Epoch 20/20
95/95 [==============================] - 150s 2s/step - loss: 0.0223 - accuracy: 0.9911 - val_loss: 0.0326 - val_accuracy: 0.9922
```



```
In [19]: # make predictions on the testing set
         prediction = final_model_MN.predict(testX, batch_size=BS)

         # for each image in the testing set we need to find the index of the
         # label with corresponding largest predicted probability
         prediction = np.argmax(prediction, axis=1)

         # show a nicely formatted classification report
         print(classification_report(testY.argmax(axis=1), prediction,
                 target_names=lb.classes_))

         # serialize the model to disk
         final_model_MN.save("mobilenetv2_mask_detector.model", save_format="h5")
```

Comments: We use the highest probability class label indices to make predictions on the test set. Then, we printed a classification report, which will be discussed in the next section. After that, we save our face mask classification model in h5 format and named it as ‘mobilenetc2_mask_detector.model



```
In [20]: # plot the training loss and accuracy
         N = EPOCHS
         plt.style.use("ggplot")
         plt.figure()
         plt.plot(np.arange(0, N), History1.history["loss"], label="train_loss")
         plt.plot(np.arange(0, N), History1.history["val_loss"], label="val_loss")
         plt.plot(np.arange(0, N), History1.history["accuracy"], label="train_acc")
         plt.plot(np.arange(0, N), History1.history["val_accuracy"], label="val_acc")
         plt.title("Training Loss and Accuracy")
         plt.xlabel("Epoch #")
         plt.ylabel("Loss/Accuracy")
         plt.legend(loc="lower left")
         plt.savefig("plot.png")
```

Comments: This block of code allows us to plot our accuracy and loss curves for our training and test sets, which will be discussed in the next section.

6.9 Model Prediction

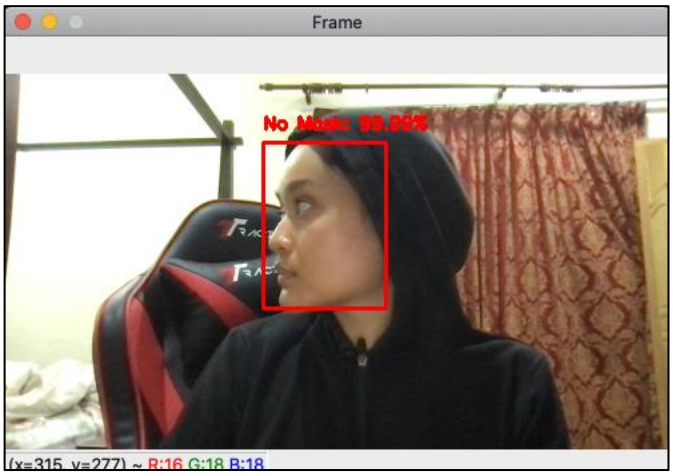
The final stage is to assess the model's performance by predicting the test data labels. Several performance criteria, including accuracy, precision, recall, and the F1-score, are utilized to evaluate the proposed CNN architecture's performance. True positive (TP), true negative (TN), false positive (FP), and false-negative (FN) are the four variables used to calculate these scores (FN). The TP variable represents a result in which the model accurately predicts the positive class. The TN variable indicates when the model accurately predicts the negative class.

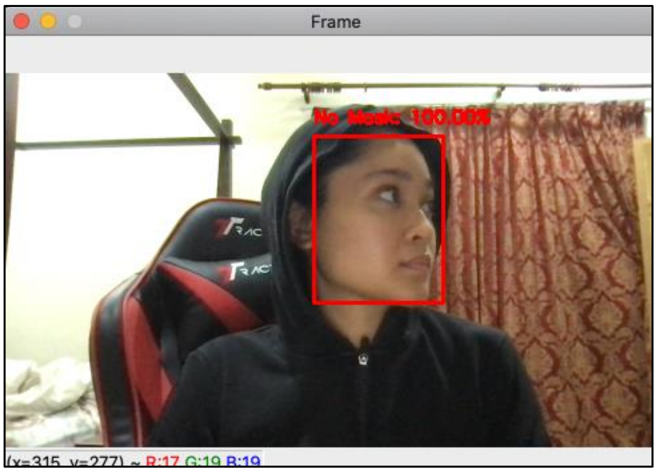
6.10 Implementing Face Mask Detector

The input image is first imported from the disk, and then a Caffe-based face detector is used to detect faces in the images. A protoxt file of the Caffe model defines the model architecture, while the weights for the actual layers are contained in the Caffe model file. By using OpenCV, the face mask detector is used in real-time video streaming.

After reading the video frame by frame, the face detection algorithm is applied. If a face is detected, the program moves on to the next step. Reprocessing will be done on detected frames containing faces, including shrinking the image size, converting to the array, and pre-processing input using MobileNetV2. The same process is being repeated with the VGG16 model.

Next, the input data from the saved model earlier was predicted. Moreover, the video frame will be labeled with whether the person is wearing a mask or not, as well as the predicted percentage shown on top of the frame. Figures 5 and 6 shows an example of how the model is put into action.





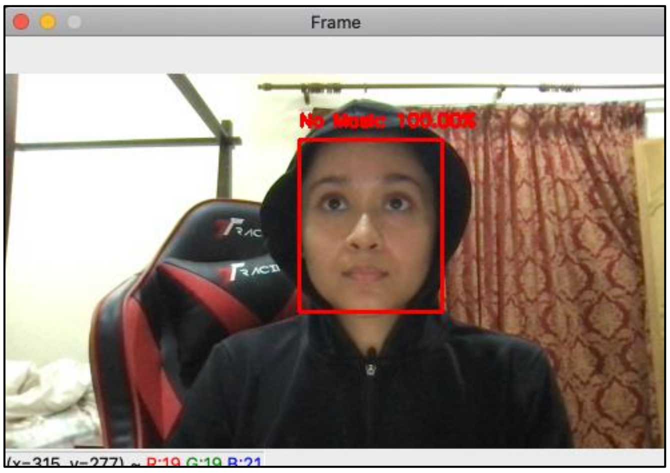
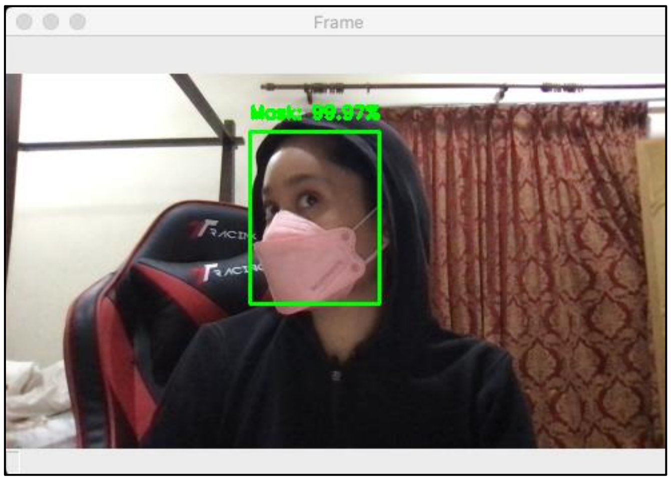
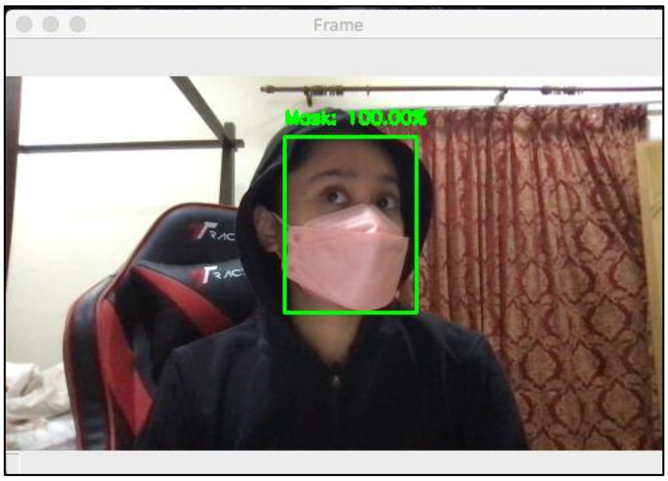


Figure 5: Screenshot of output without mask





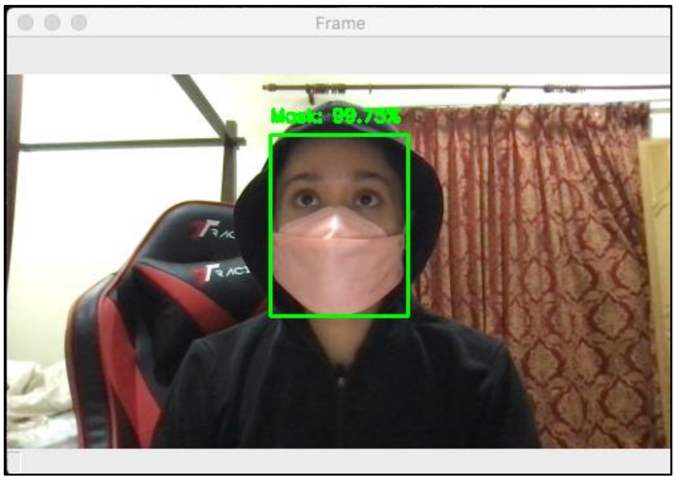
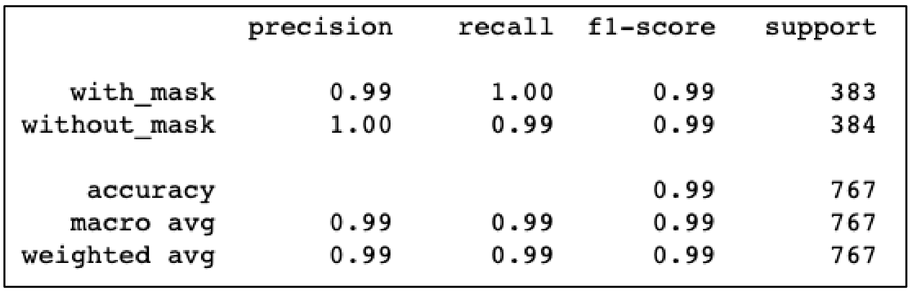


Figure 6: Screenshot of output with mask

7.0 Results and Analysis

Model: MobileNetV2



