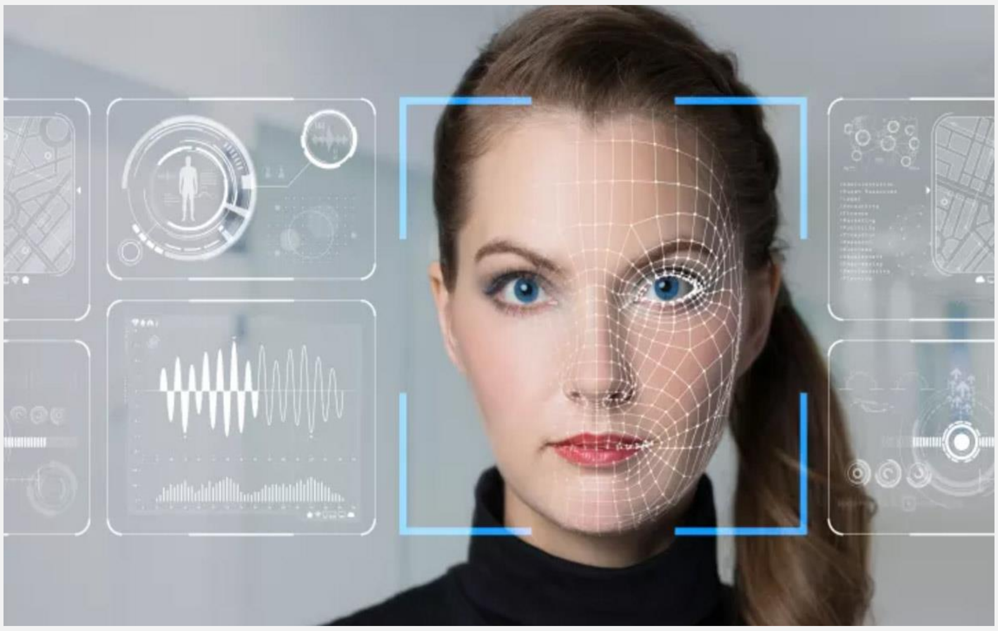
औद्योगिक प्रशिक्षण के लिए राष्ट्रीय संस्थान

National Institute for Industrial Training

One Premier Organization with Non Profit Status | Registered Under Govt. of WB
Empanelled Under Planning Commission Govt. of India
Inspired By: National Task Force on IT & SD Government of India

National Institute for Industrial Training- One Premier Organization with Non Profit Status Registered Under Govt.of West Bengal,Empanelled Under Planning Commission Govt.of India , Empanelled Under Central Social Welfare Board Govt. of India , Registered with National Career Services , Registered with National Employment Services.

Python with Artificial Intelligence



Subject: Face Mask Detection

Submitted By: Subham Kundu

Submitted To: Soumotanu Mazumdar Sir

CONTENT

ACKNOWLEDGEMENT

I would like to express my deepest appreciation to all those who provided me the possibility to complete this project. I give special gratitude to my project guider, Soumotanu Mazumdar Sir, whose contribution in stimulating ideas and encouragement helped me coordinate my project. Furthermore, I would also like to acknowledge with much appreciation the crucial role of the National Institute for Industrial Training, which gave the permission to use all required equipment and the necessary materials to complete the course “Artificial Intelligence with Python”. I have put tremendous effort in this project. However, it would not have been possible without the kind support and help of above-mentioned individual and organization. I would like to extend my sincere thanks to all of them.

STUDENT PROFILE

Name: Subham Kundu

College: Techno India University

Stream: B. Tech CSE

Email: subhamkundu486@gmail.com

LinkedIn: https://www.linkedin.com/in/subham-kundu-10654994/

GitHub Profile: https://github.com/subhamrex

Project: Face Mask Detection

Project Files: https://github.com/subhamrex/Face_mask_detection

INTRODUCTION

Python comes with a huge number of inbuilt libraries. Many of the libraries are for Artificial Intelligence and Machine Learning. Some of the libraries are TensorFlow (which is high-level neural network library), scikit-learn (for data mining, data analysis and machine learning), Keras (which provides a Python interface for artificial neural networks etc). The list keeps going and never ends.

Python has an easy implementation for OpenCV which functions mainly aimed at real-time computer vision. What makes Python favourite for everyone is its powerful and easy implementation. For other languages, students and researchers need to get to know the language before getting into ML or AI with that language. This is not the case with python. Even a programmer with very basic knowledge can easily handle python. Apart from that, the time someone spends on writing and debugging code in python is way less when compared to C, C++ or Java. This is exactly what the students of AI and ML want. They don't want to spend time on debugging the code for syntax errors, they want to spend more time on their algorithms and heuristics related to AI and ML. Not just the libraries but their tutorials, handling of interfaces are easily available online. People build their own libraries and upload them on GitHub or elsewhere to be used by others.

