UNIVERSITY OF REGINA

FACULTY OF GRADUATE STUDIES AND RESEARCH

DEPARTMENT OF COMPUTER SCIENCE

CS 815 COMPUTER VISION

PROJECT REPORT

Face Mask Detection Using TensorFlow, Keras and OpenCV

TEJASWINI BALGURI (TBM067@UREGINA.CA)

Contents

1 Introduction **1**

2 Background **1**

3 Approach **2**

3.1 Train Face Mask Detector 2

3.1.1 Load face mask dataset 3

3.1.2 Train face mask classifier with TensorFlow/Keras 3

3.1.3 Serialize face mask classifier 5

3.2 Apply Face Mask Detector 5

3.2.1 Apply Face Mask Detector on an image 5

3.2.2 Apply Face Mask Detector in a real-time video stream 6

4 Code **6**

5 Results **9**

6 Challenges **11**

7 Limitations and Possible Improvements **11**

8 Conclusions **12**

9 References **13**

1 Introduction

The COVID-19 pandemic has quickly disrupted global trade and travel, affecting our day-to-day lives. The virus, which was initially identified in Wuhan, China in December 2019, has now spread across the world. Many people are being affected and infected by this disease. It is now a cause for a large number of deaths across the world. People started using a face mask for protection. The Centers for Disease Control and Prevention (CDC) in the US advise people to use masks. Face mask detection has therefore become a vital task to aid the global society. In this project, I'll implement a face mask detector with Keras/Tensor flow, Open CV, and Deep Learning. I would like to implement the project in two different phases. In the first phase, a face mask classifier is trained using a dataset that contains images with and without masks and in the second phase, the face mask classifier is applied to different images to determine if a face in the image has a mask or not.

2 Background

To work on this project, one should have some knowledge on tensorflow and keras and how a deep learning model can be trained.

1. **TensorFlow**

   TensorFlow is a software library developed by Google and is mainly used in training a deep neural network.[4]

2. **Keras**

   Keras is a software library which provides python interface for artificial neural networks and it acts as an interface for TensorFlow library.[5]

3. **Neural network**

   A neural network mimics the way of how neurons work in a human brain. A neural network contains set of neurons in different layers (one input layer, one output layer and one or more hidden layers) that are interconnected. It helps in identifying the underlying relationship in a set of data.[7]

Convolutional Neural network (CNN) is a type of Neural network that is mainly used for visual images. MobileNetV2 is one of the examples of CNN which performs better in mobile devices.

4. **Optimizer**

   An optimizer can be considered as a function that modifies the neural network's attributes such as the learning rate and weights so that the loss can be reduced and the accuracy of the network can be improved.

5. **Hyperparameters**

   Hyperparameters are the parameters that are set before training the model or network. These parameters are modified during the model training so that the trained model yields better accuracy.

3 Approach

The project can be divided into two different phases.

1. Train Face Mask Detector

2. Apply Face Mask Detector

3.1 Train Face Mask Detector

In this phase, the face mask detector has to be trained using a dataset which can then be used to detect the face mask in images and also in real-time video streams. This phase is sub-divided into following sub-phases.

3.1.1 Load face mask dataset

To train the COVID-19 face mask detector, I used a dataset that contains 1376 images, of which, 690 are the images where the faces are with mask and rest are the images with no mask. The aim of this project is to create a deep learning model to detect whether a person wears a mask or not.

3.1.2 Train face mask classifier with TensorFlow/Keras

In this project, I fine-tuned MobileNetV2, a convolutional neural network architecture that performs better on embedded devices. The following steps are performed in training a face mask classifier.

1. **Import packages**

   Firstly, we import all the necessary packages. Keras, which is an open source python library, is used to create deep learning models. The scikit-learn package which helps in binarizing the class, splitting the dataset, is also imported. The matplotlib module is used to plot the graphs and imutils package is used to list the images from the dataset. Before importing the necessary packages, we need to make sure that the packages are installed.

2. **Setting Hyperparameters**

   Hyperparameters are the parameters that are used while training a machine learning model and is used to control the learning process of the model. To train the MobileNetV2, I used the learning rate, batch size and number of training epochs as the hyperparameters. The path of the dataset and the path where the model and plot has to created are also set.

