

Name: Nouran Ahmed Abd Elhameed
Mohamed Saad
ID: 18P4496

Face-mask-detection Project

Deep Learning
CSE 485

Table of Contents

GitHub link	3
Introduction	3
Problem Statement	3
Applications.....	4
Implementation	5
• Dataset	5
• Pre-processing.....	6
• Architecture of MobileNetV2 (Detection of mask)	6
• FaceNet (Detection of faces).....	7
• Requirements to be installed.	8
• Screenshots of the files	8
➤ Mask_detect.py.....	8
➤ face-mask-detect.py.....	12
Testing	16
Results:.....	18
Comparison between other architectures	22
References.....	23

GitHub link

https://github.com/Nouran-saad/Face_mask_detectionDL.git

Introduction

In the face of the COVID-19 pandemic, public health measures such as wearing face masks have become paramount in preventing the spread of the virus. As communities adapt to the "new normal," technological advancements have played a crucial role in enforcing mask-wearing protocols and ensuring public safety. Among these advancements, face mask detection systems have emerged as a powerful tool to monitor and enforce compliance with face mask regulations in various settings, including public spaces, workplaces, and transportation hubs.

The development and implementation of face mask detection systems have significant implications for public health and safety. By automating the process of monitoring mask-wearing, these systems can help reduce the burden on human resources and enhance the efficiency of mask compliance enforcement.

Problem Statement

Implementing a face mask detection system requires integrating multiple technologies, including computer vision, machine learning, and deep learning. The goal is to create an automated system that can analyse video frames, identify faces, and accurately classify whether a person is wearing a mask, wearing it incorrectly, or not wearing a mask at all. The system should provide real-time feedback to ensure timely intervention and enforcement of mask-wearing protocols.

To address this problem, we will utilize Python, Keras, and OpenCV, which are popular tools and libraries for computer vision and deep learning tasks. By leveraging pre-trained deep learning models and image processing techniques, we can develop an efficient and effective face mask detection system that operates on real video streams.



Applications

The face mask detection project has a wide range of applications across various domains. Some of the key applications of face mask detection include:

1. **Healthcare Facilities:** Face mask detection can be utilized in hospitals, clinics, and other healthcare settings to reinforce infection control measures. By identifying individuals who are not wearing masks or wearing them incorrectly, healthcare providers can take immediate action to prevent the spread of infections.
2. **Public Spaces and Transportation:** Face mask detection systems can be deployed in public spaces such as shopping malls, airports, train stations, and bus terminals. These systems help monitor and enforce mask-wearing compliance among individuals, ensuring a safer environment for both staff and visitors.
3. **Educational Institutions:** Schools, colleges, and universities can benefit from face mask detection systems to ensure the safety of students, teachers, and staff. These systems can be integrated into entry points or common areas, providing real-time alerts when individuals are not wearing masks correctly.
4. **Workplaces and Offices:** Face mask detection can be implemented in office buildings, factories, and other work environments to ensure that employees adhere to mask-wearing protocols. By automating the monitoring process, these systems help maintain a healthy and productive workplace.
5. **Retail and Hospitality:** Face mask detection systems can be employed in retail stores, restaurants, and hotels to maintain a safe environment for customers and employees. Monitoring mask compliance helps reducing the risk of virus transmission.
6. **Public Transportation:** Face mask detection can be integrated into public transportation systems, such as buses, trains, and subways, to ensure passenger compliance with mask-wearing regulations. This helps protect both passengers and transportation staff and promotes a safer commuting experience.
7. **Public Events and Gatherings:** During events, concerts, or conferences, face mask detection systems can play a vital role in ensuring the safety and well-being of attendees. By quickly identifying individuals without masks, event organizers can take appropriate measures to enforce mask-wearing guidelines.

Implementation

- Dataset

From Kaggle website and here's a sample of the data:

- With mask:



- Without mask:



- Pre-processing

- Pre-process the label so we performed one-hot encoding on the labels by the LabelBinarizer which is a SciKit Learn class that accepts Categorical data as input and returns a NumPy array.
- Perform data augmentation on images to generate new images by ImageDataGenerator, so we rotate our images by 30, zoom range by 0.20, width and height shift by 0.4, apply also horizontal flip and fill mode with nearest and that means This mode fills the newly created pixels with the value of the nearest pixel in the original image. It helps maintain the continuity of the image and is suitable for most cases.

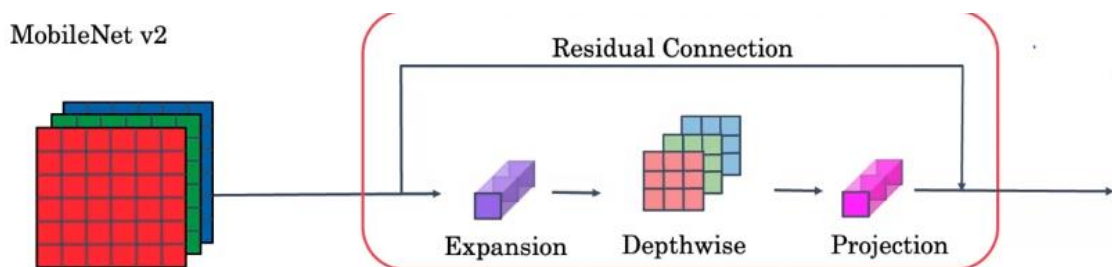
- Architecture of MobileNetV2 (Detection of mask)

MobileNetV2 is a popular convolutional neural network (CNN) architecture designed for efficient image classification and other computer vision tasks on mobile and embedded devices. While it is not specifically designed for the detection of masks, it can be adapted and trained for such a task using transfer learning.

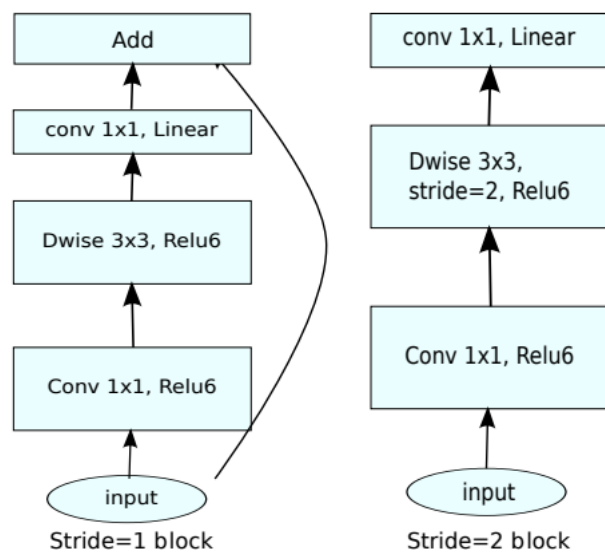
Here is an overview of the MobileNetV2 architecture:

The inverted residual structure is introduced as an alternative to the traditional residual connections. Instead of adding the input feature maps to the output, MobileNetV2 [1] employs a depth-wise convolution which is , it applies a separate filter to each input channel and produces a set of intermediate output channels followed by a point-wise linear projection to change the number of channels and It takes the intermediate output channels from the depth-wise convolution and performs a 1×1 convolution on them. This convolution is responsible for combining and building new features by computing linear combinations of the input channels. This structure reduces the number of parameters and computations compared to traditional residuals.