Hardware and software requirements

Software Requirements:

Operating System: Windows/Linux

Programming Language: Python 3.8

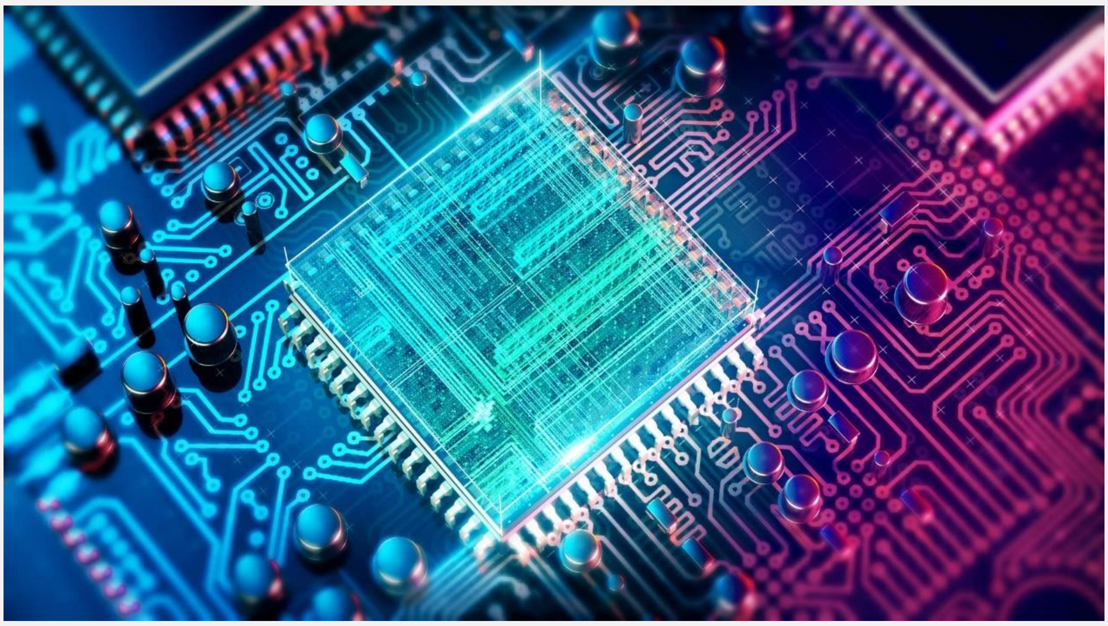
Software (IDE): PyCharm or VSCode

Hardware requirements:

Speed: 233MHz and above

Hard disk: 10GB

RAM: 512 MB

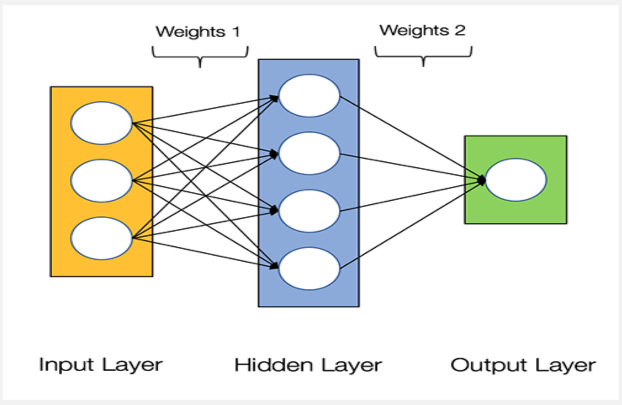


Objective & Procedure

Now-a-days Face Mask Detection has a great importance for our world. In pandemic mask is essential stuff for safety. It will basically help not to spread diseases. So, my **main objective** is to create a Face Mask detection model which will help us to detect a person wears a mask or not.

Python is a powerful open source programming language, which means that it's free to use while having all the properties that a programming language should have. Python has some 72,000 libraries in the Python Package Index that aid in scientific calculations and machine learning applications. In this project I have used Python as a programming language.

Firstly, I have used **TensorFlow** which is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. TensorFlow computations are expressed as stateful dataflow graphs. The name TensorFlow derives from the operations that such neural networks perform on multidimensional data arrays, which are referred to as tensors.



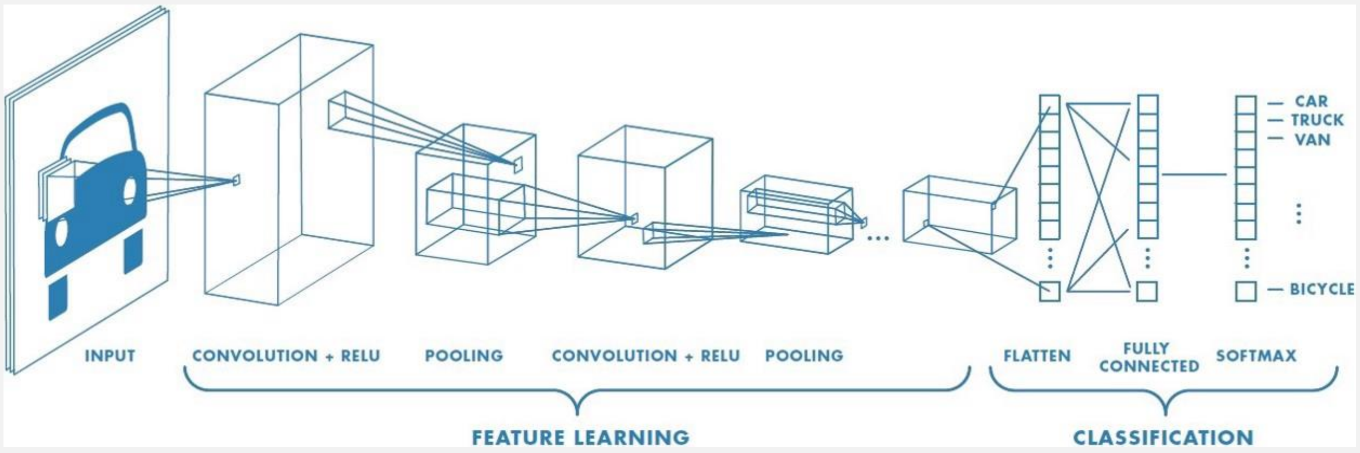
Most models are made of layers. Layers are functions with a known mathematical structure that can be reused and have trainable variables. In TensorFlow, most high-level implementations of layers and models, such as Keras. It has three layers:

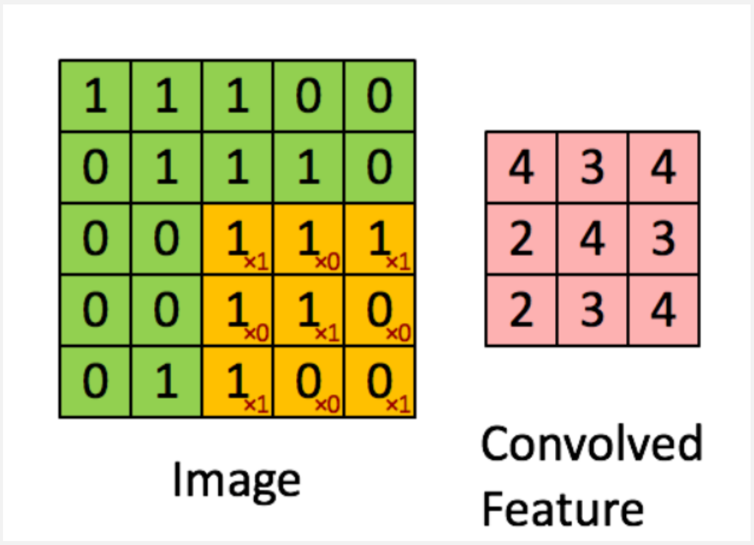
1. **Input layers**: The **input layer** of a neural network is composed of artificial **input** neurons, and brings the initial data into the system for further processing by subsequent **layers** of artificial neurons.

2. **Hidden layers**: A **hidden layer** is located between the input and output of the algorithm, in which the function applies weights to the inputs and directs them through an activation function as the output.

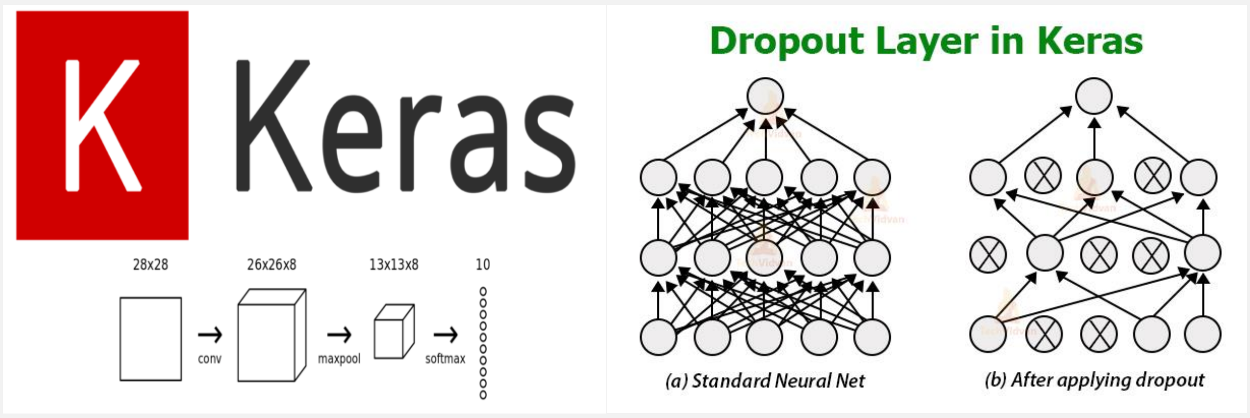
3.**Output layers**: The **output layer** is responsible for producing the final result. The **output layer** takes in the inputs which are passed in from the **layers** before it, performs the calculations via its neurons and then the **output** is computed.

Besides, I have used **Convolutional Neural Network (CNN)** which is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a CNN is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, CNN have the ability to learn these filters/characteristics. The role of the CNN is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction. I did Convolution operation on a 244x244x3 image matrix with a 3x3x3 Kernel.





Secondly, I have used **Keras** which is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library.



Lastly, I have used **OpenCV** which is a library of programming functions mainly aimed at real-time computer vision. Using this library, I have load face detection module and mask detection model in a video stream. Specially It help us to show a proper output with a video frame.

My Approach will be to create a Train Deep learning model (MobileNetV2) and apply mask detector over images / live video stream.