3. **Loading and Preprocessing the dataset**

   Once the hyperparameters are set, the next step is to load and pre-process the images. All the images from the dataset are loaded and preprocessed. During the preprocessing, the images are resized to 224x224 pixels and are converted to array and also the pixel intensities are scaled to the range [-1, 1]. The data and label lists are initialized where the data contains the images in the form of array and the label contains if the corresponding image has a face with a mask or not.

4. **One hot encoding**

   Once the labels and data are initialized, the labels are encoded using one-hot encoding. The one hot encoding is a process of converting the categorical values to binary values. In our project, there are two categorical values: image with mask and image without mask. Thus, these values are converted to 0 or 1 where 0 means the image with no mask and 1, an image with mask. The one hot encoding boosts the prediction of models.

5. **Data Splitting and Augmentation**

   As a thumb rule, 80% of the dataset should be used for training the model and 20% should be used for testing the model. The data is then augmented, which means that while training the model, on-the-fly mutations are applied on the images i.e., the images are either rotated or flipped or zoomed to improve the model generalization.

6. **Fine-tune model**

   Once the data augmentation is set aside, the model has to be fine-tuned before it can be trained. The following steps are performed in fine-tuning process.

   (a) The MobileNet network without the head is loaded along with the pre-trained image net weights.

   (b) Now, the head is constructed and then appended to the base of the network.

   (c) The network base layers are freezed so that during the backpropagation only the head layer weights are updated.

7. **Compile and train the model**

   Once the model is fine-tuned, it is now ready to be trained. The model is first compiled using

Adam optimizer and. binary cross-entropy. Once the model is compiled, it is then trained using the training data and augmentation.

8. **Evaluate the model**

   Once the model is trained, it is then evaluated using the test data. I used Accuracy as the performance metric.

3.1.3 Serialize face mask classifier

Once the model is trained and evaluated, the serialized model is saved to the disk. The model which is saved can be used predict if the image has a face with a mask or not.

3.2 Apply Face Mask Detector

The face mask detector that is trained is applied on the images and also in the real-time video stream.

3.2.1 Apply Face Mask Detector on an image

Similar to the model training, we need to import few packages to apply the mask detector on the image. I used OpenCV to display and manipulate the images and also tensorflow and keras to pre-process the input image. I used caffe (Convolutional Architecture for Fast Feature Embedding) model, which is a deep learning framework that is used to create image classification models. The caffe model and face mask classifier model are loaded and then the input image is pre-processed, where a copy of image is made and the input image is resized and mean subtraction is performed. A face detection is performed where all the faces in the image are localized. We then loop through all the detected faces and the confidence is extracted, which is compared with the threshold confidence (I have chosen 0.5 for which the model works best). A bounding box is then computed where the face falls within the box. Once the faces are bounded in a box, we perform the following steps.

1. Using NumPy, the face ROI is extracted and pre-processed
2. Mask detection is performed to predict the face with or without mask
3. Based on the probability computed by the model, the output is appropriately displayed on the image

3.2.2 Apply Face Mask Detector in a real-time video stream

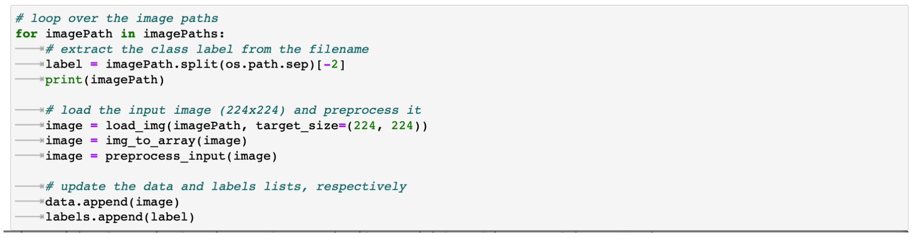
The only difference between the static image and real-time video stream is that in the latter, the model is applied on every frame. I used VideoStream class of imutils package to turn on the webcam and start the video streaming.

4 Code

I created the following three files in my project.