MobileNetV2 [1] also introduce the concept of linear bottlenecks, which are used to increase the non-linearity of the network without adding additional parameters. By applying a lightweight non-linearity, such as a ReLU (Rectified Linear Unit), after the depth-wise convolution, the network can capture more complex features while keeping the model size small.



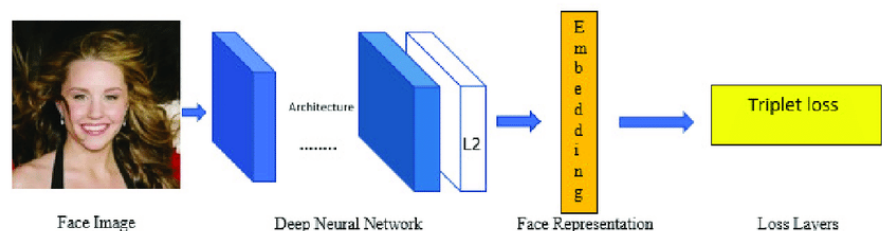
- In MobileNetV2 [1], there are two types of blocks. One is residual block with stride of 1. Another one is block with stride of 2 for downsizing.
- There are 3 layers for both types of blocks.
- This time, the first layer is 1×1 convolution with ReLU6.
- The second layer is the depth-wise convolution.
- The third layer is another 1×1 convolution but without any non-linearity. It is claimed that if ReLU is used again, the deep networks only have the power of a linear classifier on the non-zero volume part of the output domain.
- There is an expansion factor t . And $t=6$ for all main experiments.



- FaceNet (Detection of faces)

FaceNet [2] is a deep learning architecture for face recognition developed by researchers at Google. It uses a deep convolutional neural network (CNN) to extract high-level features from face images and then maps these features into a lower-dimensional space called an embedding. The key idea behind FaceNet is to learn a robust representation of faces that is invariant to variations in pose, lighting conditions, and other factors.

I used pre-trained Caffe models to Detect and classify faces with third-party DNNs which is res10_300x300_ssd_iter_140000. And a prototxt file which is model definition files that are required when you train a model.



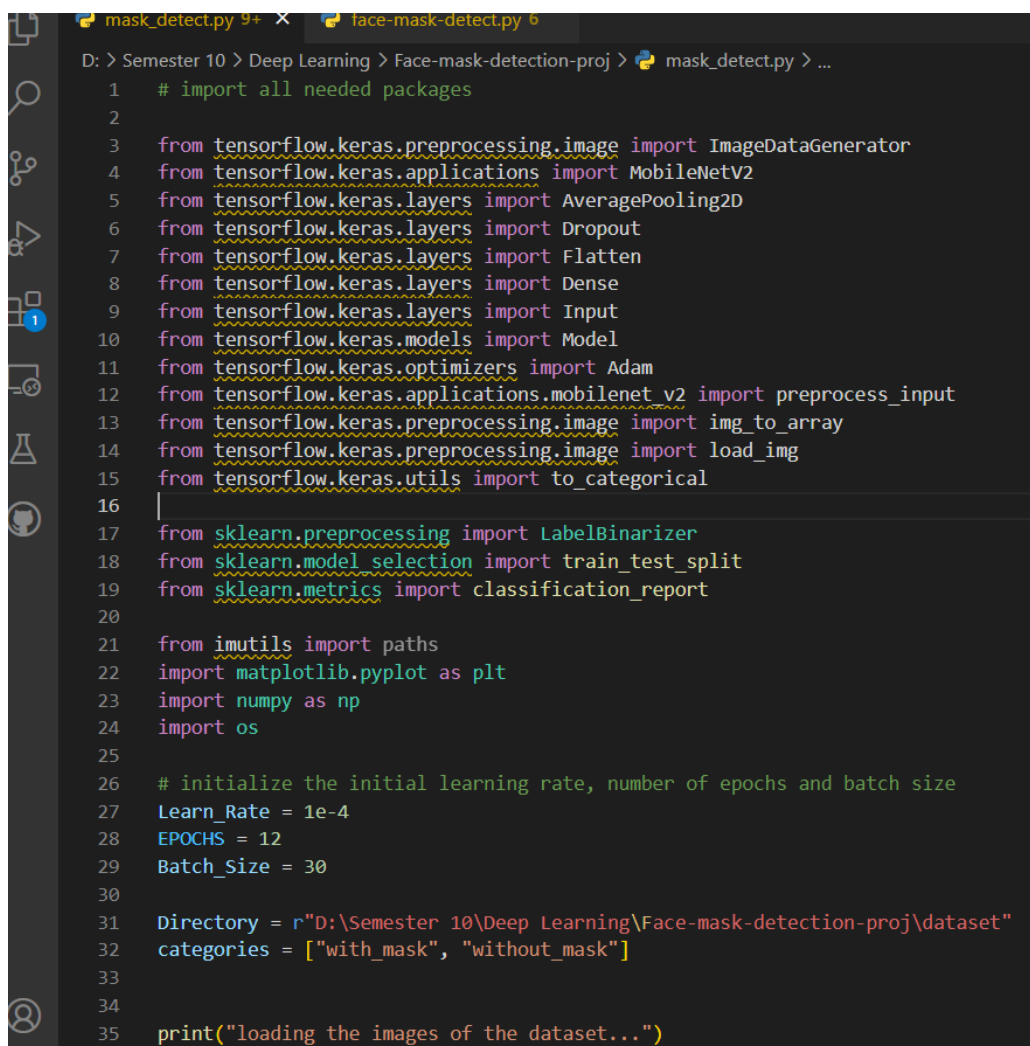
- Requirements to be installed.

```
tensorflow>=1.15.2
keras==2.3.1
imutils==0.5.3
numpy==1.18.2
opencv-python==4.2.0.*
matplotlib==3.2.1
scipy==1.4.1
```