```
              precision    recall  f1-score   support

   with_mask       0.99      1.00      0.99       383
without_mask       1.00      0.99      0.99       384

    accuracy                           0.99       767
   macro avg       0.99      0.99      0.99       767
weighted avg       0.99      0.99      0.99       767
```

Figure 7: Screenshot of the classification report

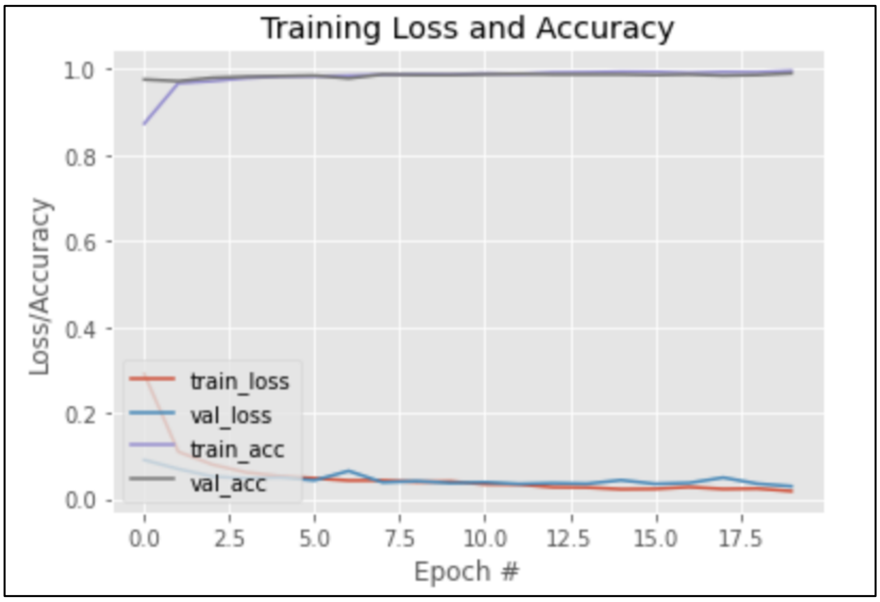


Figure 8: Screenshot of the training loss and accuracy curves

Comments: From the curves, we can clearly see that we have successfully obtained an accuracy of approximately 99.00% on both the training and validations accuracy. Moreover, we can say that the model has a good fit as the validation loss is neither too low nor too high than the training loss. We can also observe that the loss gradually becoming constant from epoch 14 onwards.

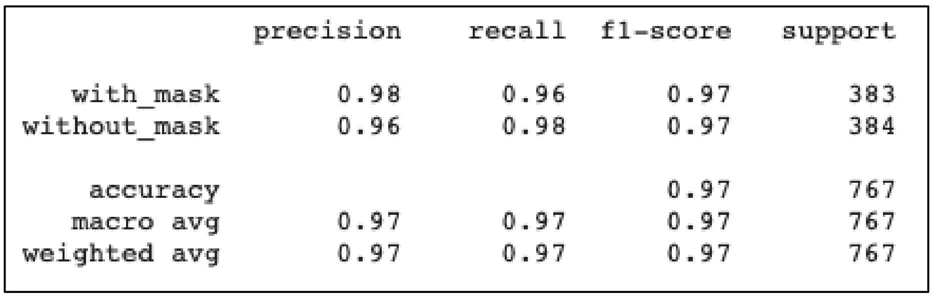
Table 1: Iteration of checking loss and accuracy

Epoch	Loss	Accuracy	Val_loss	Val_acc
1/20	0.2896	0.8774	0.0991	0.9752
2/20	0.1036	0.9644	0.0694	0. 9791
3/20	0.0784	0.9756	0.0597	0. 9817
4/20	0.0714	0.9740	0.0645	0.9804
5/20	0.0484	0.9852	0.0485	0.9831
6/20	0. 0480	0.9822	0.0452	0.9844
7/20	0.0455	0.9862	0.0454	0.9844
8/20	0.0336	0.9891	0.0454	0.9857
9/20	0.0323	0.9891	0.0391	0.9870
10/20	0.0317	0.9898	0.0406	0.9883
11/20	0.0315	0.9898	0.0424	0.9870
12/20	0.0361	0.9868	0.0368	0.9896
13/20	0.0314	0.9891	0.0328	0.9909
14/20	0.0297	0.9901	0.0360	0.9883
15/20	0.0278	0.9918	0.0437	0.9844
16/20	0.0289	0.9911	0.0301	0.9922
17/20	0.0234	0.9941	0.0314	0.9922
18/20	0.0287	0.9918	0.0303	0.9935
19/20	0.0214	0.9927	0.0288	0.9922
20/20	0.0223	0.9911	0.0326	0.9922

Table 2: Model evaluation

	Accuracy	Precision	Recall	F1-score
With mask	0.99	1.00	0.99	0.99
Without mask	1.00	0.99	0.99	0.99

Model: VGG16



```
              precision    recall  f1-score   support

   with_mask       0.98      0.96      0.97       383
without_mask       0.96      0.98      0.97       384