1. **train_mask_detector.ipynb**

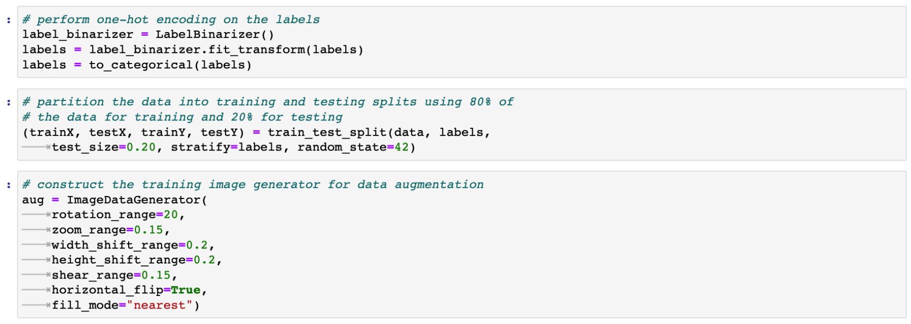
   This file is used to train the mask detector. The following are few snapshots of the file.



```
# loop over the image paths
for imagePath in imagePaths:
	# extract the class label from the filename
	label = imagePath.split(os.path.sep)[-2]
	print(imagePath)

	# load the input image (224x224) and preprocess it
	image = load_img(imagePath, target_size=(224, 224))
	image = img_to_array(image)
	image = preprocess_input(image)

	# update the data and labels lists, respectively
	data.append(image)
	labels.append(label)
```

Figure 1: Code that preprocess dataset images

The code in figure 1 loops over different images in the dataset and preprocess each image.



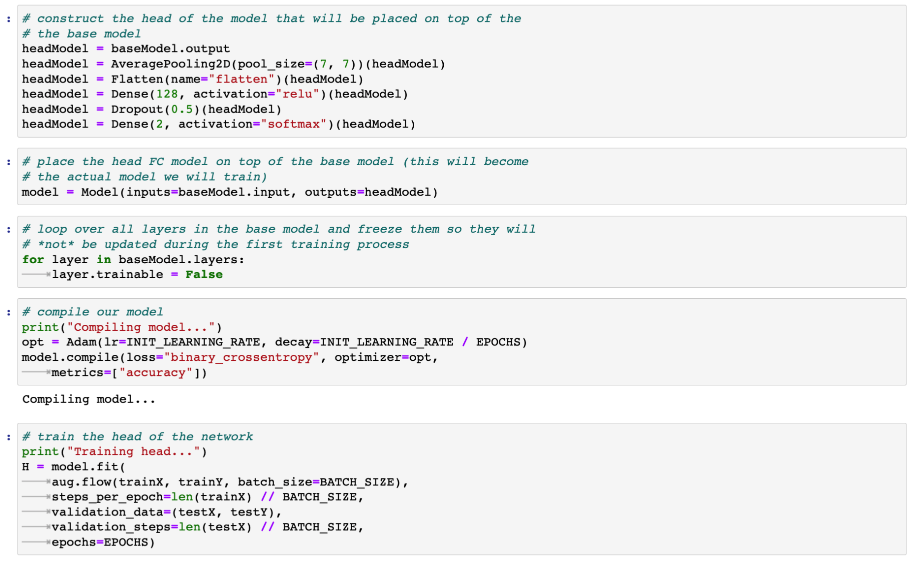
```
# perform one-hot encoding on the labels
label_binarizer = LabelBinarizer()
labels = label_binarizer.fit_transform(labels)
labels = to_categorical(labels)
```

```
# partition the data into training and testing splits using 80% of
# the data for training and 20% for testing
(trainX, testX, trainY, testY) = train_test_split(data, labels,
    test_size=0.20, stratify=labels, random_state=42)
```

```
# construct the training image generator for data augmentation
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")
```

Figure 2: Code that performs data splitting and augmentation

The code in figure 2 performs one-hot encoding, splits the dataset into training and test data and creates an augmentation.



```
# construct the head of the model that will be placed on top of the
# the base model
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(128, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="softmax")(headModel)
```

```
# place the head FC model on top of the base model (this will become
# the actual model we will train)
model = Model(inputs=baseModel.input, outputs=headModel)
```

```
# loop over all layers in the base model and freeze them so they will
# *not* be updated during the first training process
for layer in baseModel.layers:
    layer.trainable = False
```

```
# compile our model
print("Compiling model...")
opt = Adam(lr=INIT_LEARNING_RATE, decay=INIT_LEARNING_RATE / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
    metrics=["accuracy"])

Compiling model...
```