- Screenshots of the files

➤ Mask_detect.py

First, import all the needed packages (TensorFlow, sklearn, imutils, NumPy and os) then initialize the learning rate, number of epochs and batch size then assign the directory of the dataset and the 2 types of categories (with Mask and Without Mask).



```
mask_detect.py 9+ x face-mask-detect.py 6
D:\Semester 10 > Deep Learning > Face-mask-detection-proj > mask_detect.py > ...
1 # import all needed packages
2
3 from tensorflow.keras.preprocessing.image import ImageDataGenerator
4 from tensorflow.keras.applications import MobileNetV2
5 from tensorflow.keras.layers import AveragePooling2D
6 from tensorflow.keras.layers import Dropout
7 from tensorflow.keras.layers import Flatten
8 from tensorflow.keras.layers import Dense
9 from tensorflow.keras.layers import Input
10 from tensorflow.keras.models import Model
11 from tensorflow.keras.optimizers import Adam
12 from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
13 from tensorflow.keras.preprocessing.image import img_to_array
14 from tensorflow.keras.preprocessing.image import load_img
15 from tensorflow.keras.utils import to_categorical
16
17 from sklearn.preprocessing import LabelBinarizer
18 from sklearn.model_selection import train_test_split
19 from sklearn.metrics import classification_report
20
21 from imutils import paths
22 import matplotlib.pyplot as plt
23 import numpy as np
24 import os
25
26 # initialize the initial learning rate, number of epochs and batch size
27 Learn_Rate = 1e-4
28 EPOCHS = 12
29 Batch_Size = 30
30
31 Directory = r"D:\Semester 10\Deep Learning\Face-mask-detection-proj\dataset"
32 categories = ["with_mask", "without_mask"]
33
34
35 print("loading the images of the dataset...")
```


After converting the data then Split The data into 80 % train and 20 test then perform data augmentation by imageDataGenerator to generate new images so we make the rotation range by 30, zoom range by 0.20, width and height width range by 0.4, shear range by 0.18 and apply horizontal flip and fill mode by the nearest pixel.

Then we make the base Model which is The MobileNetV2 [1] by the pretrained model (imageNet), make sure that FC Layers are left off by setting include_top by false and resize the input image by 224*224 , then build the head model which is average pool 2d by kernel size 7*7 then flatten the output of the pooling then FC layer by 128 neurons and apply the

ReLU function on it then apply dropout to avoid the overfitting then the last layer with 2 output and apply the SoftMax.

Then create our model by placing the head model on top of base model and freeze all layers in the base model as we replaced the standard convolutional neural network. Then we compile our model by the Adam optimizer and the accuracy metric.

```
# load the MobileNetV2 network and make sure the head FC layer sets are left off by setting include_
# imagenet : pretrained model for images and input shape 224*224
baseModel = MobileNetV2(weights="imagenet", include_top=False,
    input_tensor=Input(shape=(224, 224, 3)))

# build 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)
# to avoid the overfitting
headModel = Dropout(0.5)(headModel)
# softmax deal with binary representation
headModel = Dense(2, activation="softmax")(headModel)

# place the head FC model on top of the base model
model = Model(inputs=baseModel.input, outputs=headModel)

#freeze all layers in the base model because we replace the standard convolutional neural networks
for layer in baseModel.layers:
    layer.trainable = False

# compile our model
print("compiling model...")

opt = Adam(lr=Learn_Rate, decay=Learn_Rate / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
    metrics=["accuracy"])
```

Then we train our model by the function fit which is s called to start the training process. It takes the augmented training data batches to enhance the training process and improve the model's ability to generalize to unseen data, the steps per epoch, the validation data, the validation steps, and the number of epochs as input parameters. The **fit ()** method updates the model's weights based on the training data and evaluates its performance on the validation data after each epoch.