    accuracy                           0.97       767
   macro avg       0.97      0.97      0.97       767
weighted avg       0.97      0.97      0.97       767
```

Figure 9: Screenshot of the classification report

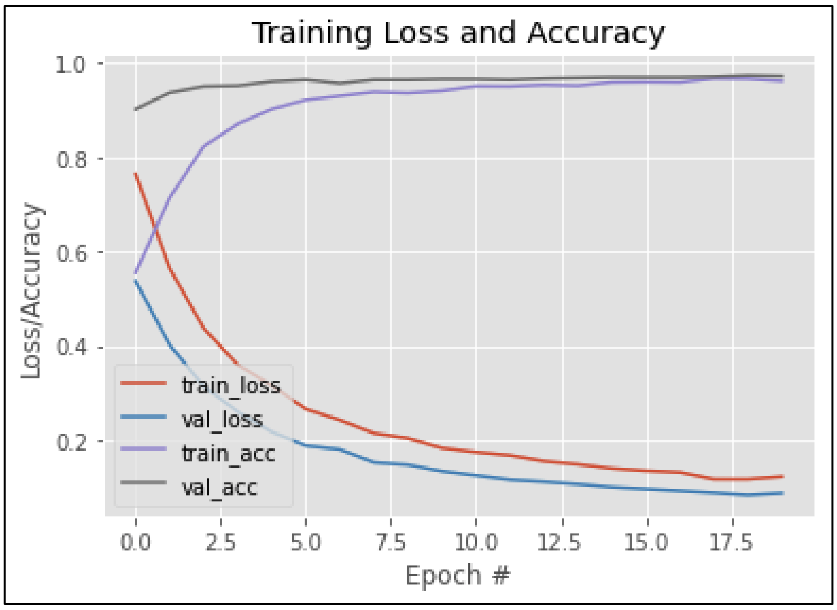


Figure 10: Screenshot of the training loss and accuracy curves

Comments: From the curves, we can clearly see that we have successfully obtained an accuracy of approximately 97.00% on both the training and validations accuracy. However, we can observe that the model is slightly overfit as the validation loss is seen lower than the training loss. We can also see that the validation loss is consistently lesser than the training loss, however the gap between the two lessens after each epoch. This could possibly be due to the dropout function used in the model as it is activated when training but deactivated when evaluating on the validation set.

Table 3: Iteration of checking loss and accuracy

Epoch	Loss	Accuracy	Val_loss	Val_acc
1/20	0.7645	0.5550	0.5368	0.9022
2/20	0.5638	0.7142	0.4014	0.9374
3/20	0.4372	0.8230	0.3136	0.9505
4/20	0.3593	0.8711	0.2596	0.9518
5/20	0.3153	0.9024	0.2168	0.9609
6/20	0.2654	0.9216	0.1875	0.9648
7/20	0.2418	0.9305	0.1795	0.9570
8/20	0.2139	0.9390	0.1520	0.9648
9/20	0.2032	0.9367	0.1469	0.9648
10/20	0.1824	0.9413	0.1331	0.9661
11/20	0.1737	0.9509	0.1240	0.9661
12/20	0.1668	0.9506	0.1150	0.9648
13/20	0.1547	0.9535	0.1107	0.9674
14/20	0.1479	0.9520	0.1056	0.9687
15/20	0.1391	0.9591	0.0992	0.9700
16/20	0.1340	0.9598	0.9593	0.9700
17/20	0.1306	0.9591	0.0913	0.9700
18/20	0.1160	0.9674	0.0872	0.9713
19/20	0.1160	0.9667	0.0825	0.9739
20/20	0.1215	0.9621	0.0866	0.9726

Table 4: Model evaluation

	Accuracy	Precision	Recall	F1-score
With mask	0.97	0.98	0.96	0.97
Without mask	0.97	0.96	0.98	0.97

8.0 Conclusion and Recommendation

In this project, we have successfully designed a CNN model with the integration of transfer learning models such as the MobileNetV2 and VGG16 to achieve our aim, which is to develop a model than can detect whether a person is wearing a mask or not in real time. Out of the two models, the MobileNetV2 outperforms the VGG16 with a training and testing accuracy of 99% each. Moreover, we can also deduce that the MobileNetV2 model took a shorter time to train compared to the VGG16 model. Thus, we recommend the use of the MobileNetV2 model to be integrated into a system that can be used to monitor the people in public locations to help reduce the transmission of the COVID-19 virus.

9.0 References