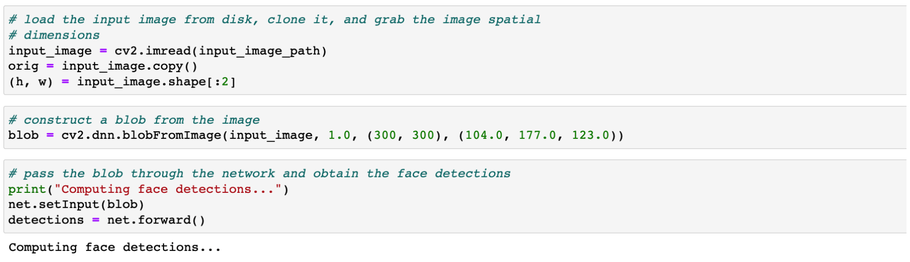
```
# train the head of the network
print("Training head...")
H = model.fit(
    aug.flow(trainX, trainY, batch_size=BATCH_SIZE),
    steps_per_epoch=len(trainX) // BATCH_SIZE,
    validation_data=(testX, testY),
    validation_steps=len(testX) // BATCH_SIZE,
    epochs=EPOCHS)
```

Figure 3: MobileNetV2 code

The code in figure 3 is more specific to MobileNetV2 where the head of the model is constructed and placed on the top of the base model and then the model is compiled and trained using the training dataset.

1. **detect_mask_image.ipynb**

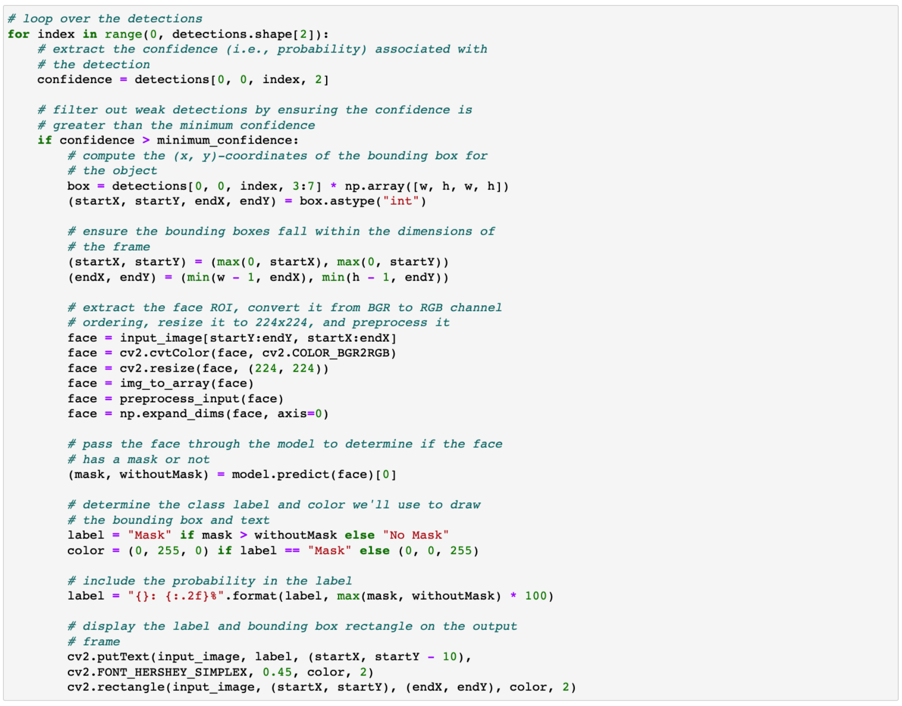
   This file is used to detect if a face contains a mask or not using the trained mask detector. The following are few snapshots of the file.



```python
# load the input image from disk, clone it, and grab the image spatial
# dimensions
input_image = cv2.imread(input_image_path)
orig = input_image.copy()
(h, w) = input_image.shape[:2]

# construct a blob from the image
blob = cv2.dnn.blobFromImage(input_image, 1.0, (300, 300), (104.0, 177.0, 123.0))

# pass the blob through the network and obtain the face detections
print("Computing face detections...")
net.setInput(blob)
detections = net.forward()
```

```
Computing face detections...
```

