REPORT Covid-19 Face Mask Detection



STUDENT NAME:

Mohammad ALShahrour

DOCTOR SUPERVISED:

DR.MASSA ALBAALI

COURES:

NUERAL NETWORK

* Time:

It took 1 week to cleanup to pre-preparing the dataset before we start the algorithms and training

Then it took 1 week to implement the main algorithms (CNN , KERAS , TENSARFLOW, Scikit-Learn,MOBILE NET V2) and.

* Abstract:

COVID-19 pandemic has rapidly affected our day-to-day life disrupting the world trade and movements. Wearing a protective face mask has become a new normal. In the near future, many public service providers will ask the customers to wear masks correctly to avail of their services. Therefore, face mask detection has become a crucial task to help global society. This paper presents a simplified approach to achieve this purpose using some basic Machine Learning packages like (TensorFlow, Keras, ,MOBILE NET V2 and Scikit-Learn. The proposed method detects the face from the image correctly and then identifies if it has a mask on it or not. As a surveillance task performer, it can also detect a face along with a mask in motion. The method attains accuracy up to 95.77% and 94.58% respectively on two different datasets. We explore optimized values of parameters using the Sequential Convolutional Neural Network model to detect the presence of masks correctly without causing over-fitting.

## **Introduction:**

According to the World Health Organization (WHO)’s official Situation Report – 205, coronavirus disease 2019 (COVID-19) has globally infected over 20 million people causing over 0.7million deaths . Individuals with COVID-19 have had a wide scope of symptoms reported – going from mellow manifestations to serious illness. Respiratory problems like shortness of breath or difficulty in breathing is one of them. Elder people having lung disease can possess serious complications from COVID-19 illness as they appear to be at higher risk . Some common human coronaviruses that infect public around the world are 229E, HKU1, OC43, and NL63. Before debilitating individuals, viruses like 2019-nCoV, SARS-CoV, and MERS-CoV infect animals and evolve to human coronaviruses . Persons having respiratory problems can expose anyone (who is in close contact with them) to infective beads. Surroundings of a tainted individual can cause contact transmission as droplets carrying virus may withal arrive on his adjacent surfaces .

To curb certain respiratory viral ailments, including COVID-19, wearing a clinical mask is very necessary. The public should be aware of whether to put on the mask for source control or aversion of COVID-19. Potential points of interest of the utilization of masks lie in reducing vulnerability of risk from a noxious individual during the "pre-symptomatic" period and stigmatization of discrete persons putting on masks to restraint the spread of virus. WHO stresses on prioritizing medical masks and respirators for health care assistants. Therefore, face mask detection has become a crucial task in present global society.

Face mask detection involves in detecting the location of the face and then determining whether it has a mask on it or not. The issue is proximately cognate to general object detection to detect the classes of objects. Face identification categorically deals with distinguishing a specific group of entities i.e. Face. It has numerous applications, such as autonomous driving, education, surveillance, and so on . This paper presents a simplified approach to serve the above purpose using the basic Machine Learning (ML) packages such as TensorFlow, Keras, MOBILE NET V2 and Scikit-Learn.

* **Related Work:**

In face detection method, a face is detected from an image that has several attributes in it.According to D. Meena and R. Sharan, "An approach to face detection and recognition", 2016 International Conference on Recent Advances and Innovations in Engineering (ICRAIE), pp. 1-6, 2016., research into face detection requires expression recognition, face tracking, and pose estimation. Given a solitary image, the challenge is to identify the face from the picture. Face detection is a difficult errand because the faces change in size, shape, color, etc and they are not immutable. It becomes a laborious job for opaque image impeded by some other thing not confronting camera,AUTHORIS IN S. Ge, J. Li, Q. Ye and Z. Luo, "Detecting Masked Faces in the Wild with LLE-CNNs", 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 426-434, 201 and so forth. think occlusive face detection comes with two major challenges: 1) unavailability of sizably voluminous datasets containing both masked and unmasked faces, and 2) exclusion of facial expression in the covered area. Utilizing the locally linear embedding (LLE) algorithm and the dictionaries trained on an immensely colossal pool of masked faces, synthesized mundane faces, several mislaid expressions can be recuperated and the ascendancy of facial cues can be mitigated to great extent. According to the work reported in S. Ghosh, N. Das and M. Nasipuri, "Reshaping inputs for convolutional neural network: Some common and uncommon methods", Pattern Recognition, vol. 93, pp. 79-94, 2019. , convolutional neural network (CNNs) in computer vision comes with a strict constraint regarding the size of the input image. The prevalent practice reconfigures the images before fitting them into the network to surmount the inhibition.

Here the main challenge of the task is to detect the face from the image correctly and then identify if it has a mask on it or not. In order to perform surveillance tasks, the proposed method should also detect a face along with a mask in motion.

## **Dataset:**

Two datasets have been used for experimenting the current method. Dataset 1 [16] consists of 1376 images in which 690 images with people wearing face masks and the rest 686 images with people who do not wear face masks. mostly contains front face pose with single face in the frame and with same type of mask having white color only.



EXAMPLE OF DATA SET WITH OUT MASK



EXAMPLE OF DATA SET WITH MASK

## **Incorporated Packages:**

### A. TensorFlow

TensorFlow, an interface for expressing machine learning algorithms, is utilized for implementing ML systems into fabrication over a bunch of areas of computer science, including sentiment analysis, voice recognition, geographic information extraction, computer vision, text summarization, information retrieval, computational drug discovery and flaw detection to pursue research . In the proposed model, the whole Sequential CNN architecture (consists of several layers) uses TensorFlow at backend. It is also used to reshape the data (image) in the data processing.