Alzu'bi, A., Albalas, F., Al-Hadhrami, T., Younis, L. B., & Bashayreh, A. (2021). Masked face recognition using deep learning: A review. *Electronics (Switzerland)*, *10*(21), 1–35. https://doi.org/10.3390/electronics10212666

Arora, M., Garg, S., & A., S. (2021). Face Mask Detection System using Mobilenetv2. *International Journal of Engineering and Advanced Technology*, *10*(4), 127–129. https://doi.org/10.35940/ijeat.d2404.0410421

Boulos, M. M. (2021). *Montclair State University Digital Commons Facial Recognition and Face Mask Detection Using Machine Learning Techniques*.

Face Recognition Using Transfer Learning with VGG16 | by Shikhar Srivastava | Medium. (n.d.). https://medium.com/@shikharsrivastava_14544/face-recognition-using-transfer-learning-with-vgg16-3caeca4a916e

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. http://arxiv.org/abs/1704.04861

Howard, J., Huang, A., Li, Z., Tufekci, Z., Zdimal, V., van der Westhuizen, H. M., von Delft, A., Price, A., Fridman, L., Tang, L. H., Tang, V., Watson, G. L., Bax, C. E., Shaikh, R., Questier, F., Hernandez, D., Chu, L. F., Ramirez, C. M., & Rimoin, A. W. (2021). An evidence review of face masks against COVID-19. *Proceedings of the National Academy of Sciences of the United States of America*, *118*(4), 1–12. https://doi.org/10.1073/pnas.2014564118

Loey, M., Manogaran, G., Taha, M. H. N., & Khalifa, N. E. M. (2021). A hybrid deep transfer learning model with machine learning methods for face mask detection in the era of the COVID-19 pandemic. *Measurement: Journal of the International Measurement Confederation*, *167*(July 2020), 108288. https://doi.org/10.1016/j.measurement.2020.108288

Mehedi Shamrat, F. M. J., Chakraborty, S., Billah, M. M., Jubair, M. Al, Islam, M. S., & Ranjan, R. (2021). Face Mask Detection using Convolutional Neural Network (CNN) to reduce the spread of Covid-19. *Proceedings of the 5th International Conference on Trends in Electronics and Informatics, ICOEI 2021*, *May*, 1231–1237. https://doi.org/10.1109/ICOEI51242.2021.9452836

Moyo, M., & Yuefeng, C. (2022). COVID-19 Face Mask Detection Alert System. *Computer Engineering and Intelligent Systems*, *March*, 0–15. https://doi.org/10.7176/ceis/13-2-01

Mundial, I. Q., Ul Hassan, M. S., Tiwana, M. I., Qureshi, W. S., & Alanazi, E. (2020). Towards facial recognition problem in COVID-19 pandemic. *2020 4th International Conference on Electrical, Telecommunication and Computer Engineering, ELTICOM 2020 - Proceedings*, 210–214. https://doi.org/10.1109/ELTICOM50775.2020.9230504

Rahman, M. M., Manik, M. M. H., Islam, M. M., Mahmud, S., & Kim, J. H. (2020). An automated system to limit COVID-19 using facial mask detection in smart city network. *IEMTRONICS 2020 - International IOT, Electronics and Mechatronics Conference, Proceedings*, *October*. https://doi.org/10.1109/IEMTRONICS51293.2020.9216386