Then evaluate our model by using function predict () and it take the tettix for testing the model and the batch size and then we print the classification report then save the model and plot the training loss and accuracy throughout the epochs.

```

mask_detect.py 9+ X face-mask-detect.py 6
D: > Semester 10 > Deep Learning > Face-mask-detection-proj > mask_detect.py > ...
102 print("training head...")
103 train_head = model.fit(
104     aug.flow(train_x, train_y, batch_size=Batch_Size),
105     steps_per_epoch=len(train_y) // Batch_Size,
106     validation_data=(test_x, test_y),
107     validation_steps=len(test_x) // Batch_Size,
108     epochs=EPOCHS)
109
110 # Evaluate our model
111 print("evaluating the model...")
112 predIdxs = model.predict(test_x, batch_size=Batch_Size)
113
114
115 # find the index of the label with maximum predicted probability for every image
116 predIdxs = np.argmax(predIdxs, axis=1)
117
118 # show a nicely formatted classification report
119 print(classification_report(test_y.argmax(axis=1), predIdxs,
120     target_names=lb.classes_))
121
122 # save the model
123 print("saving mask detector model...")
124 model.save("mask_detection.model", save_format="h5")
125 # plot the training loss and accuracy
126 No_Of_Epochs = EPOCHS
127 plt.style.use("ggplot")
128 plt.figure()
129 plt.plot(np.arange(0, No_Of_Epochs), train_head.history["loss"], label="train_loss")
130 plt.plot(np.arange(0, No_Of_Epochs), train_head.history["val_loss"], label="val_loss")
131 plt.plot(np.arange(0, No_Of_Epochs), train_head.history["accuracy"], label="train_acc")
132 plt.plot(np.arange(0, No_Of_Epochs), train_head.history["val_accuracy"], label="val_acc")
133 plt.title("Training Loss and Accuracy")
134 plt.xlabel("Epoch #")
135 plt.ylabel("Loss/Accuracy")
136 plt.legend(loc="lower left")
137 plt.savefig("loss_plot.png")

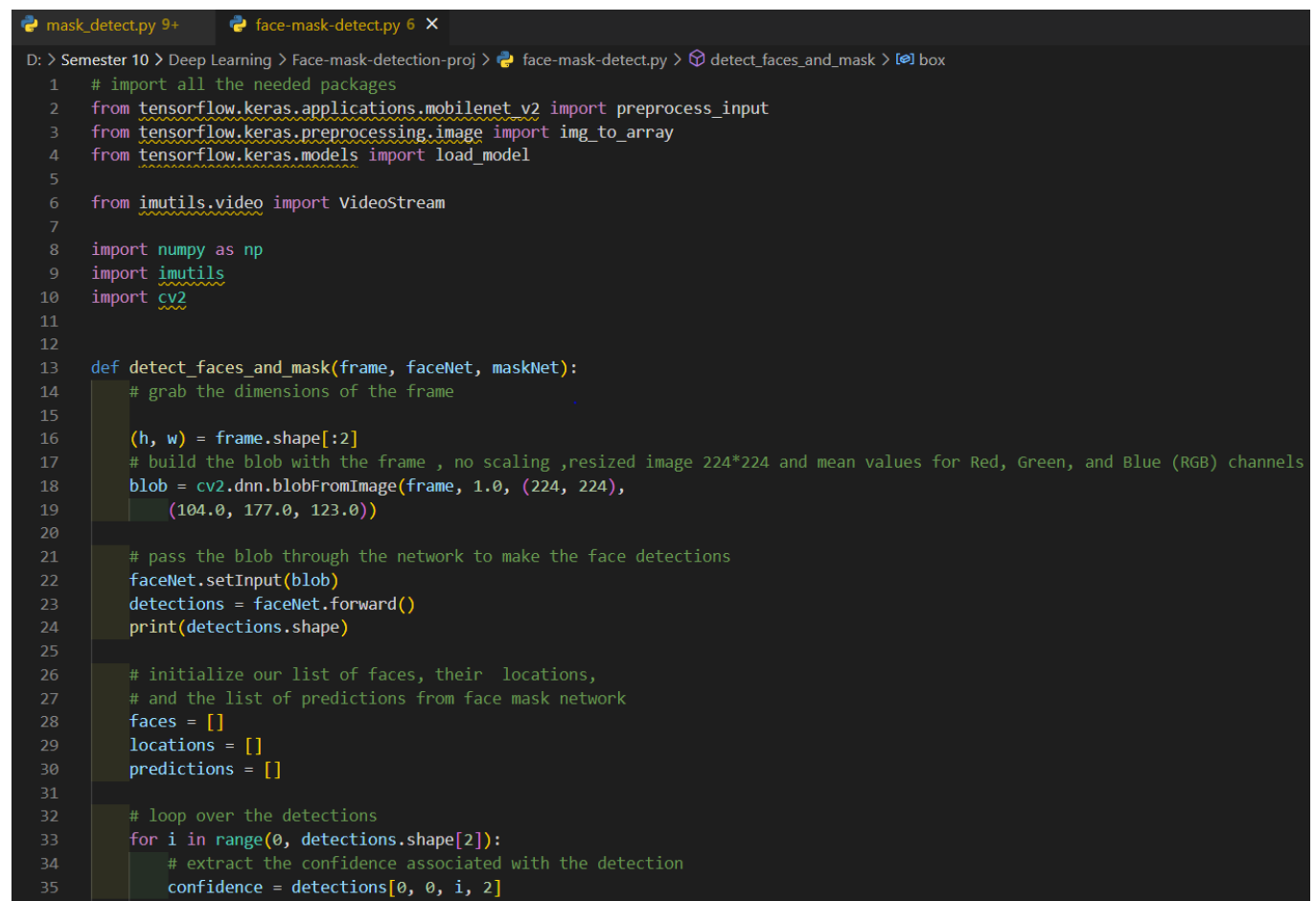
```

➤ face-mask-detect.py

First, import all the needed packages by TensorFlow keras and imutils, NumPy, cv2 then define the function of detect_faces_and_mask () that take the frame, faceNet pretrained model and maskNet model that we saved it from another python file as mask detection.

At the function we grab the dimensions of the frame and build the blob with the frame, no scaling, resized image 224*224 and mean values for Red, Green, and Blue (RGB) channels then pass the blob through the network to make the face detection.

Then initialize the list of faces and their locations and list of predictions from face mask network and loop over the detections to extract the confidence associated with the detection.



```
D: > Semester 10 > Deep Learning > Face-mask-detection-proj > face-mask-detect.py > detect_faces_and_mask > box
1  # import all the needed packages
2  from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
3  from tensorflow.keras.preprocessing.image import img_to_array
4  from tensorflow.keras.models import load_model
5
6  from imutils.video import VideoStream
7
8  import numpy as np
9  import imutils
10 import cv2
11
12
13 def detect_faces_and_mask(frame, faceNet, maskNet):
14     # grab the dimensions of the frame
15
16     (h, w) = frame.shape[:2]
17     # build the blob with the frame , no scaling ,resized image 224*224 and mean values for Red, Green, and Blue (RGB) channels
18     blob = cv2.dnn.blobFromImage(frame, 1.0, (224, 224),
19     (104.0, 177.0, 123.0))
20
21     # pass the blob through the network to make the face detections
22     faceNet.setInput(blob)
23     detections = faceNet.forward()
24     print(detections.shape)
25
26     # initialize our list of faces, their locations,
27     # and the list of predictions from face mask network
28     faces = []
29     locations = []
30     predictions = []
31
32     # loop over the detections
33     for i in range(0, detections.shape[2]):
34         # extract the confidence associated with the detection
35         confidence = detections[0, 0, i, 2]
```

Then if detection confidence greater than the minimum confidence which is 0.5 keep it else filter it out then if keep it we compute the x and y coordinates of the bounding box then extract the face ROI and convert it from BGR to RGB then order it and resize it to 224*224 and pre-process it then add the face and bounding box to their lists.

```

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

    # if detection greater than the minimum confidence keep it else filter it out
    if confidence > 0.5:
        # compute the (x, y)-coordinates of the bounding box.
        box = detections[0, 0, i, 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 = 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 the lists
        faces.append(face)
        locations.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

    faces = np.array(faces, dtype="float32")
    predictions = maskNet.predict(faces, batch_size=30)

```

Then if at least there is one face was detected we make the batch predictions on all faces at the same time in the for loop by the function predict () which is takes the faces and batch size 30 then we return a 2-tuple of the face locations and their locations.