**TensorFlow White Papers", TensorFlow, 2020, [online] Available: https://www.tensorflow.org/about/bib.**

### B. Keras

Keras gives fundamental reflections and building units for creation and transportation of ML arrangements with high iteration velocity. It takes full advantage of the scalability and cross-platform capabilities of TensorFlow. The core data structures of Keras are layers and models . All the layers used in the CNN model are implemented using Keras. Along with the conversion of the class vector to the binary class matrix in data processing, it helps to compile the overall model.

**Keras documentation: About Keras", 2020, [online] Available: Keras.io.**

### C. MOBILE NET V2

**MobileNetV2 is a general architecture and can be used for multiple use cases. Depending on the use case, it can use different input layer size and different width factors. This allows different width models to reduce the number of multiply-adds and thereby reduce inference cost on mobile devices**.

* **DENS LAYER DESCRIPTION:**

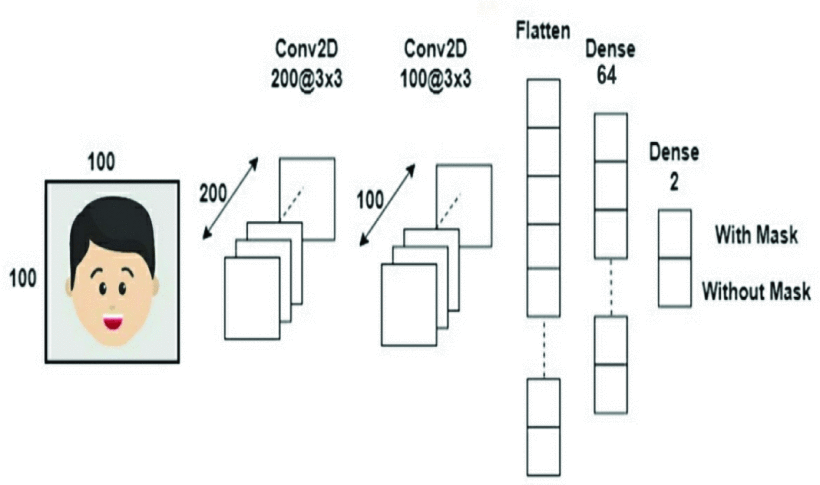
**IN FIGURE IMAGE IS 100\*100 BUT WE USE 224\*224**

**AND CONV 2D 7\*7**

**IN OUT PUT WE USE DENSE 2 BECAUSE THE OUTPUT WILL BE 2 (WITH MASK OR WITH OUT MASK)**

Flatten:Suppose you’re using a Convolutional Neural Network whose initial layers are Convolution and Pooling layers. They layers have multidimensional tensors as their outputs. If you wanted to use a Dense(a fully connected layer) after your convolution layers, you would need to ‘unstack’ all this multidimensional tensor into a very long 1D tensor. You can achieve this using Flatten.

**Dense layer** is the regular deeply connected neural network layer. It is most common and frequently used layer. Dense layer does the below operation on the input and return the output.



* **CODE AND DESCRIPTION:**

**Doctor we description all code with comment in code**

-#here is library import drive to go to the drive

from google.colab import drive

drive.mount('/content/MyDrive')

#here is path of project that uploded to drive and read and extract all file in zip

project\_path = '/content/MyDrive/My Drive/Colab Notebooks/Face-Mask-Detection-master/'

training\_zip\_path = project\_path + 'face-mask-detector.zip'

from zipfile import ZipFile

with ZipFile(training\_zip\_path, 'r') as z:

  z.extractall()

print("Training zip extraction done!")

%cd /content/face-mask-detector

#here is all file in 'face-mask-detector.zip'

!ls

# USAGE

# python train\_mask\_detector.py --dataset dataset

# import the necessary packages

#Keras ImageDataGenerator class provides a quick and easy way to augment your images. It provides a host of different augmentation techniques like standardization, rotation, shifts, flips, brightness change, and many more

from tensorflow.keras.preprocessing.image import ImageDataGenerator

#This allows different width models to reduce the number of multiply-adds and thereby reduce inference cost on mobile devices.

from tensorflow.keras.applications import MobileNetV2

#Average pooling operation for spatial data.

from tensorflow.keras.layers import AveragePooling2D

#dropout helps prevent overfitting

from tensorflow.keras.layers import Dropout

#convert 2d to 1d

from tensorflow.keras.layers import Flatten

#we use dense to train the network Dense layer is the regular deeply connected neural network layer. It is most common and frequently used  Dense layer does the below operation on the input and return the output.

from tensorflow.keras.layers import Dense

#use input in keras

from tensorflow.keras.layers import Input

#To use model in keras

from tensorflow.keras.models import Model

#Optimizer to enhancing result with learning rate(1e-4)

from tensorflow.keras.optimizers import Adam

#to adequate your image to the format the model requires.

from tensorflow.keras.applications.mobilenet\_v2 import preprocess\_input

#function for converting a loaded image in PIL format into a NumPy array for use with deep learning models.

from tensorflow.keras.preprocessing.image import img\_to\_array

#Loads an image into PIL format.