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-December*, 779–788. https://doi.org/10.1109/CVPR.2016.91

Sandesara, A. G., Joshi, D. D., & Joshi, S. D. (2020). Facial Mask Detection Using Stacked CNN Model. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, *3307*, 264–270. https://doi.org/10.32628/cseit206553

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 4510–4520. https://doi.org/10.1109/CVPR.2018.00474

Setiowati, S., Zulfanahri, Franita, E. L., & Ardiyanto, I. (2017). A review of optimization method in face recognition: Comparison deep learning and non-deep learning methods. *2017 9th International Conference on Information Technology and Electrical Engineering, ICITEE 2017*, *2018-January*, 1–6. https://doi.org/10.1109/ICITEED.2017.8250484

Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 1–14.

Singh, S., Ahuja, U., Kumar, M., Kumar, K., & Sachdeva, M. (2021). Face mask detection using YOLOv3 and faster R-CNN models: COVID-19 environment. *Multimedia Tools and Applications*, *80*(13), 19753–19768. https://doi.org/10.1007/s11042-021-10711-8

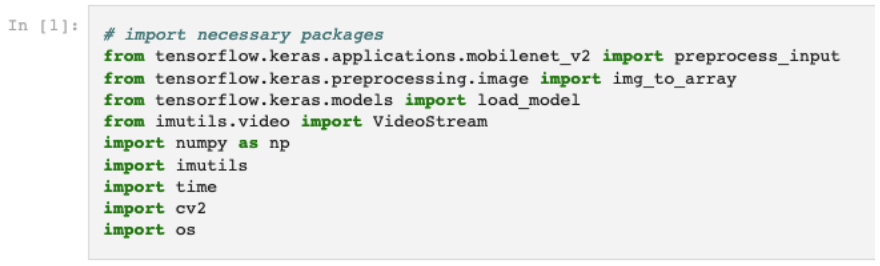
Ullah, N., Javed, A., Ali Ghazanfar, M., Alsufyani, A., & Bourouis, S. (2022). A novel DeepMaskNet model for face mask detection and masked facial recognition. *Journal of King Saud University - Computer and Information Sciences*, *xxxx*. https://doi.org/10.1016/j.jksuci.2021.12.017

Xu, X., Du, M., Guo, H., Chang, J., & Zhao, X. (2021). Lightweight FaceNet Based on MobileNet. *International Journal of Intelligence Science*, *11*(01), 1–16. https://doi.org/10.4236/ijis.2021.111001

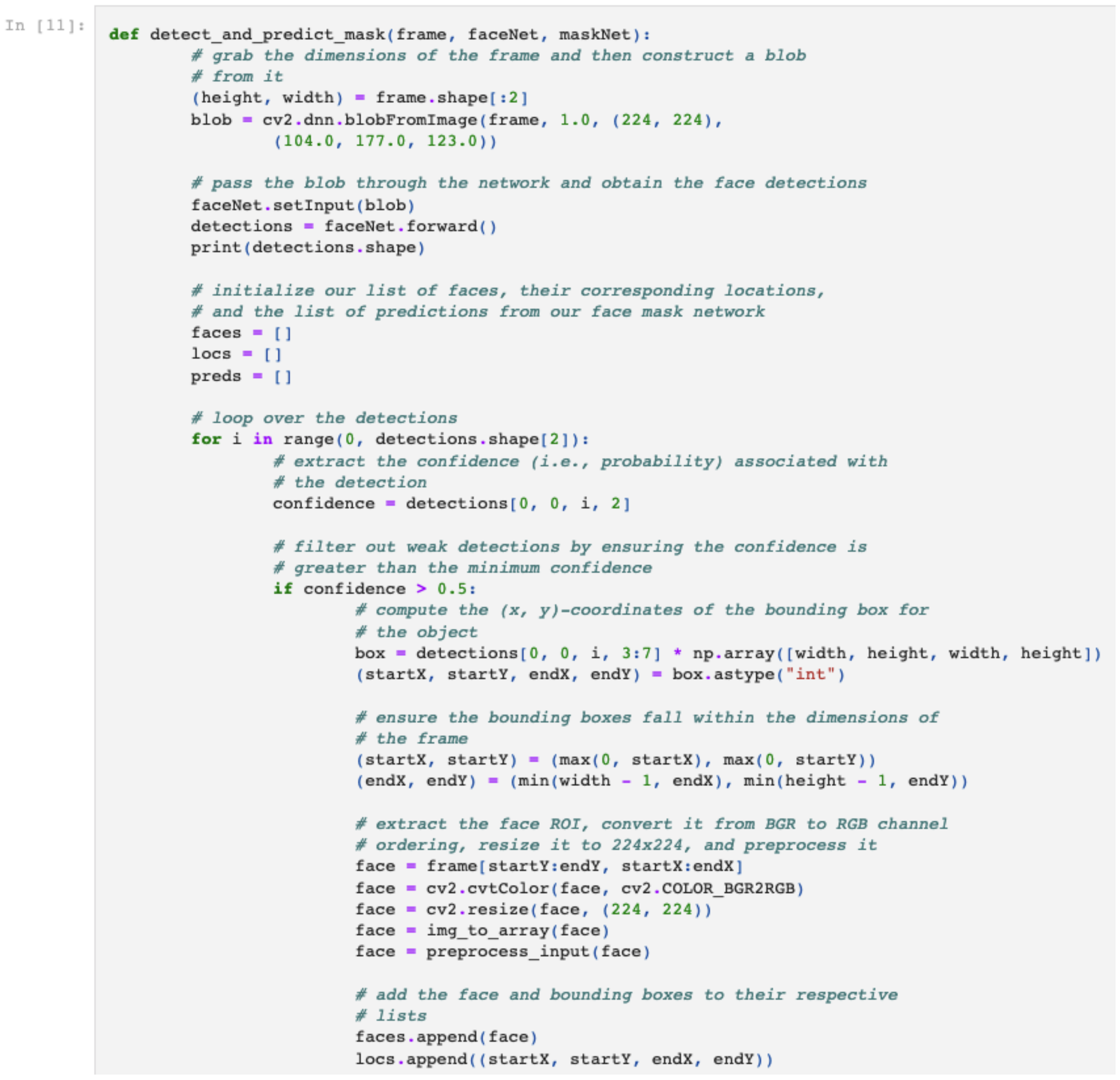
Yadav, S. (2020). Deep Learning based Safe Social Distancing and Face Mask Detection in Public Areas for COVID-19 Safety Guidelines Adherence. *International Journal for Research in Applied Science and Engineering Technology*, *8*(7), 1368–1375. https://doi.org/10.22214/ijraset.2020.30560

10.0 Appendix

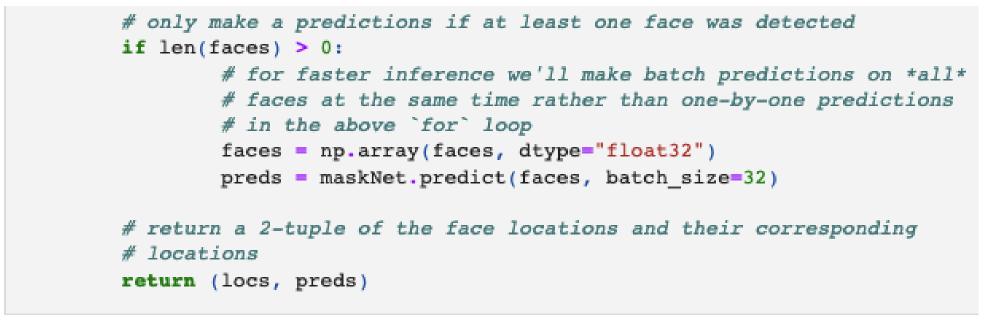
Codes to detect face mask with the MobileNetV2 model using OpenCV



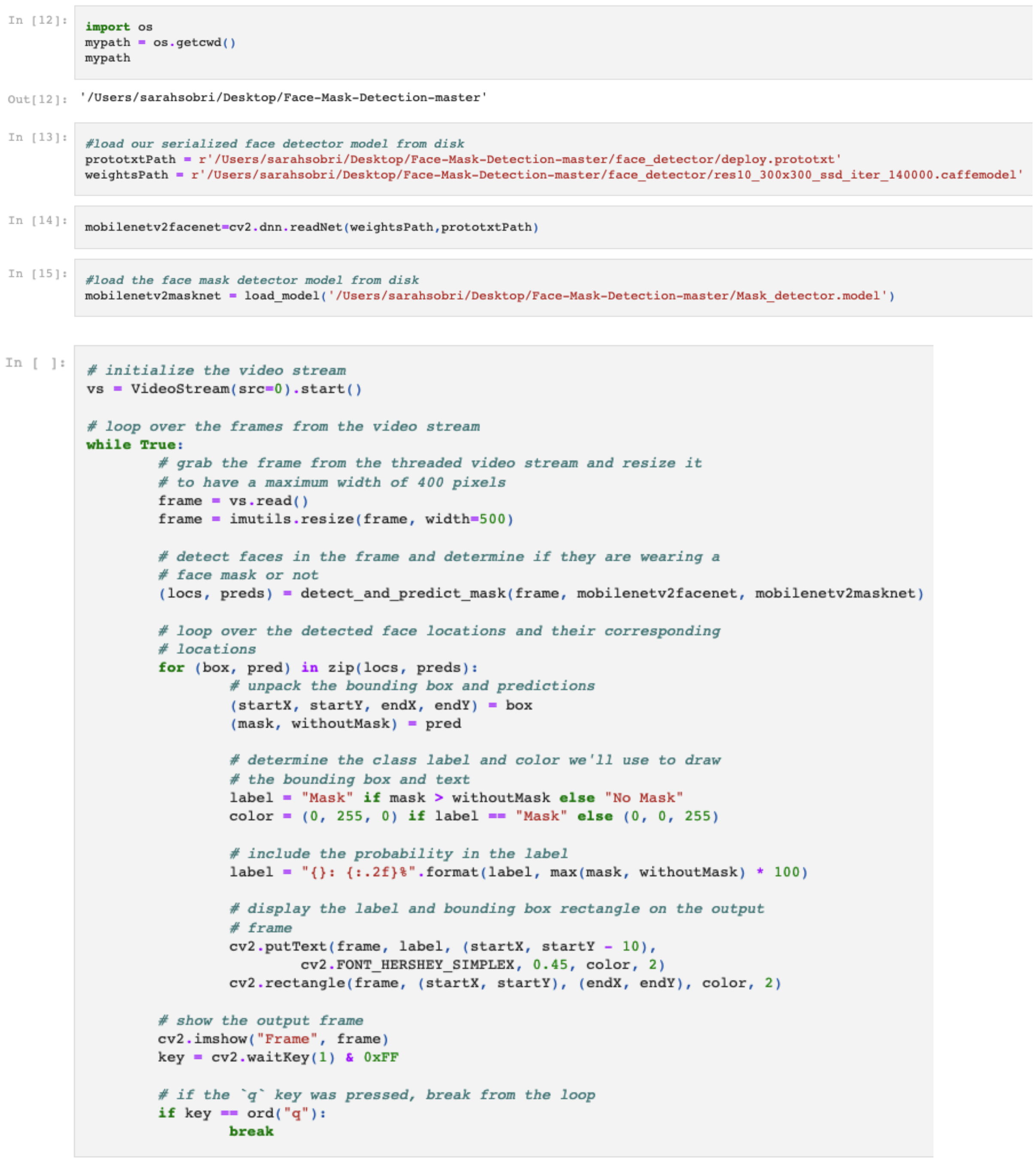
```
In [1]: # import necessary packages
        from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
        from tensorflow.keras.preprocessing.image import img_to_array
        from tensorflow.keras.models import load_model
        from imutils.video import VideoStream
        import numpy as np
        import imutils
        import time
        import cv2
        import os
```