* Initially, I have to create a train_mask_detector.py file which will help us to train our data.

train_mask_detector.py =>

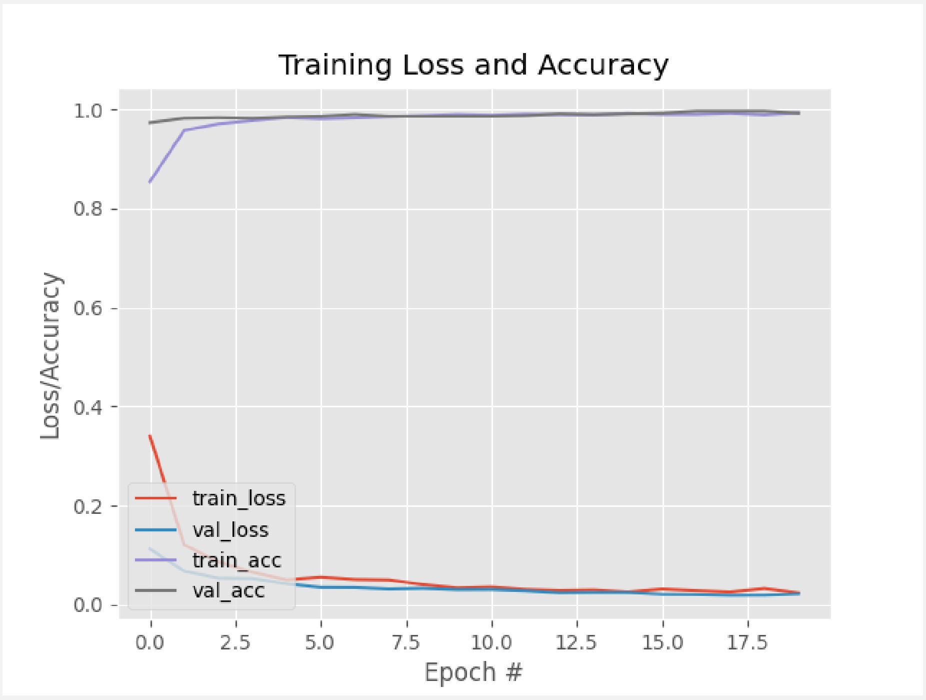
```
# import the necessary packages
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from 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
import matplotlib.pyplot as plt
import numpy as np
import os

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

DIRECTORY = r"dataset"
CATEGORIES = ["with_mask", "without_mask"]
```

```
# grab the list of images in our dataset directory, then initialize the list of data (i.e., images) and class
#images
print("[INFO] loading images...")


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))

            image = img_to_array(image)

            image = preprocess_input(image)

            data.append(image)

            labels.append(category)

# perform one-hot encoding on the labels

lb = LabelBinarizer()

labels = lb.fit_transform(labels)

labels = to_categorical(labels)


data = np.array(data, dtype="float32")

labels = np.array(labels)

# Here I have taken 1930 images as train datasets and 1930 images as test datasets.


(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")

# load the MobileNetV2 network, ensuring the head FC layer sets are left off
# Center cropping the image with the pixel value of 224x224x3
baseModel = MobileNetV2(weights="imagenet", include_top=False,
	input_tensor=Input(shape=(224, 224, 3)))

# construct the head of the model that will be placed on top of 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)

# So here our input layer is baseModel and headModel is output layer.
# 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("[INFO] compiling model...")
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
```

```
model.compile(loss="binary_crossentropy", optimizer=opt,
	metrics=["accuracy"])

# train the head of the network
print("[INFO] training head...")
H = model.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)

# make predictions on the testing set
print("[INFO] evaluating network...")
predIdxs = model.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
predIdxs = np.argmax(predIdxs, axis=1)

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

# serialize the model to disk
print("[INFO] saving mask detector model...")
model.save("mask_detector.model", save_format="h5")

# plot the training loss and accuracy
N = EPOCHS
plt.style.use("ggplot")
plt.figure()

plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")

plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")

plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")

plt.plot(np.arange(0, N), H.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")
```

Output:

It will give a **mask_detector.model** as trained model. Besides It will provide a plot related to training loss and accuracy.



To check our model, I have created a **detect_mask_video.py** file.

detect_mask_video.py =>

```
------------------------------------------------------------------------------------------------------------------------

# import the 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

import numpy as np

import time

import cv2

import os


def detect_and_predict_mask(frame, faceNet, maskNet):

   # grab the dimensions of the frame and then construct a blob from it

   (h, w) = 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 = []
```

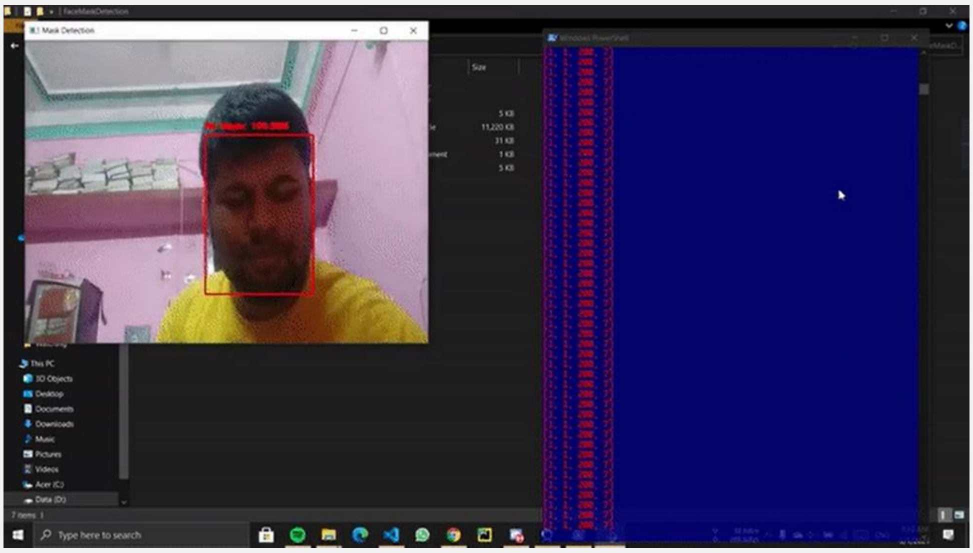
```
# loop over the detections

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([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 region of interest from the face, convert it from BGR to RGB channel

        #ordering, resize it to 224x224, and pre-process 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 prediction 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)


# load our serialized face detector model from disk

prototxtPath = r"face_detector\deploy.prototxt"

weightsPath = r"face_detector\res10_300x300_ssd_iter_140000.caffemodel"

faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)


# load the face mask detector model from disk

maskNet = load_model("mask_detector.model")


# initialize the video stream

print("[INFO] starting video stream...")

# Change camera_id for external webcam

camera_id = 0

vs = cv2.VideoCapture(camera_id)
```