from tensorflow.keras.preprocessing.image import load\_img

#To categorical output to classification

from tensorflow.keras.utils import to\_categorical

from sklearn.preprocessing import LabelBinarizer

#use train and test to train and test project

from sklearn.model\_selection import train\_test\_split

#Build a text report showing the main classification metrics

from sklearn.metrics import classification\_report

from imutils import paths

#plt use to plot

import matplotlib.pyplot as plt

# NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data

import numpy as np

import argparse

import os

# initialize the initial learning rate, number of epochs to train for,

# and batch size

INIT\_LR = 1e-4

EPOCHS = 20

BS = 32

# 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...")

imagePaths = list(paths.list\_images('/content/face-mask-detector/dataset'))

data = []

labels = []

# loop over the image paths

for imagePath in imagePaths:

  # extract the class label from the filename

  label = imagePath.split(os.path.sep)[-2]

  # load the input image (224x224) and preprocess it and convert image to array

  image = load\_img(imagePath, target\_size=(224, 224))

  image = img\_to\_array(image)

  image = preprocess\_input(image)

# update the data and labels lists, respectively that take image and put them in list of data and labels

  data.append(image)

  labels.append(label)

# convert the data and labels to NumPy arrays

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

labels = np.array(labels)

# perform one-hot encoding on the labels

lb = LabelBinarizer()

#grab each labels and trasnform

labels = lb.fit\_transform(labels)

#Converts a class vector(integers) to binary class matrix.

labels = to\_categorical(labels)

# partition the data into training and testing splits using 75% of

# the data for training and the remaining 25% 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")

# load the MobileNetV2 network, ensuring the head FC layer sets are

# left off

#(224,244,3)is 3 inputs height, width and channels(RGB)

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

# the base model

#take input image with(224X224X3)

headModel = baseModel.output

# its spatial dimensions (height and width)it transfer from 3d to 2d

headModel = AveragePooling2D(pool\_size=(7, 7))(headModel)

#transfer from 2d to 1d

headModel = Flatten(name="flatten")(headModel)

#we take 1d from flatten layer use dense layer to train with Convolutional neural network and use relu activation function with 128 dense

headModel = Dense(128, activation="relu")(headModel)

#use dropout with 0.5 which helps prevent overfitting in our project

headModel = Dropout(0.5)(headModel)

#the output of our project we use 2 dense because we have 2 output(with mask,without mask) and we use activation softmax beacuse we have 2 classifacation output in our project

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("[INFO] compiling model...")

#we use adam optimizer with learning rate (1e-4)

opt = Adam(lr=INIT\_LR, decay=INIT\_LR / EPOCHS)

#we Compile of model with optimizer to enhance result we use loss="binary\_crossentropy to use binary classification(with mask,withoutmask)

#and opt is adam optimizer we define in up statment we use accuracy metrics

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\_))

# 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")

# 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")

# USAGE

# python detect\_mask\_image.py --image examples/example\_01.png

# 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 argparse

import cv2

import os

from google.colab.patches import cv2\_imshow

# load the input image from disk, clone it, and grab the image spatial

# dimensions

image = cv2.imread('/content/face-mask-detector/examples/example\_02.png')

orig = image.copy()

(h, w) = image.shape[:2]

# construct a blob from the image

blob = cv2.dnn.blobFromImage(image, 1.0, (300, 300),

  (104.0, 177.0, 123.0))

#grap face detection model

net=cv2.dnn.readNet('/content/face-mask-detector/face\_detector/deploy.prototxt','/content/face-mask-detector/face\_detector/res10\_300x300\_ssd\_iter\_140000.caffemodel')

# pass the blob through the network and obtain the face detections

print("[INFO] computing face detections...")

net.setInput(blob)

detections = net.forward()

# 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 = 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(image, label, (startX, startY - 10),

      cv2.FONT\_HERSHEY\_SIMPLEX, 0.45, color, 2)

    cv2.rectangle(image, (startX, startY), (endX, endY), color, 2)

# show the output image example of our project

cv2\_imshow(image)

cv2.waitKey(0)

# USAGE

# python 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

from imutils.video import VideoStream

import numpy as np

import argparse

import imutils

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, (300, 300),

    (104.0, 177.0, 123.0))

  # pass the blob through the network and obtain the face detections

  faceNet.setInput(blob)

  detections = 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 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)

faceNet=cv2.dnn.readNet('/content/face-mask-detector/face\_detector/deploy.prototxt','/content/face-mask-detector/face\_detector/res10\_300x300\_ssd\_iter\_140000.caffemodel')

import base64

import html

import io

import time

from IPython.display import display, Javascript

from google.colab.output import eval\_js

import numpy as np

from PIL import Image

import cv2

def start\_input():

  js = Javascript('''

    var video;

    var div = null;

    var stream;

    var captureCanvas;

    var imgElement;

    var labelElement;

    var pendingResolve = null;

    var shutdown = false;

    function removeDom() {

       stream.getVideoTracks()[0].stop();

       video.remove();

       div.remove();

       video = null;

       div = null;

       stream = null;

       imgElement = null;

       captureCanvas = null;

       labelElement = null;

    }

    function onAnimationFrame() {

      if (!shutdown) {

        window.requestAnimationFrame(onAnimationFrame);

      }

      if (pendingResolve) {

        var result = "";

        if (!shutdown) {

          captureCanvas.getContext('2d').drawImage(video, 0, 0, 512, 512);

          result = captureCanvas.toDataURL('image/jpeg', 0.8)

        }

        var lp = pendingResolve;

        pendingResolve = null;

        lp(result);

      }

    }

    async function createDom() {

      if (div !== null) {

        return stream;