```
In [11]: def detect_and_predict_mask(frame, faceNet, maskNet):
                 # grab the dimensions of the frame and then construct a blob
                 # from it
                 (height, width) = frame.shape[:2]
                 blob = cv2.dnn.blobFromImage(frame, 1.0, (224, 224),
                         (104.0, 177.0, 123.0))

                 # pass the blob through the network and obtain the face detections
                 faceNet.setInput(blob)
                 detections = faceNet.forward()
                 print(detections.shape)

                 # initialize our list of faces, their corresponding locations,
                 # and the list of predictions from our face mask network
                 faces = []
                 locs = []
                 preds = []

                 # loop over the detections
                 for i in range(0, detections.shape[2]):
                         # extract the confidence (i.e., probability) associated with
                         # the detection
                         confidence = detections[0, 0, i, 2]

                         # filter out weak detections by ensuring the confidence is
                         # greater than the minimum confidence
                         if confidence > 0.5:
                                 # compute the (x, y)-coordinates of the bounding box for
                                 # the object
                                 box = detections[0, 0, i, 3:7] * np.array([width, height, width, height])
                                 (startX, startY, endX, endY) = box.astype("int")

                                 # ensure the bounding boxes fall within the dimensions of
                                 # the frame
                                 (startX, startY) = (max(0, startX), max(0, startY))
                                 (endX, endY) = (min(width - 1, endX), min(height - 1, endY))

                                 # extract the face ROI, convert it from BGR to RGB channel
                                 # ordering, resize it to 224x224, and preprocess it
                                 face = frame[startY:endY, startX:endX]
                                 face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
                                 face = cv2.resize(face, (224, 224))
                                 face = img_to_array(face)
                                 face = preprocess_input(face)