Figure 4: Code that preprocess the input image

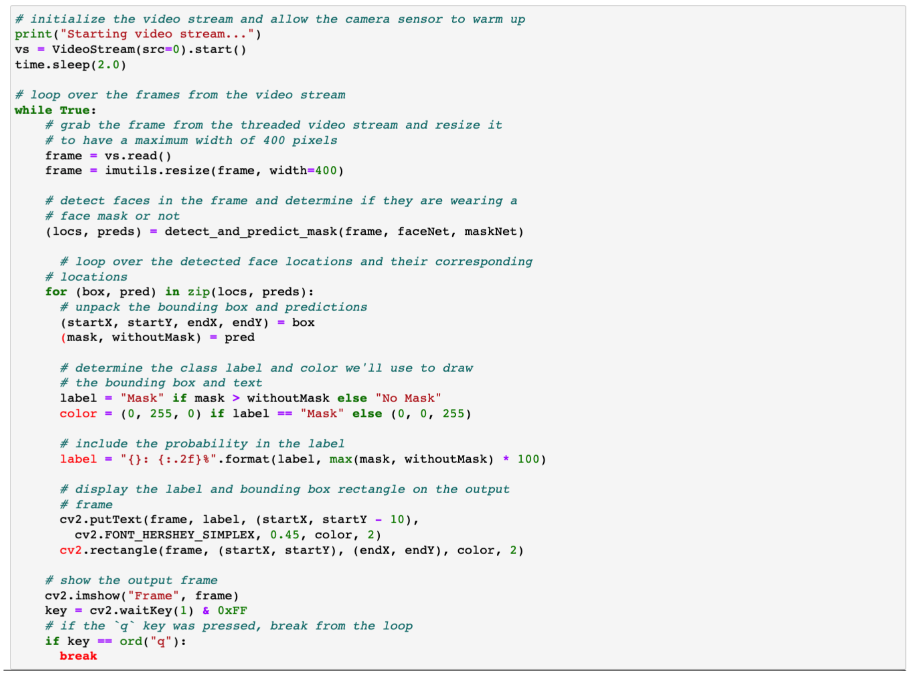


```python
# loop over the detections
for index in range(0, detections.shape[2]):
    # extract the confidence (i.e., probability) associated with
    # the detection
    confidence = detections[0, 0, index, 2]

    # filter out weak detections by ensuring the confidence is
    # greater than the minimum confidence
    if confidence > minimum_confidence:
        # compute the (x, y)-coordinates of the bounding box for
        # the object
        box = detections[0, 0, index, 3:7] * np.array([w, h, w, h])
        (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(w - 1, endX), min(h - 1, endY))

        # extract the face ROI, convert it from BGR to RGB channel
        # ordering, resize it to 224x224, and preprocess it
        face = input_image[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)
        face = np.expand_dims(face, axis=0)

        # pass the face through the model to determine if the face
        # has a mask or not
        (mask, withoutMask) = model.predict(face)[0]

        # 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(input_image, label, (startX, startY - 10),
        cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
        cv2.rectangle(input_image, (startX, startY), (endX, endY), color, 2)
```

Figure 5: Code that detects faces in input image

1. **detect_mask_video.ipynb**

   This file streams the real-time video through webcam and detects if the face has a mask or not for each frame. Figure 6 contains the code that loops through each frame and applies the model.



```
# initialize the video stream and allow the camera sensor to warm up
print("Starting video stream...")
vs = VideoStream(src=0).start()
time.sleep(2.0)

# 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=400)

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

      # 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
```

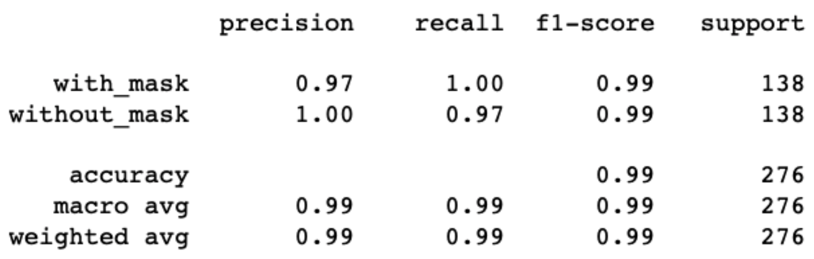
Figure 6: Code that detects face in each frame

5 Results

The following are the results that are obtained in this project.