Then we read the faceNet model which pre-trained model at the caffe library and then load the mask detection model then initialize video stream and loop over every frame in the video stream and grab the frame from the video stream and resize it to have a max width of 400 pixels or read images then call the function of detect_faces_and_mask () to determine if they are wearing a face mask or not then loop over the detect face locations and unpack the bounding box and the predictions then determine the class label and colour the bounding box with label 'Mask' with blue and 'No Mask' with Red

```

75 # load the face mask detector model
76 maskNet = load_model("mask_detection.model")
77
78 # initialize the video stream
79 #print("starting video stream...")
80 #vs = VideoStream(src=0).start()
81
82 # read img by cv2 library
83 img = cv2.imread("group.jpeg", cv2.IMREAD_COLOR)
84
85 # loop over every frames in the video stream
86 while True:
87     # grab the frame from the video stream and resize it to have a maximum width of 400 pixels
88     #frame = vs.read()
89     frame = imutils.resize(img, width=400)
90
91     # detect faces then determine if they are wearing a face mask or not
92     (locations, predictions) = detect_faces_and_mask(frame, faceNet, maskNet)
93
94     # loop over the detected face locations and the bounding box predictions
95     for (box, prediction) in zip(locations, predictions):
96         # unpack the bounding box and predictions
97         (startX, startY, endX, endY) = box
98         (mask, No_mask) = prediction
99
100        # determine the class label and color the bounding box with label 'Mask' with blue and 'No Mask' with Red
101        label = "Mask" if mask > No_mask else "No Mask"
102        color = (255, 0, 0) if label == "Mask" else (0, 0, 255)
103
104        # include the max probability between mask and No_mask in the label
105        label = "{}: {:.2f}%".format(label, max(mask, No_mask) * 100)

```

This is the case of reading images.

```

76 maskNet = load_model("mask_detection.model")
77
78 # initialize the video stream
79 print("starting video stream...")
80 vs = VideoStream(src=0).start()
81
82 # read img by cv2 library
83 #img = cv2.imread("group.jpeg", cv2.IMREAD_COLOR)
84
85 # loop over every frames in the video stream
86 while True:
87     # grab the frame from the video stream and resize it to have a maximum width of 400 pixels
88     frame = vs.read()
89     frame = imutils.resize(frame, width=400)
90
91     # detect faces then determine if they are wearing a face mask or not
92     (locations, predictions) = detect_faces_and_mask(frame, faceNet, maskNet)
93
94     # loop over the detected face locations and the bounding box predictions
95     for (box, prediction) in zip(locations, predictions):
96         # unpack the bounding box and predictions
97         (startX, startY, endX, endY) = box
98         (mask, No_mask) = prediction
99
100        # determine the class label and color the bounding box with label 'Mask' with blue and 'No Mask' with Red
101        label = "Mask" if mask > No_mask else "No Mask"
102        color = (255, 0, 0) if label == "Mask" else (0, 0, 255)
103
104        # include the max probability between mask and No_mask in the label
105        label = "{}: {:.2f}%".format(label, max(mask, No_mask) * 100)

```

This is the case of reading video streams.

Then include the max probability between mask and No_mask in the label and # display the label and bounding box rectangle on the result frame then show the output frame and finally # if the `q` key was pressed, break from the loop

```
104     # include the max probability between mask and No_mask in the label
105     label = "{}: {:.2f}%".format(label, max(mask, No_mask) * 100)
106
107     # display the label and bounding box rectangle on the result frame
108     cv2.putText(frame, label, (startX, startY - 10),
109                 cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
110     cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)
111
112     # show the output frame
113     cv2.imshow("Frame", frame)
114     key = cv2.waitKey(1) & 0xFF
115
116     # if the `q` key was pressed, break from the loop
117     if key == ord("q"):
118         break
119
120     # cleanup
121     cv2.destroyAllWindows()
122     #vs.stop()
123     # To hold the window on screen, we use cv2.waitKey method
124     cv2.waitKey(0)
```

Testing

I trained my model by 12 epochs and here is the result of every epoch in the training process.

```
Command Prompt
compiling model...
training head...
Epoch 1/12
102/102 [=====] - 79s 775ms/step - loss: 0.3974 - accuracy: 0.8165 - val_loss: 0.0957 - val_acc
uracy: 0.9844
Epoch 2/12
102/102 [=====] - 84s 822ms/step - loss: 0.1900 - accuracy: 0.9239 - val_loss: 0.0625 - val_acc
uracy: 0.9857
Epoch 3/12
102/102 [=====] - 484s 5s/step - loss: 0.1453 - accuracy: 0.9473 - val_loss: 0.0503 - val_accu
racy: 0.9857
Epoch 4/12
102/102 [=====] - 91s 894ms/step - loss: 0.1205 - accuracy: 0.9559 - val_loss: 0.0409 - val_acc
uracy: 0.9883
Epoch 5/12
102/102 [=====] - 92s 898ms/step - loss: 0.1144 - accuracy: 0.9572 - val_loss: 0.0366 - val_acc
uracy: 0.9896
Epoch 6/12
102/102 [=====] - 83s 812ms/step - loss: 0.0937 - accuracy: 0.9648 - val_loss: 0.0319 - val_acc
uracy: 0.9909
Epoch 7/12
102/102 [=====] - 82s 803ms/step - loss: 0.0884 - accuracy: 0.9657 - val_loss: 0.0306 - val_acc
uracy: 0.9922
Epoch 8/12
102/102 [=====] - 91s 892ms/step - loss: 0.0746 - accuracy: 0.9743 - val_loss: 0.0279 - val_acc
uracy: 0.9909
Epoch 9/12
102/102 [=====] - 95s 929ms/step - loss: 0.0848 - accuracy: 0.9707 - val_loss: 0.0263 - val_acc
uracy: 0.9935
Epoch 10/12
```