                                 # add the face and bounding boxes to their respective
                                 # lists
                                 faces.append(face)
                                 locs.append((startX, startY, endX, endY))
```

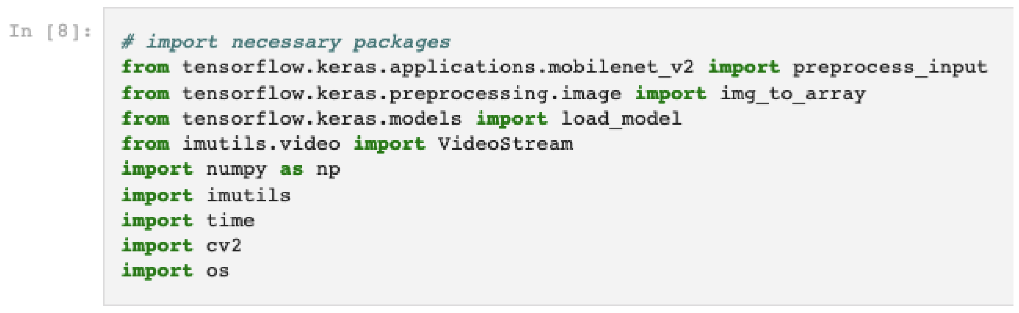


```
    # only make a predictions if at least one face was detected
    if len(faces) > 0:
            # for faster inference we'll make batch predictions on *all*
            # faces at the same time rather than one-by-one predictions
            # in the above `for` loop
            faces = np.array(faces, dtype="float32")
            preds = maskNet.predict(faces, batch_size=32)

    # return a 2-tuple of the face locations and their corresponding
    # locations
    return (locs, preds)
```

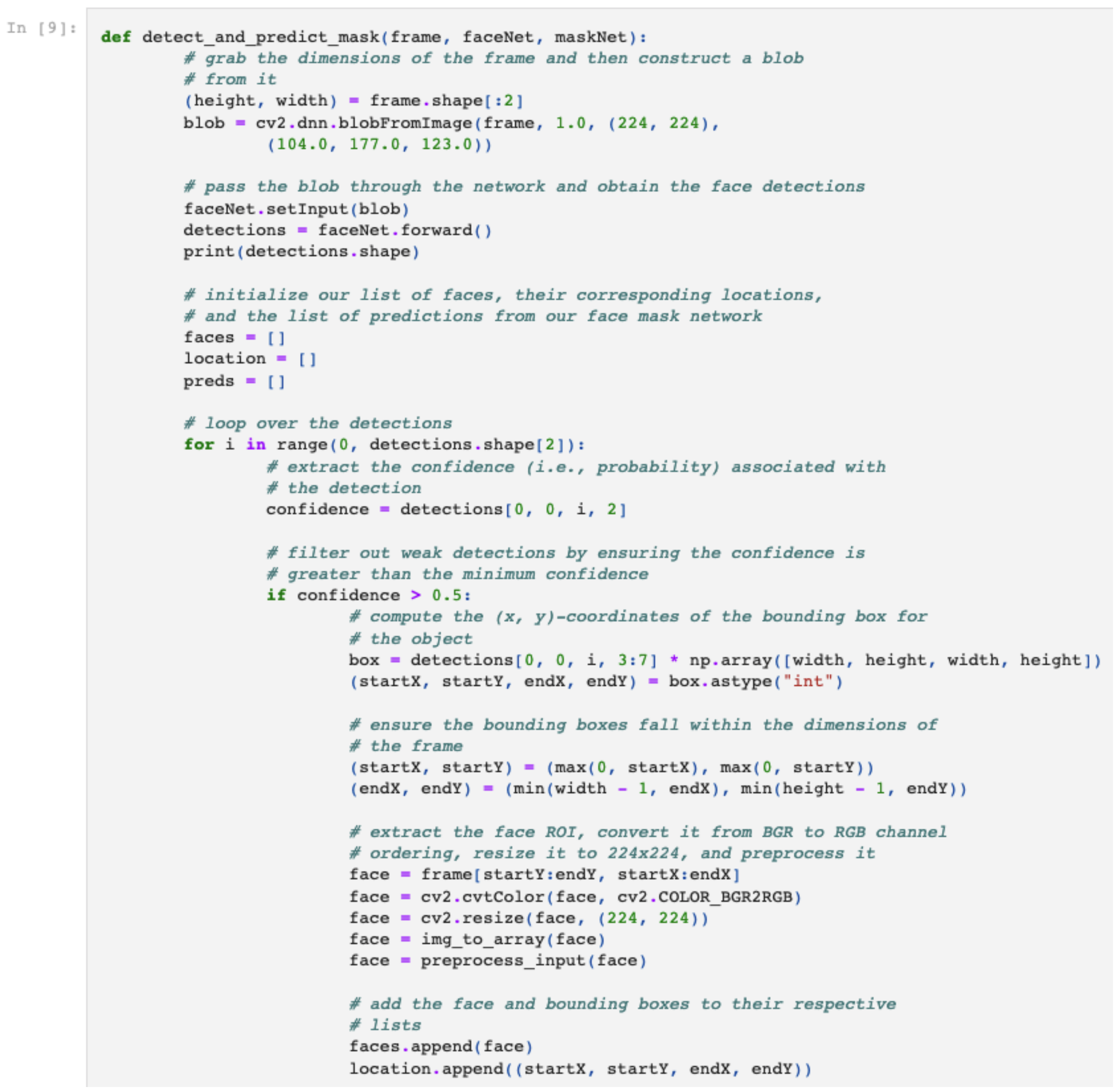


```
In [12]: import os
         mypath = os.getcwd()
         mypath

Out[12]: '/Users/sarahsobri/Desktop/Face-Mask-Detection-master'

In [13]: #load our serialized face detector model from disk
         prototxtPath = r'/Users/sarahsobri/Desktop/Face-Mask-Detection-master/face_detector/deploy.prototxt'
         weightsPath = r'/Users/sarahsobri/Desktop/Face-Mask-Detection-master/face_detector/res10_300x300_ssd_iter_140000.caffemodel'

In [14]: mobilenetv2facenet=cv2.dnn.readNet(weightsPath,prototxtPath)

In [15]: #load the face mask detector model from disk
         mobilenetv2masknet = load_model('/Users/sarahsobri/Desktop/Face-Mask-Detection-master/Mask_detector.model')

In [ ]: # initialize the video stream
        vs = VideoStream(src=0).start()

        # loop over the frames from the video stream
        while True:
                # grab the frame from the threaded video stream and resize it
                # to have a maximum width of 400 pixels
                frame = vs.read()
                frame = imutils.resize(frame, width=500)