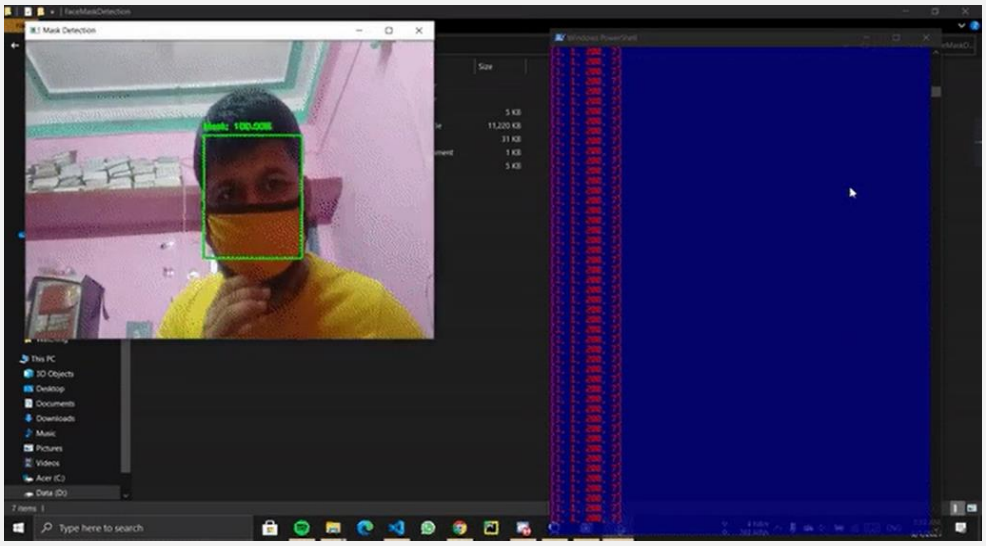
```
# loop over the frames from the video stream
while True:
    success, frame = vs.read()
    # detect faces in the frame and determine if they are wearing a face mask or not
    (locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)

    # 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("Mask Detection", frame)

    key = cv2.waitKey(1) & 0xFF

    # if the `q` key was pressed, break from the loop

    if key == ord("q"):

        break

# do a bit of clean-up

cv2.destroyAllWindows()
```

Output:

It will give us frame where our video stream will be opened and concurrently it will detect a person wears a mask or not with the help of computer vision (OpenCV) and deep learning algorithm (TensorFlow and Keras).