Then we can see the classification report with the precision, recall, f1-score, and support with says that our model is very good.

```
Epoch 10/12
102/102 [=====] - 85s 838ms/step - loss: 0.0749 - accuracy: 0.9740 - val_loss: 0.0272 - val_acc
uracy: 0.9909
Epoch 11/12
102/102 [=====] - 95s 930ms/step - loss: 0.0759 - accuracy: 0.9717 - val_loss: 0.0268 - val_acc
uracy: 0.9922
Epoch 12/12
102/102 [=====] - 79s 776ms/step - loss: 0.0731 - accuracy: 0.9743 - val_loss: 0.0295 - val_acc
uracy: 0.9909
evaluating the model...
      precision    recall  f1-score   support

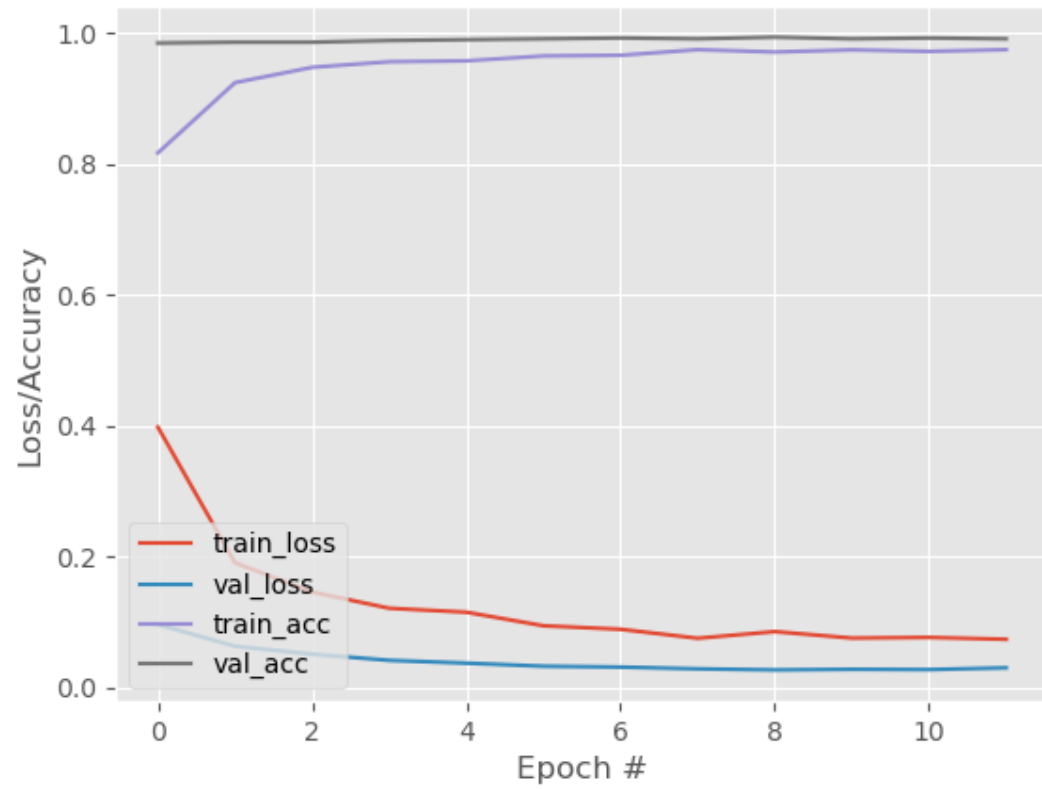
with_mask      0.99      0.99      0.99       383
without_mask    0.99      0.99      0.99       384

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

saving mask detector model...
```

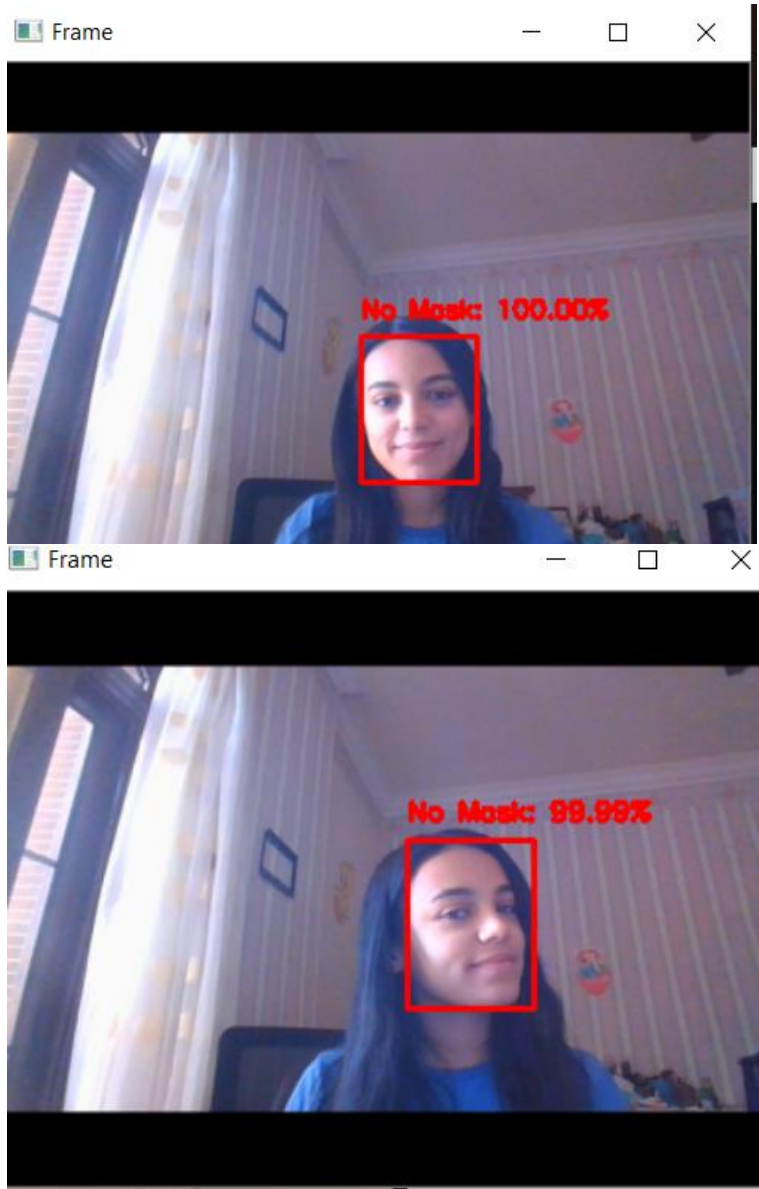
And we can also see that at the graph of loss and accuracy throughout the 12 epochs and it seems the loss is decreasing, and the accuracy is increasing which is a very good model.