      }

      div = document.createElement('div');

      div.style.border = '2px solid black';

      div.style.padding = '3px';

      div.style.width = '100%';

      div.style.maxWidth = '600px';

      document.body.appendChild(div);

      const modelOut = document.createElement('div');

      modelOut.innerHTML = "<span>Status:</span>";

      labelElement = document.createElement('span');

      labelElement.innerText = 'No data';

      labelElement.style.fontWeight = 'bold';

      modelOut.appendChild(labelElement);

      div.appendChild(modelOut);

      video = document.createElement('video');

      video.style.display = 'block';

      video.width = div.clientWidth - 6;

      video.setAttribute('playsinline', '');

      video.onclick = () => { shutdown = true; };

      stream = await navigator.mediaDevices.getUserMedia(

          {video: { facingMode: "environment"}});

      div.appendChild(video);

      imgElement = document.createElement('img');

      imgElement.style.position = 'absolute';

      imgElement.style.zIndex = 1;

      imgElement.onclick = () => { shutdown = true; };

      div.appendChild(imgElement);

      const instruction = document.createElement('div');

      instruction.innerHTML =

          '<span style="color: red; font-weight: bold;">' +

          'When finished, click here or on the video to stop this demo</span>';

      div.appendChild(instruction);

      instruction.onclick = () => { shutdown = true; };

      video.srcObject = stream;

      await video.play();

      captureCanvas = document.createElement('canvas');

      captureCanvas.width = 512; //video.videoWidth;

      captureCanvas.height = 512; //video.videoHeight;

      window.requestAnimationFrame(onAnimationFrame);

      return stream;

    }

    async function takePhoto(label, imgData) {

      if (shutdown) {

        removeDom();

        shutdown = false;

        return '';

      }

      var preCreate = Date.now();

      stream = await createDom();

      var preShow = Date.now();

      if (label != "") {

        labelElement.innerHTML = label;

      }

      if (imgData != "") {

        var videoRect = video.getClientRects()[0];

        imgElement.style.top = videoRect.top + "px";

        imgElement.style.left = videoRect.left + "px";

        imgElement.style.width = videoRect.width + "px";

        imgElement.style.height = videoRect.height + "px";

        imgElement.src = imgData;

      }

      var preCapture = Date.now();

      var result = await new Promise(function(resolve, reject) {

        pendingResolve = resolve;

      });

      shutdown = false;

      return {'create': preShow - preCreate,

              'show': preCapture - preShow,

              'capture': Date.now() - preCapture,

              'img': result};

    }

    ''')

  display(js)

def take\_photo(label, img\_data):

  data = eval\_js('takePhoto("{}", "{}")'.format(label, img\_data))

  return data

def js\_reply\_to\_image(js\_reply):

    """

    input:

          js\_reply: JavaScript object, contain image from webcam

    output:

          image\_array: image array RGB size 512 x 512 from webcam

    """

    jpeg\_bytes = base64.b64decode(js\_reply['img'].split(',')[1])

    image\_PIL = Image.open(io.BytesIO(jpeg\_bytes))

    image\_array = np.array(image\_PIL)

    return image\_array

import imutils

start\_input()

label\_html = 'Capturing...'

img\_data = ''

count = 0

from google.colab.patches import cv2\_imshow

while True:

  js\_reply = take\_photo(label\_html, img\_data)

  if not js\_reply:

    break

  image = js\_reply\_to\_image(js\_reply)

  # grab the frame from the threaded video stream and resize it

  # to have a maximum width of 400 pixels

  frame = image

  v=True

  if v == True:

    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, model)