                # detect faces in the frame and determine if they are wearing a
                # face mask or not
                (locs, preds) = detect_and_predict_mask(frame, mobilenetv2facenet, mobilenetv2masknet)

                # loop over the detected face locations and their corresponding
                # locations
                for (box, pred) in zip(locs, preds):
                        # unpack the bounding box and predictions
                        (startX, startY, endX, endY) = box
                        (mask, withoutMask) = pred

                        # determine the class label and color we'll use to draw
                        # the bounding box and text
                        label = "Mask" if mask > withoutMask else "No Mask"
                        color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

                        # include the probability in the label
                        label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)

                        # display the label and bounding box rectangle on the output
                        # frame
                        cv2.putText(frame, label, (startX, startY - 10),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
                        cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)

                # show the output frame
                cv2.imshow("Frame", frame)
                key = cv2.waitKey(1) & 0xFF

                # if the `q` key was pressed, break from the loop
                if key == ord("q"):
                        break
```

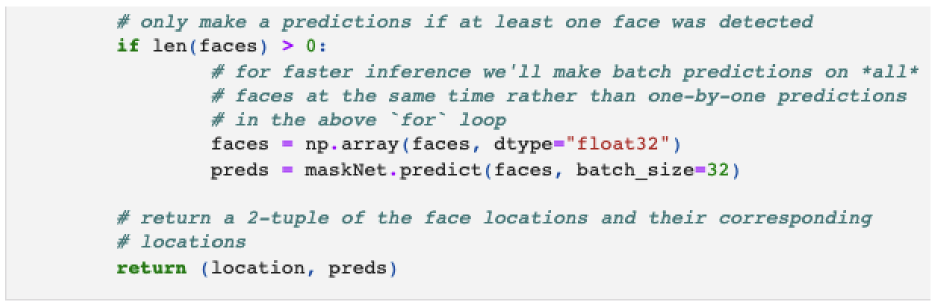
Codes to detect face mask with the VGG16 model using OpenCV



```
In [8]:
# import necessary packages
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
from imutils.video import VideoStream
import numpy as np
import imutils
import time
import cv2
import os
```



```
In [9]:
def detect_and_predict_mask(frame, faceNet, maskNet):
        # grab the dimensions of the frame and then construct a blob
        # from it
        (height, width) = frame.shape[:2]
        blob = cv2.dnn.blobFromImage(frame, 1.0, (224, 224),
                (104.0, 177.0, 123.0))

        # pass the blob through the network and obtain the face detections
        faceNet.setInput(blob)
        detections = faceNet.forward()
        print(detections.shape)

        # initialize our list of faces, their corresponding locations,
        # and the list of predictions from our face mask network
        faces = []
        location = []
        preds = []

        # loop over the detections
        for i in range(0, detections.shape[2]):
                # extract the confidence (i.e., probability) associated with
                # the detection
                confidence = detections[0, 0, i, 2]

                # filter out weak detections by ensuring the confidence is
                # greater than the minimum confidence
                if confidence > 0.5:
                        # compute the (x, y)-coordinates of the bounding box for
                        # the object
                        box = detections[0, 0, i, 3:7] * np.array([width, height, width, height])
                        (startX, startY, endX, endY) = box.astype("int")

                        # ensure the bounding boxes fall within the dimensions of
                        # the frame
                        (startX, startY) = (max(0, startX), max(0, startY))
                        (endX, endY) = (min(width - 1, endX), min(height - 1, endY))

                        # extract the face ROI, convert it from BGR to RGB channel
                        # ordering, resize it to 224x224, and preprocess it
                        face = frame[startY:endY, startX:endX]
                        face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
                        face = cv2.resize(face, (224, 224))
                        face = img_to_array(face)
                        face = preprocess_input(face)

                        # add the face and bounding boxes to their respective
                        # lists
                        faces.append(face)
                        location.append((startX, startY, endX, endY))
```



```
        # only make a predictions if at least one face was detected
        if len(faces) > 0:
                # for faster inference we'll make batch predictions on *all*
                # faces at the same time rather than one-by-one predictions
                # in the above `for` loop
                faces = np.array(faces, dtype="float32")
                preds = maskNet.predict(faces, batch_size=32)

        # return a 2-tuple of the face locations and their corresponding
        # locations
        return (location, preds)
```

In [10]:
```python
#load our serialized face detector model from disk
prototxtPath = r'/Users/sarahsobri/Desktop/Face-Mask-Detection-master/face_detector/deploy.prototxt'
weightsPath = r'/Users/sarahsobri/Desktop/Face-Mask-Detection-master/face_detector/res10_300x300_ssd_iter_140000.caffemodel'
```

In [11]:
```python
vgg16facenet=cv2.dnn.readNet(weightsPath,prototxtPath)
```

In [14]:
```python
#load the face mask detector model from disk
vgg16masknet = load_model('/Users/sarahsobri/Desktop/Face-Mask-Detection-master/vgg_mask_detector.model')
```

In [ ]:
```python
# initialize the video stream
vs = VideoStream(src=0).start()

# loop over the frames from the video stream
while True:
        # grab the frame from the threaded video stream and resize it
        # to have a maximum width of 400 pixels
        frame = vs.read()
        frame = imutils.resize(frame, width=500)

        # detect faces in the frame and determine if they are wearing a
        # face mask or not
        (locs, preds) = detect_and_predict_mask(frame, vgg16facenet, vgg16masknet)

        # loop over the detected face locations and their corresponding
        # locations
        for (box, pred) in zip(locs, preds):
                # unpack the bounding box and predictions
                (startX, startY, endX, endY) = box
                (mask, withoutMask) = pred

                # determine the class label and color we'll use to draw
                # the bounding box and text
                label = "Mask" if mask > withoutMask else "No Mask"
                color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

                # include the probability in the label
                label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)

                # display the label and bounding box rectangle on the output
                # frame
                cv2.putText(frame, label, (startX, startY - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
                cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)

        # show the output frame
        cv2.imshow("Frame", frame)
        key = cv2.waitKey(1) & 0xFF

        # if the `q` key was pressed, break from the loop
        if key == ord("q"):
                break
```