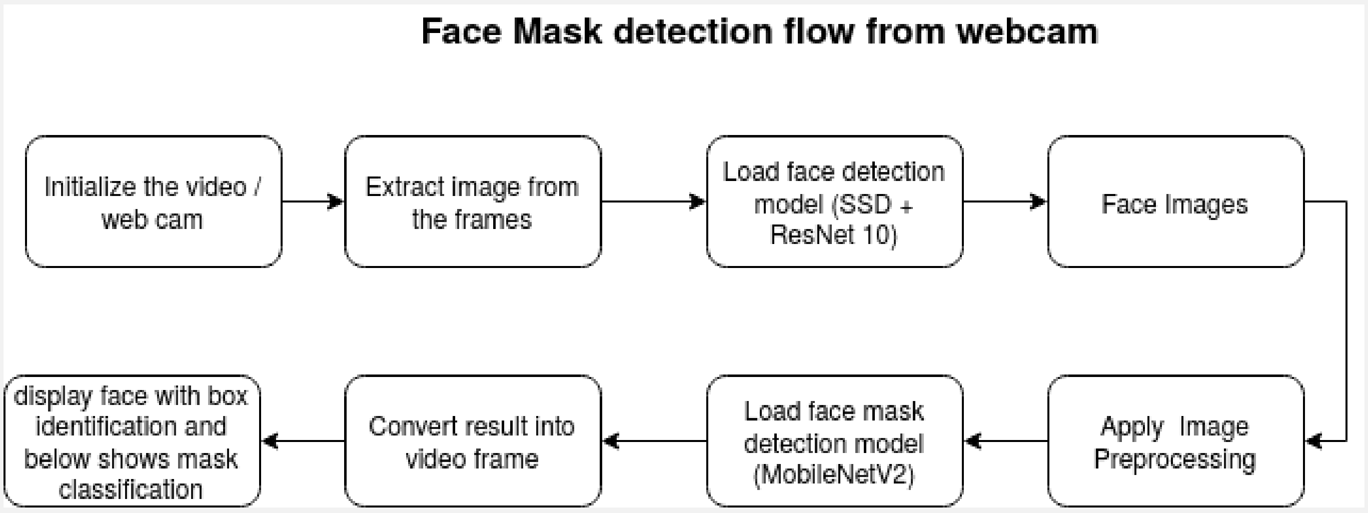
No mask



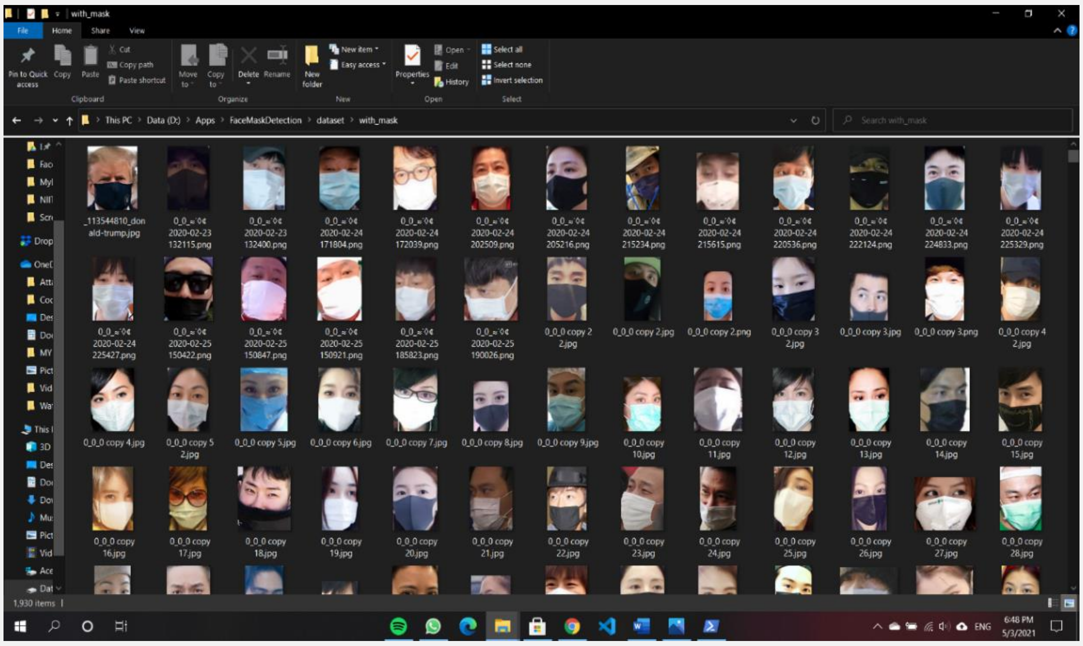
With mask

[Click here to see output]

Flowchart:

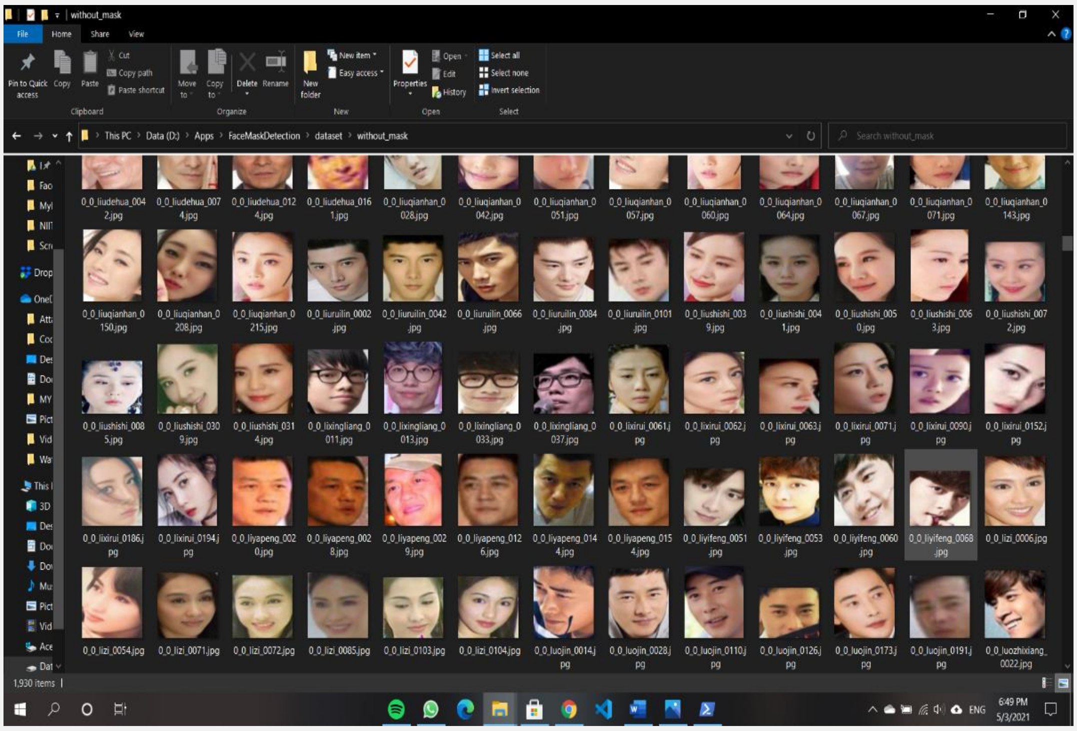

Face Mask detection flow from webcam
Initialize the video / web cam
Extract image from the frames
Load face detection model (SSD + ResNet 10)
Face Images
Apply Image Preprocessing
Load face mask detection model (MobileNetV2)
Convert result into video frame
display face with box identification and below shows mask classification

Used Datasets:



with_mask
This PC › Data (D:) › Apps › FaceMaskDetection › dataset › with_mask
1,930 items

With mask



without_mask
This PC › Data (D:) › Apps › FaceMaskDetection › dataset › without_mask
1,930 items

Without mask

Here I have 1930 images with mask and 1930 images without mask.