1. **train_mask_detector.ipynb**

   Figure 7 shows the classification report of the trained model. The report is used to measure the quality of predictions of the trained model. Figure 8 shows the graph between the number of epochs and loss/accuracy. The graph depicts that the accuracy steadily increases and the loss steadily decreases to an extent when the number of epochs are increased.



```
              precision    recall  f1-score   support

   with_mask       0.97      1.00      0.99       138
without_mask       1.00      0.97      0.99       138

    accuracy                           0.99       276
   macro avg       0.99      0.99      0.99       276
weighted avg       0.99      0.99      0.99       276
```

Figure 7: Classification Report

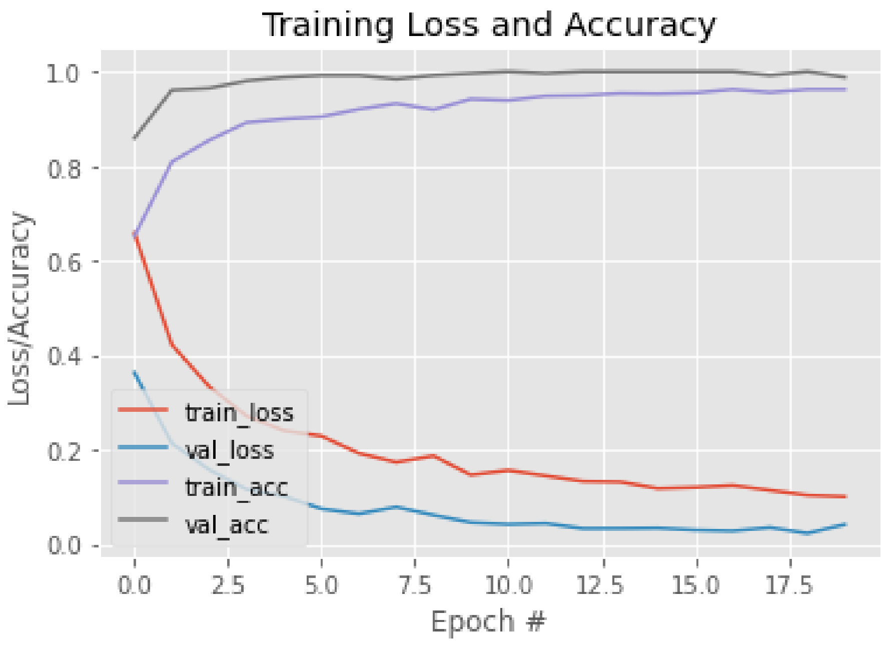
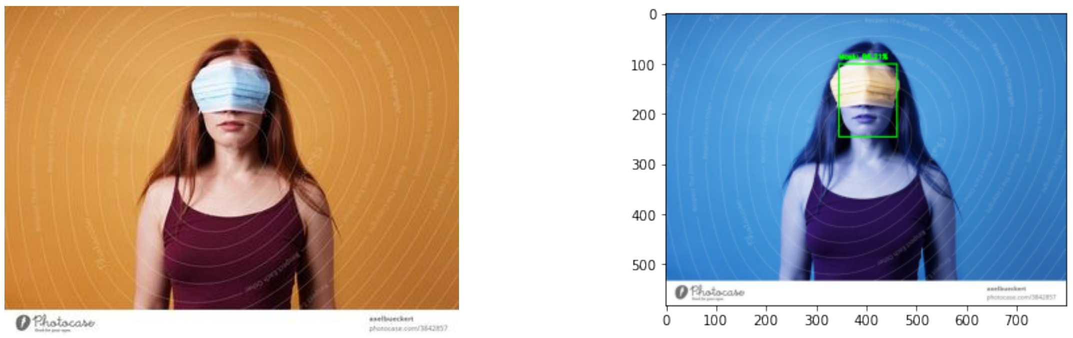


Figure 8: Training Loss and Accuracy

2. **detect_mask_image.ipynb**

Figure 9 shows the input image to the model. Since the input image contains a face with a mask, figure 10 shows the input image with the predicted output (mask) by the model along the corresponding probability.