    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

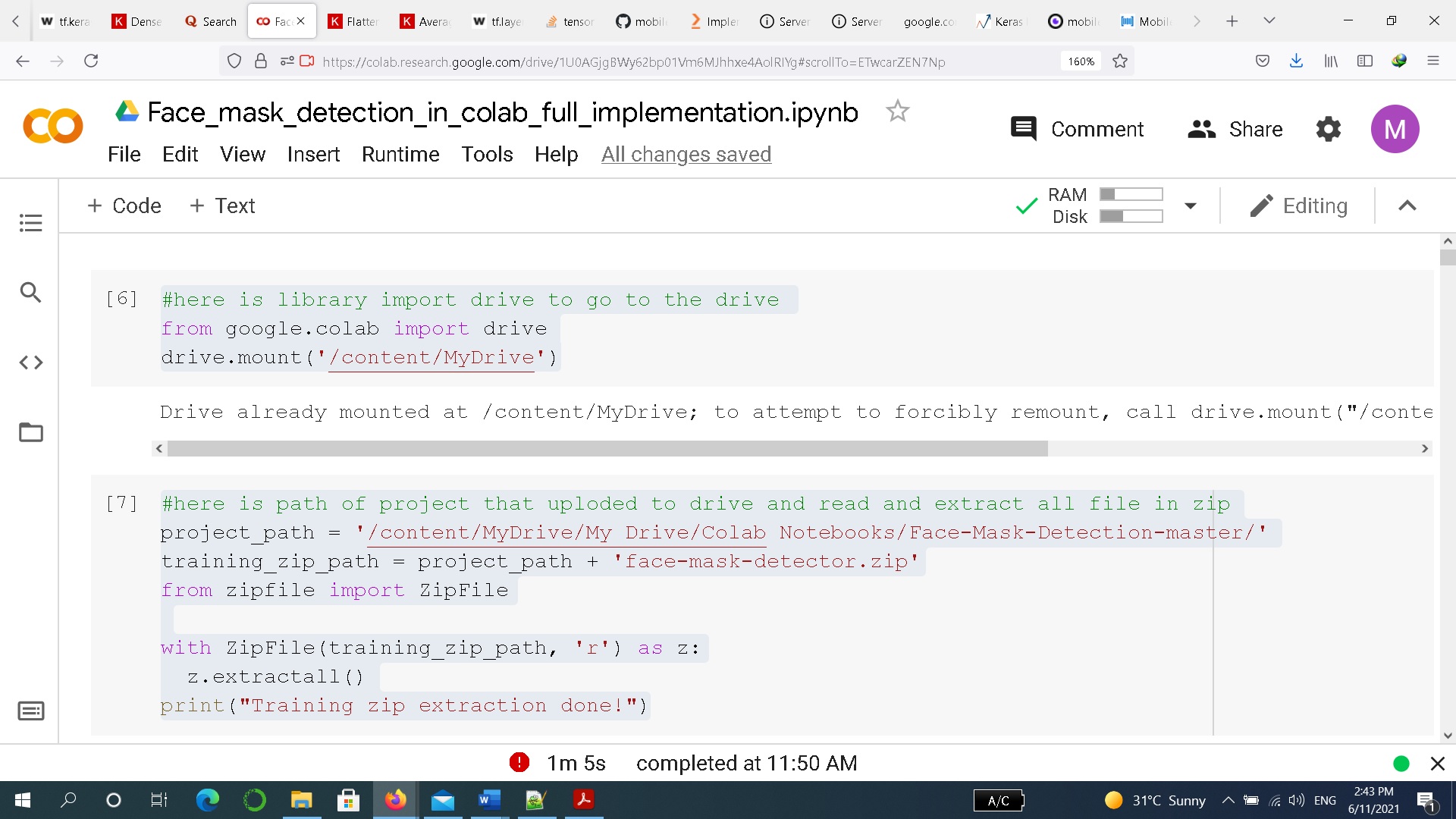
    # frame

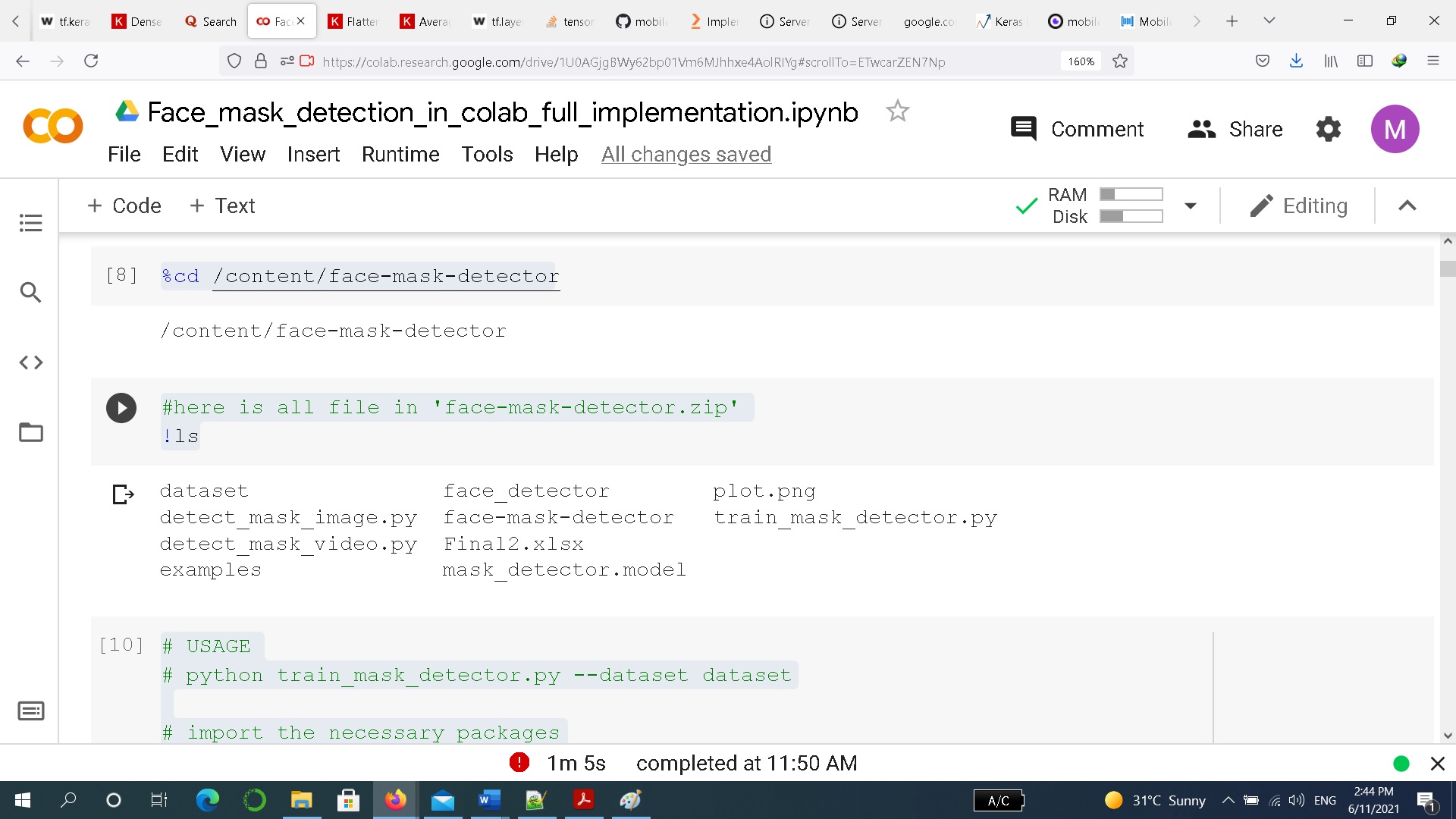
      frame=cv2.putText(frame, label, (startX, startY - 10),cv2.FONT\_HERSHEY\_SIMPLEX, 0.45, color, 2)