Update Project files: After running my project in cmd, I thought It would be better if It has a user interface. That’s why I have done some little changes in my project. I have developed a web app using flask. Here are my updated codes:

mask_detect_video_app.py =>

```
# import the 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
import numpy as np
import time
import cv2
import os

class VideoCamera(object):
    def __init__(self):
        self.video = cv2.VideoCapture(0)
        # load our serialized face detector model from disk
        self.prototxtPath = r"face_detector\deploy.prototxt"
        self.weightsPath = r"face_detector\res10_300x300_ssd_iter_140000.caffemodel"
        self.faceNet = cv2.dnn.readNet(self.prototxtPath, self.weightsPath)
        self.maskNet = load_model("mask_detector.model")
    def __del__(self):
        self.video.release()

    def detect_and_predict_mask(self,frame):
        # grab the dimensions of the frame and then construct a blob from it
        (h, w) = 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
```

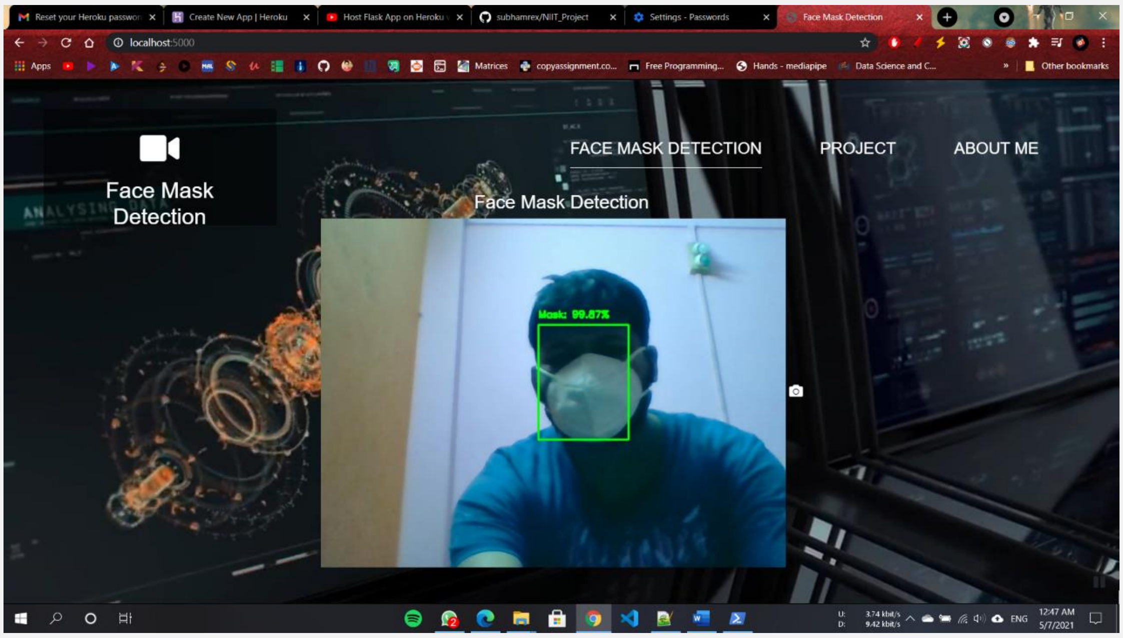
```
self.faceNet.setInput(blob)
detections = self.faceNet.forward()
# initialize our list of faces, their corresponding locations,
# and the list of predictions from our face mask network
faces = []
locs = []
preds = []

# 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([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 their respective lists
      faces.append(face)
      locs.append((startX, startY, endX, endY))

# only make a prediction 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 = self.maskNet.predict(faces, batch_size=32)

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

  def get_frame(self):
    ret, frame = self.video.read()
    (locs, preds) = self.detect_and_predict_mask(frame)
    # 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)

    ret, jpeg = cv2.imencode('.jpg', frame)
    return jpeg.tobytes()
```

App.py =>

```
------------------------------------------------------------------------------------------------------------------------

from flask import Flask, render_template, Response
from mask_detect_video_app import VideoCamera
from flask import send_file

app = Flask(__name__)

@app.route('/')
def index():
    """Video streaming home page."""
    return render_template('index.html')

@app.route('/project')
def project():
    """ Project Page"""
    return render_template('project.html')

@app.route('/about_me')
def about_me():
    """ About Me page"""
    return render_template('about_me.html')

@app.route('/download')
def downloadFile ():
    #For windows you need to use drive name [ex: F:/Example.pdf]
    path = "NIIT_Project.pdf"
    return send_file(path, as_attachment=True)

def gen(camera):
    """Video streaming generator function."""
    while True:
        frame = camera.get_frame()

        yield (b'--frame\r\n'b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')


@app.route('/video_feed')

def video_feed():

    """Video streaming route. Put this in the src attribute of an img tag."""

    return Response(gen(VideoCamera()),mimetype='multipart/x-mixed-replace; boundary=frame')


if __name__ == "__main__":

    app.run(host="0.0.0.0",port="5000",debug=True)
```

Output:



To see all codes related to this app, click here

Future Scope & Advantages

Python has become a formidable language in the data science, artificial intelligence, and machine learning spheres. This is largely due to the language's flexibility and community, but it's also a direct result of the production of many ultra-powerful, high-quality packages and modules. Further considerations should include the situations where data science tasks (analytics, machine learning, artificial intelligence) are carried out either on a local desktop or laptop machine by a data scientist (for example), or where these tasks are performed on servers (usually in the cloud).