Training Loss and Accuracy

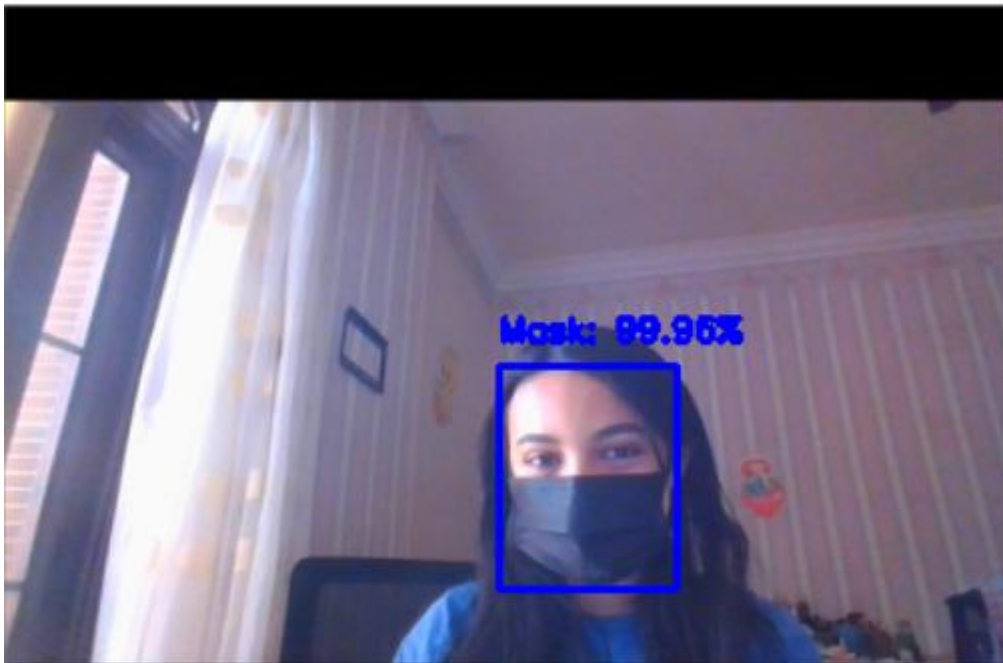


Results:

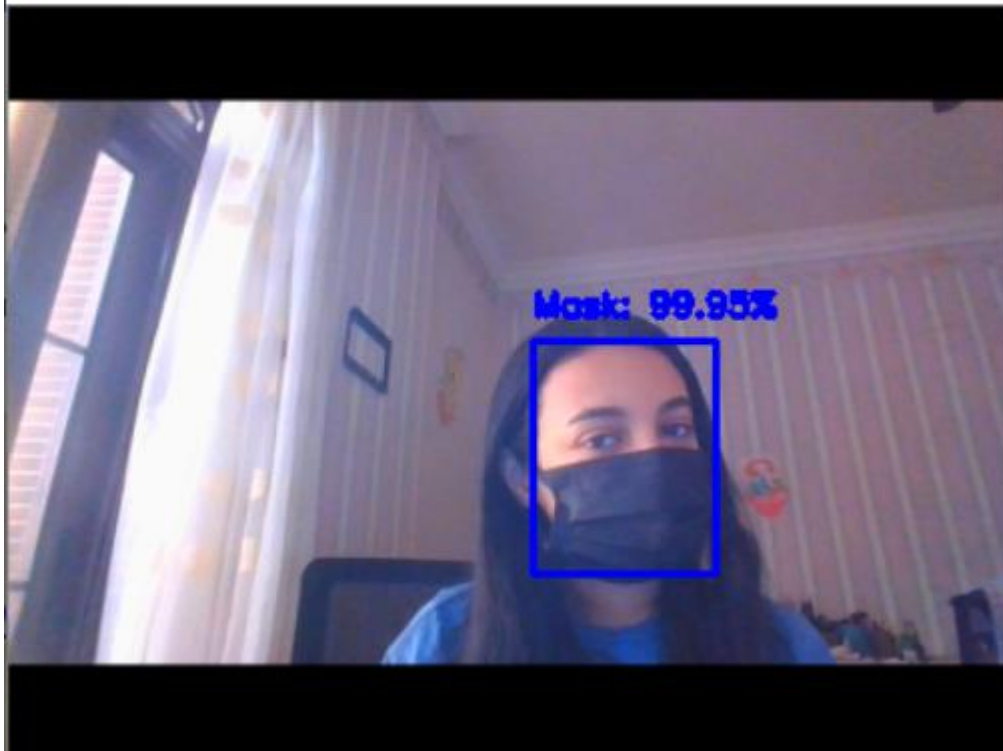
- On real time video streams with mask and no mask.