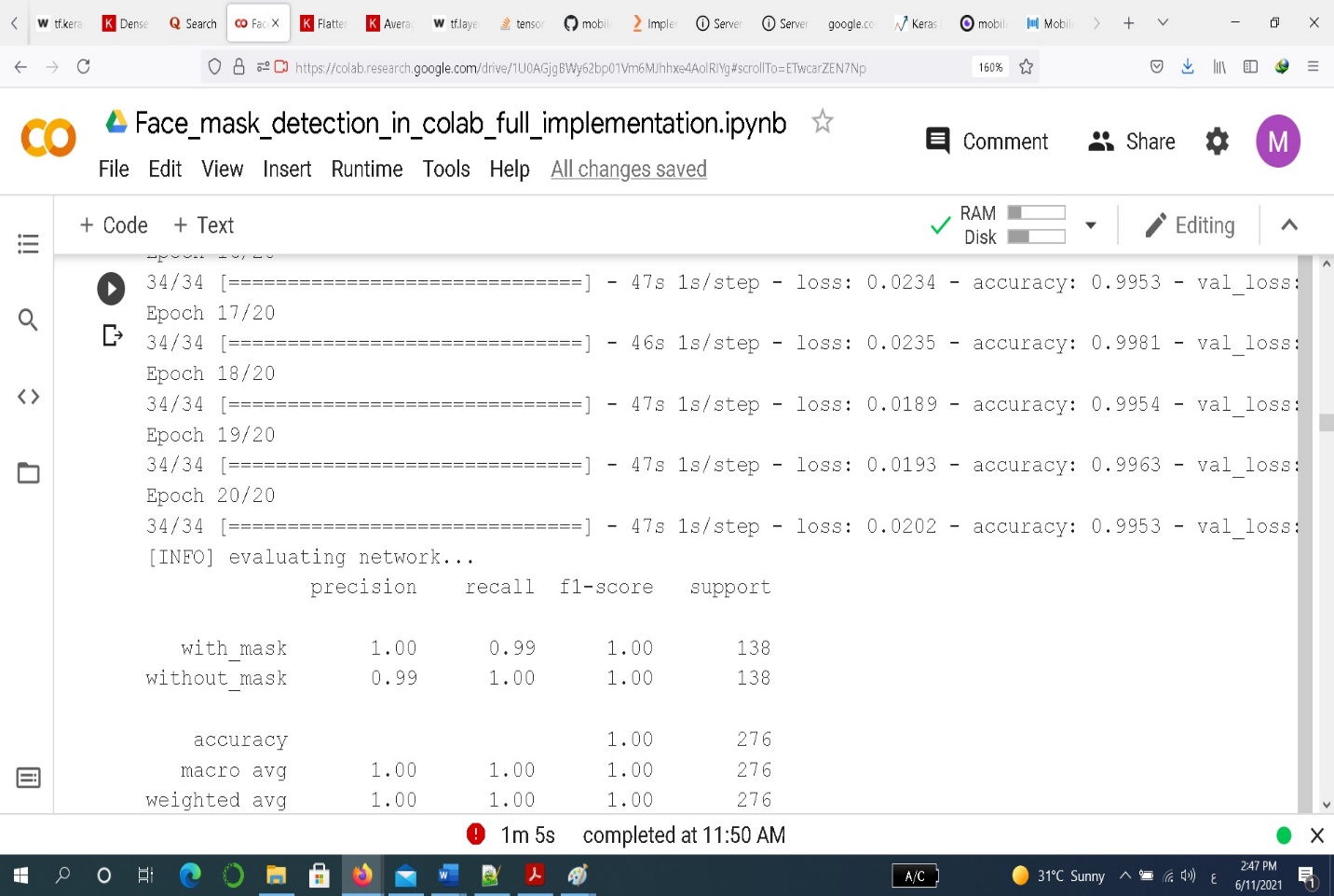
      frame=cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)