(a) The input image to the model (b) Input image with predicted output by model

Figure 9: Input image along with the predicted value of model

3. **detect_mask_video.ipynb**

Figure 10 shows the screenshot captured during the real-time video stream. Since I'm wearing a mask, the model detected the mask along with the probability (99.83%).

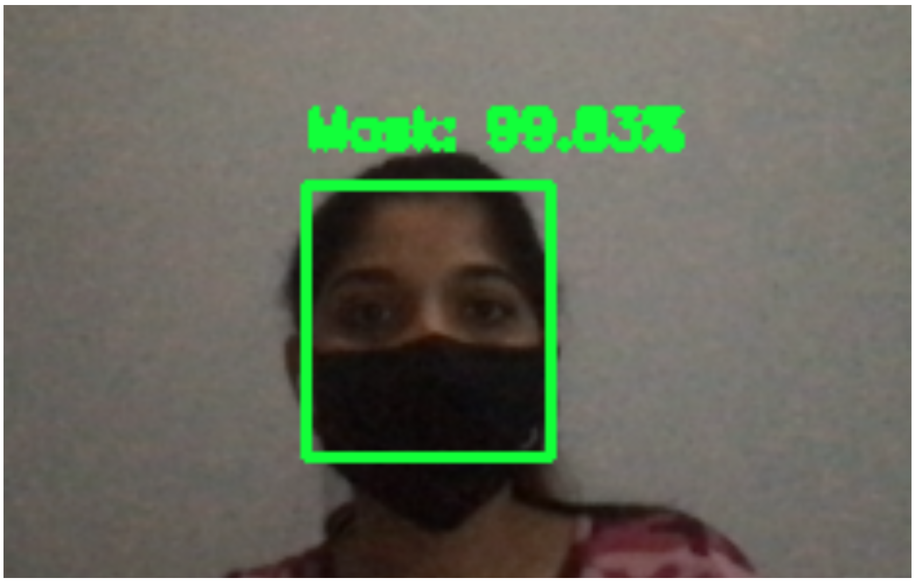


Figure 10: Screenshot captured during the real-time video stream

6 Challenges

One of the challenges I faced in this project is finding out the right dataset to train the mask detector. I haven't worked on machine/deep learning before and working on it for the first time is quite challenging with respect to hyperparameter tuning and using the right optimizer.

7 Limitations and Possible Improvements

The following are the limitations of the project.

1. **Dataset Size**

   I have considered only 1400 images in the dataset which is of smaller size.

2. **Masked images in the dataset**

   The dataset I considered has the masked images that are artificially created.

The project can be improved by considering the huge set of data for training the model. Considering the huge set of data will improve the performance of the model. Rather than considering the artificially generated masked images, the actually faces with the masks are to be considered to make the model predict effectively.

8 Conclusions

To sum up, I trained a MobileNetV2 CNN architecture on a dataset that contains images with faces with a mask and without a mask. Once the model is trained, using the trained model, I predicted if a face in an image contains a mask or not and also in the real-time video stream. I used Accuracy as a performance metric and also I used classification report that tells us how well the model works on the test dataset.

9 References

[1] Goyal, H., Sidana, K., Singh, C. et al. A real time face mask detection system using convolutional neural network. Multimed Tools Appl 81, 14999–15015 (2022). https://doi.org/10.1007/s11042-022-12166-x

[2] Boulila, Wadii & Alzahem, Ayyub & Almoudi, Aseel & Afifi, Muhanad & Alturki, Ibrahim & Driss, Maha. (2021). A Deep Learning-based Approach for Real-time Facemask Detection.

[3] Nagoriya, Harsh & Parekh, Mohak. (2021). Live Facemask Detection System. International Journal of Imaging and Robotics. 21. 1-8.

[4] https://www.tensorflow.org/

[5] https://keras.io/

[6] https://keras.io/api/applications/mobilenet/

[7] https://en.wikipedia.org/wiki/Neural_network