This project definitely will open a future path in computer vision and deep learning. Working in the ML and AI industry means dealing with a bunch of data that you need to process in the most convenient and effective way. The low entry barrier allows more data scientists to quickly pick up Python and start using it for AI development without wasting too much effort on learning the language. Python for machine learning development can run on any platform including Windows, MacOS, Linux, Unix, and twenty-one others.

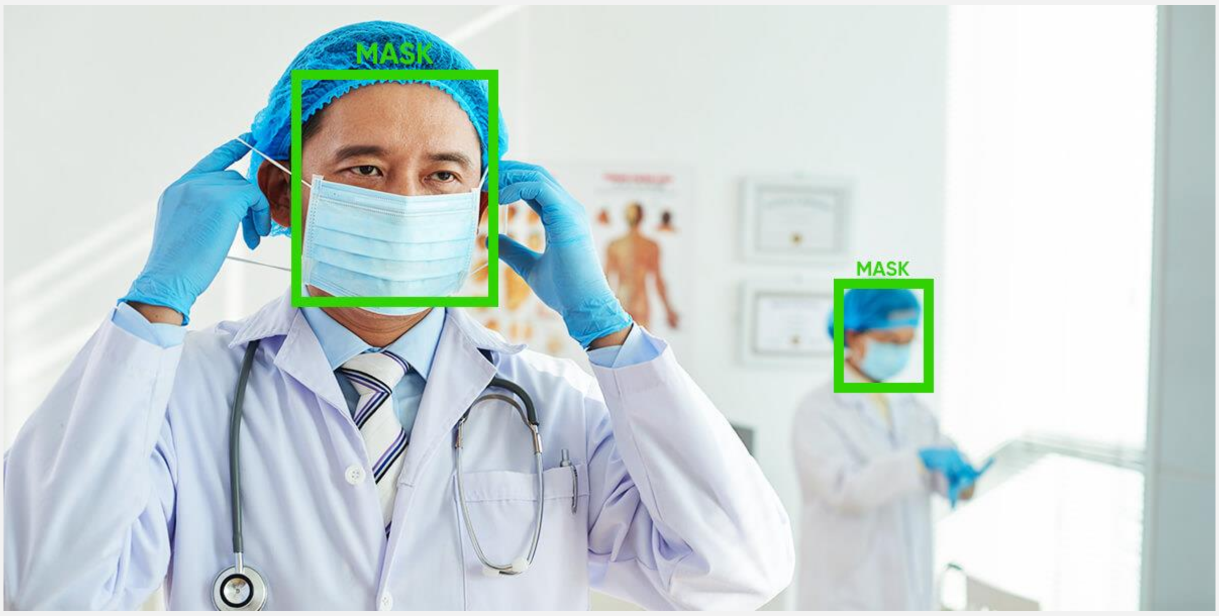
Here are a few use cases where this mask detection technology could beneficial.

Airports:

The Face Mask Detection System could be used at airports to detect travellers without masks. Face data of travellers can be captured in the system at the entrance. If a traveller is found to be without a face mask, their picture is sent to the airport authorities so that they could take quick action.

Hospitals:

Using Face Mask Detector System, Hospitals can monitor if quarantined people required to wear a mask are doing so or not. The same holds good for monitoring staff on duty too.



Offices & Working Spaces:

The Face Mask Detection System can be used at office premises to ascertain if employees are maintaining safety standards at work. It monitors employees without masks and sends them a reminder to wear a mask.

Government:

To limit the spread of coronavirus, the police could deploy the face mask detector on its fleet of surveillance cameras to enforce the compulsory wearing of face masks in public places.

Conclusion

The current study used OpenCV, TensorFlow, Keras and CNN to detect whether people were wearing face masks or not. The models were tested with images and real-time video streams. Even though the accuracy of the model is around 70%, the optimization of the model is a continuous process and we are building a highly accurate solution by tuning the hyperparameters. MobileNetV2 was used to build the mobile version of the same. This specific model could be used as a use case for edge analytics. There are no doubts that AI technologies are the future. Considering the increasing popularity of the trend and the number of people ready to invest in it, the global AI market is going to reach $89.8 billion by 2025. Basically, most scenarios about future AI are hypothetical, but they present us with existential questions. There is a never-ending chase of the Machine and humans, and it will run in unparalleled till one conquers the other. So, we can say that in further future AI also will help us to solve real world problem like Face Mask detection.

Bibliography

The contents have been gathered from the following:

1.Google Search(https://www.google.com/)

2.TensorFlow(https://www.tensorflow.org/tutorials/)

3.Keras(https://keras.io/api/)

4. OpenCV(https://opencv.org/courses/)

5.Youtube Tutorials (https://www.youtube.com/)