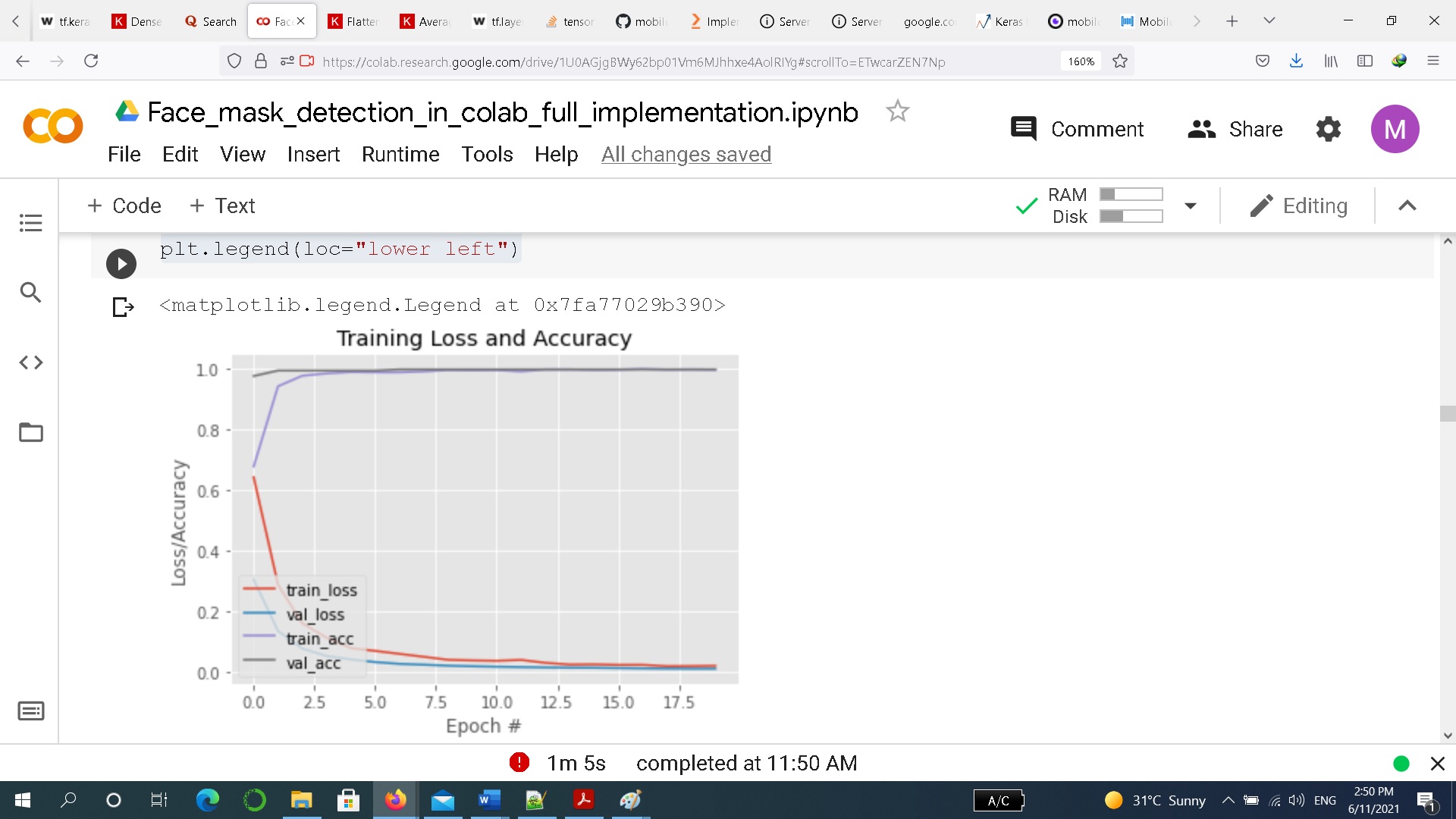
# show the output frame

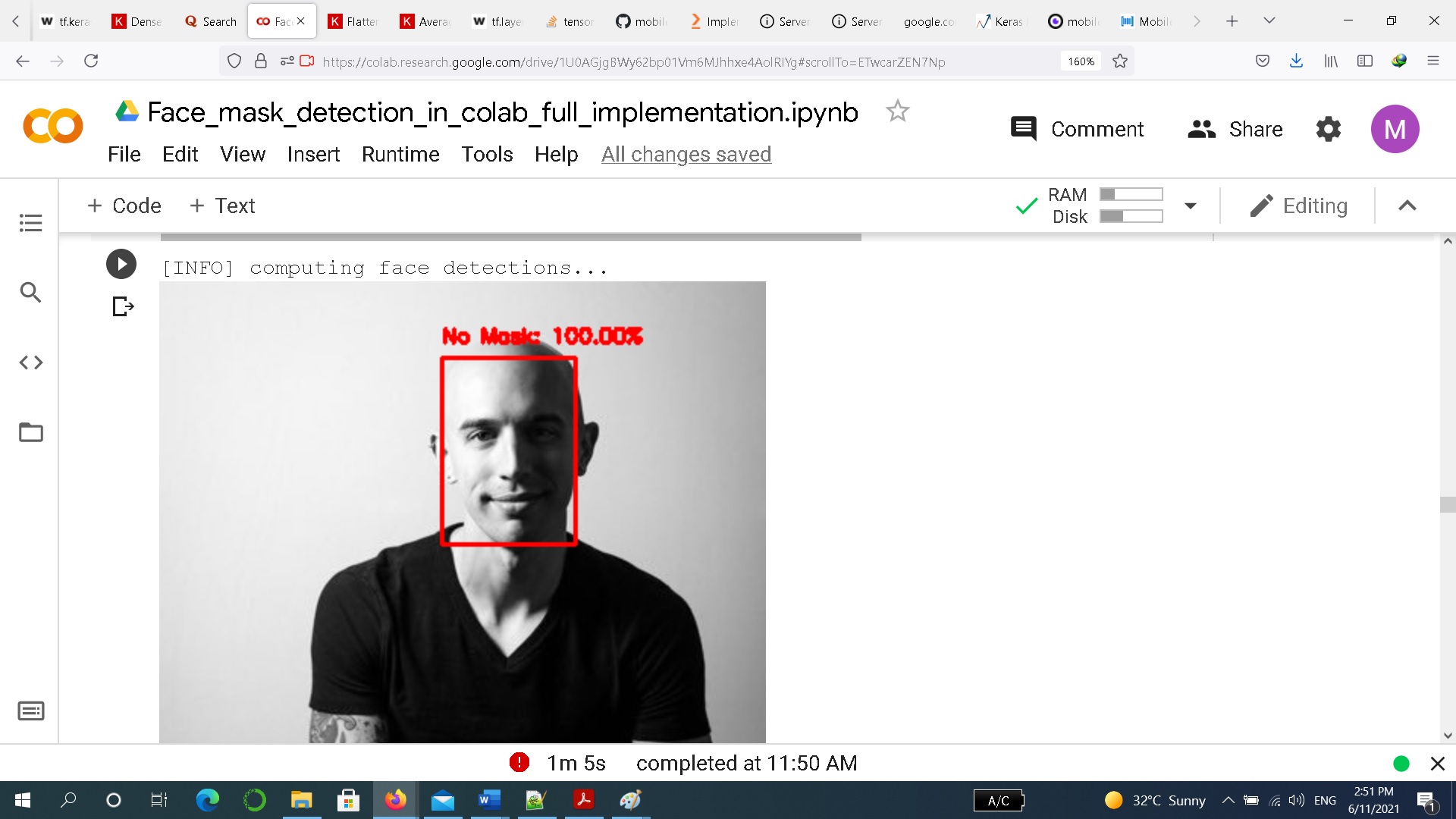
cv2\_imshow(frame)