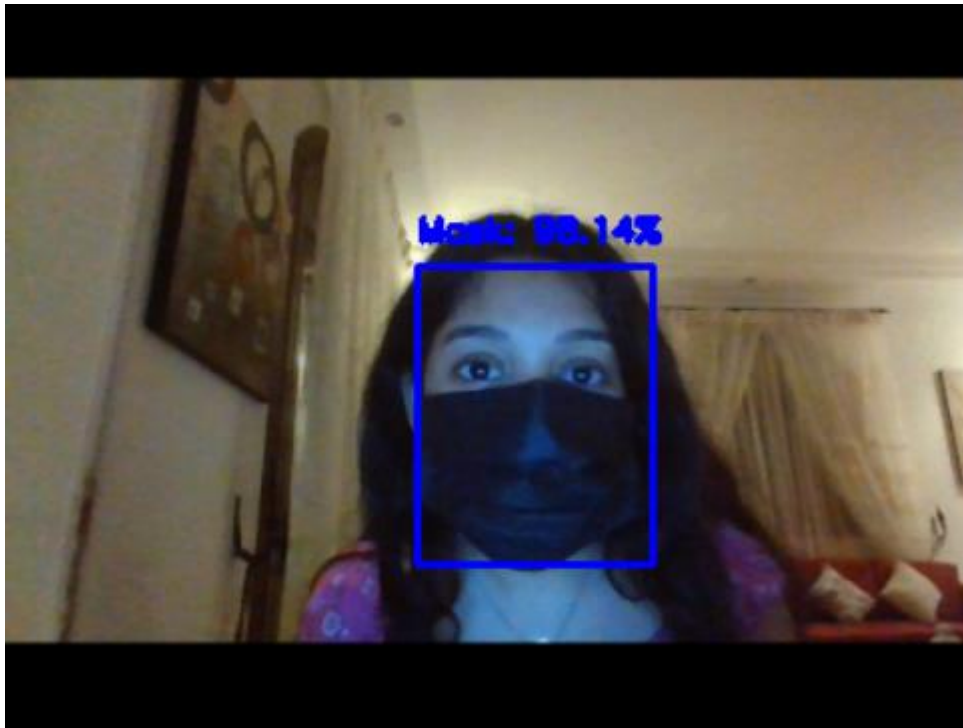


Frame

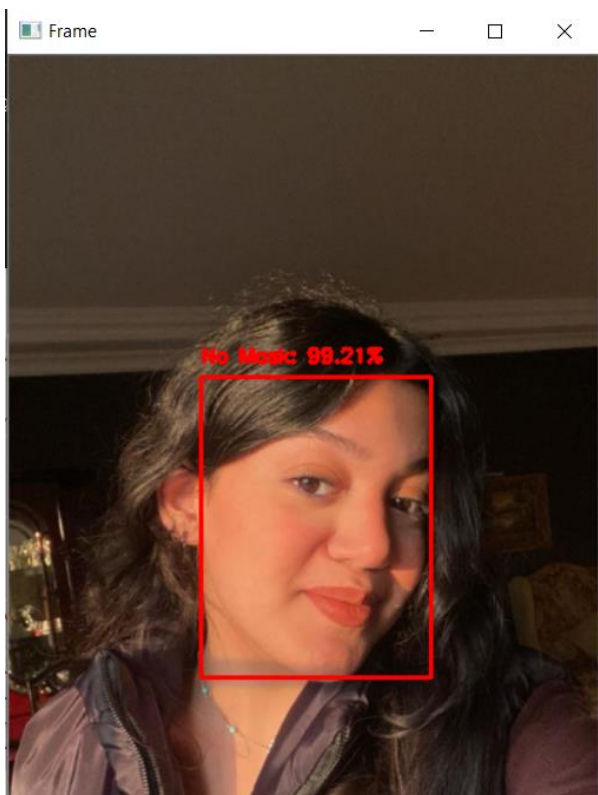


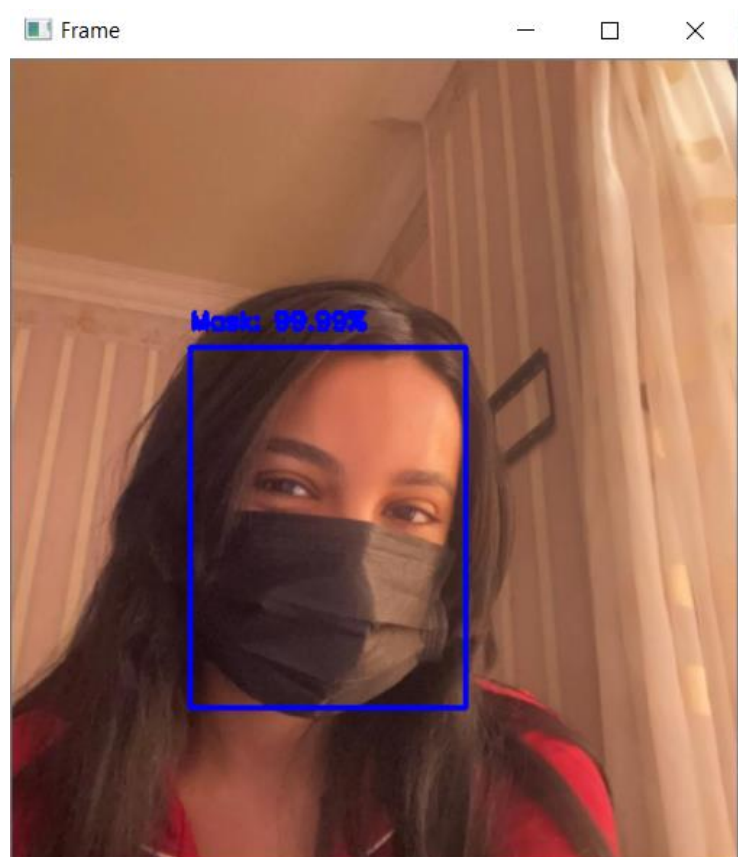
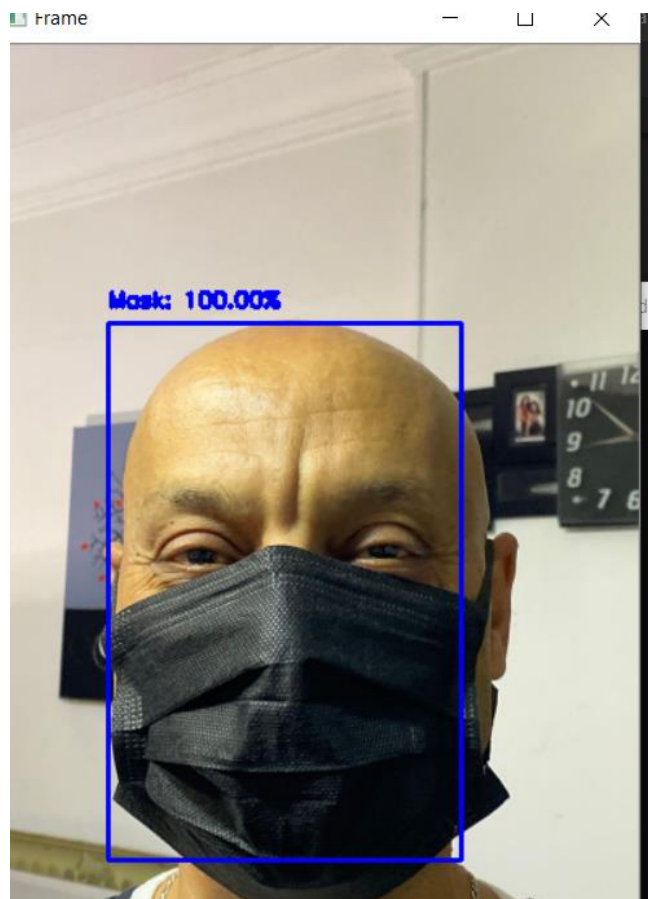
Frame





➤ On images with mask and no mask.





Comparison between other architectures

- MobileNetV2 [1]:
 - Pros: MobileNetV2 is lightweight and efficient, making it suitable for deployment on resource-constrained devices. It offers a good balance between model size, speed, and accuracy.
 - Cons: Due to its reduced complexity, MobileNetV2 may have limitations in capturing intricate details and handling complex patterns compared to deeper architectures.
- ResNet [3]:
 - Pros: ResNet's residual connections enable effective training of very deep networks and help capture intricate features. It has achieved state-of-the-art performance on various computer vision tasks.
 - Cons: Deeper ResNet architectures may be computationally expensive and require a larger amount of training data to prevent overfitting.
- DenseNet [4]:
 - Pros: DenseNet's dense connectivity promotes feature reuse and enhances gradient flow throughout the network. It allows for efficient parameter usage and has shown strong performance with limited training data.
 - Cons: DenseNet architectures may have higher memory requirements compared to other models due to the dense connections.
- Inception [5]:
 - Pros: Inception models capture information at multiple spatial resolutions, allowing them to extract both local and global features effectively. They have demonstrated strong performance in various computer vision tasks.
 - Cons: Inception architectures can be computationally expensive and may require more resources during training and inference compared to simpler models.

References

- [1] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," 2019.
- [2] F. Schroff, D. Kalenichenko and J. Philbin, "FaceNet: A Unified Embedding for Face Recognition and Clustering," 2015.
- [3] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2015.
- [4] G. Huang, Z. Liu and L. van der Maaten, "Densely Connected Convolutional Networks," 2018.
- [5] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going deeper with convolutions," 2014.