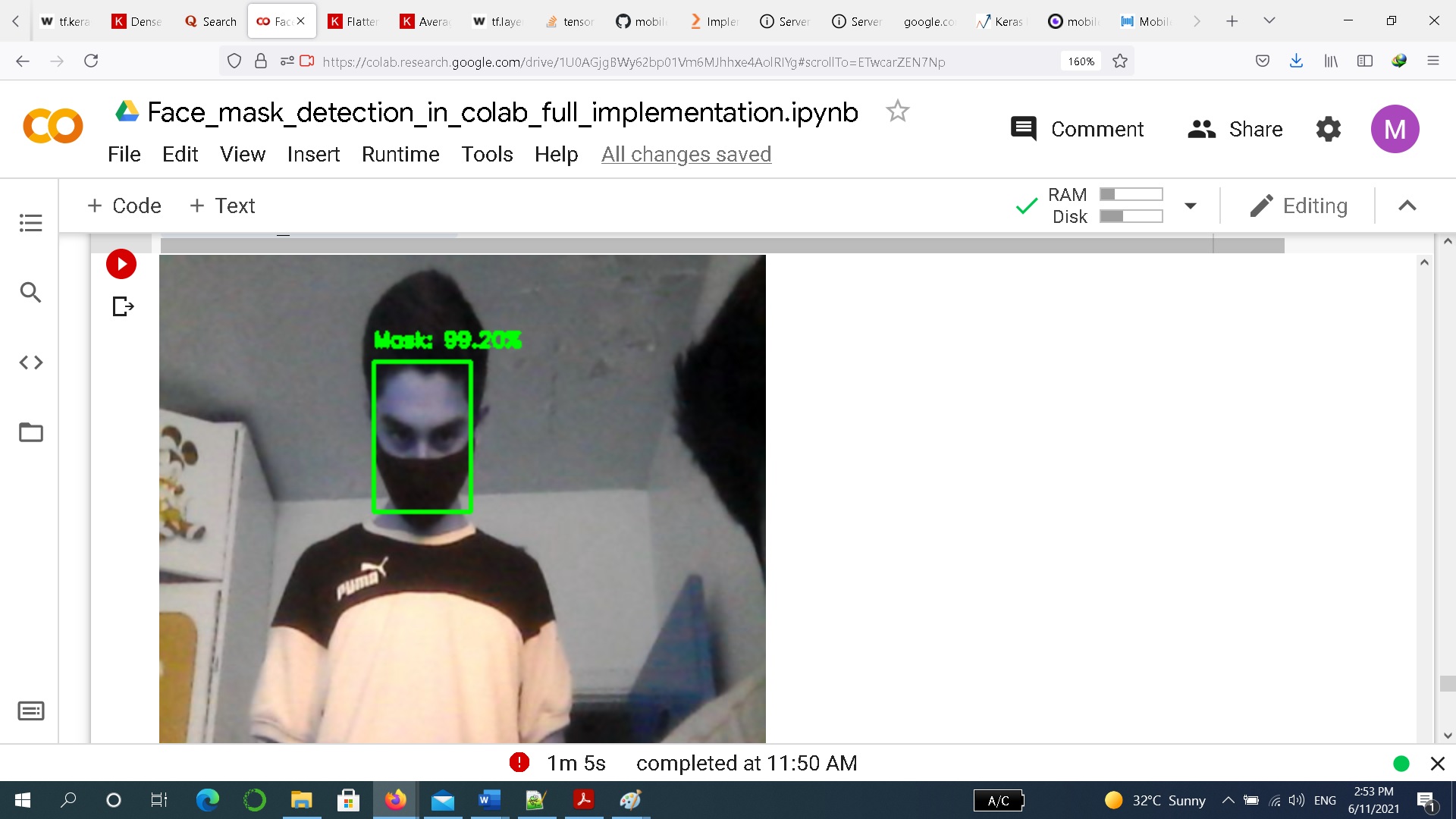
* **screenshot of code:**
* **screenshot of code:**

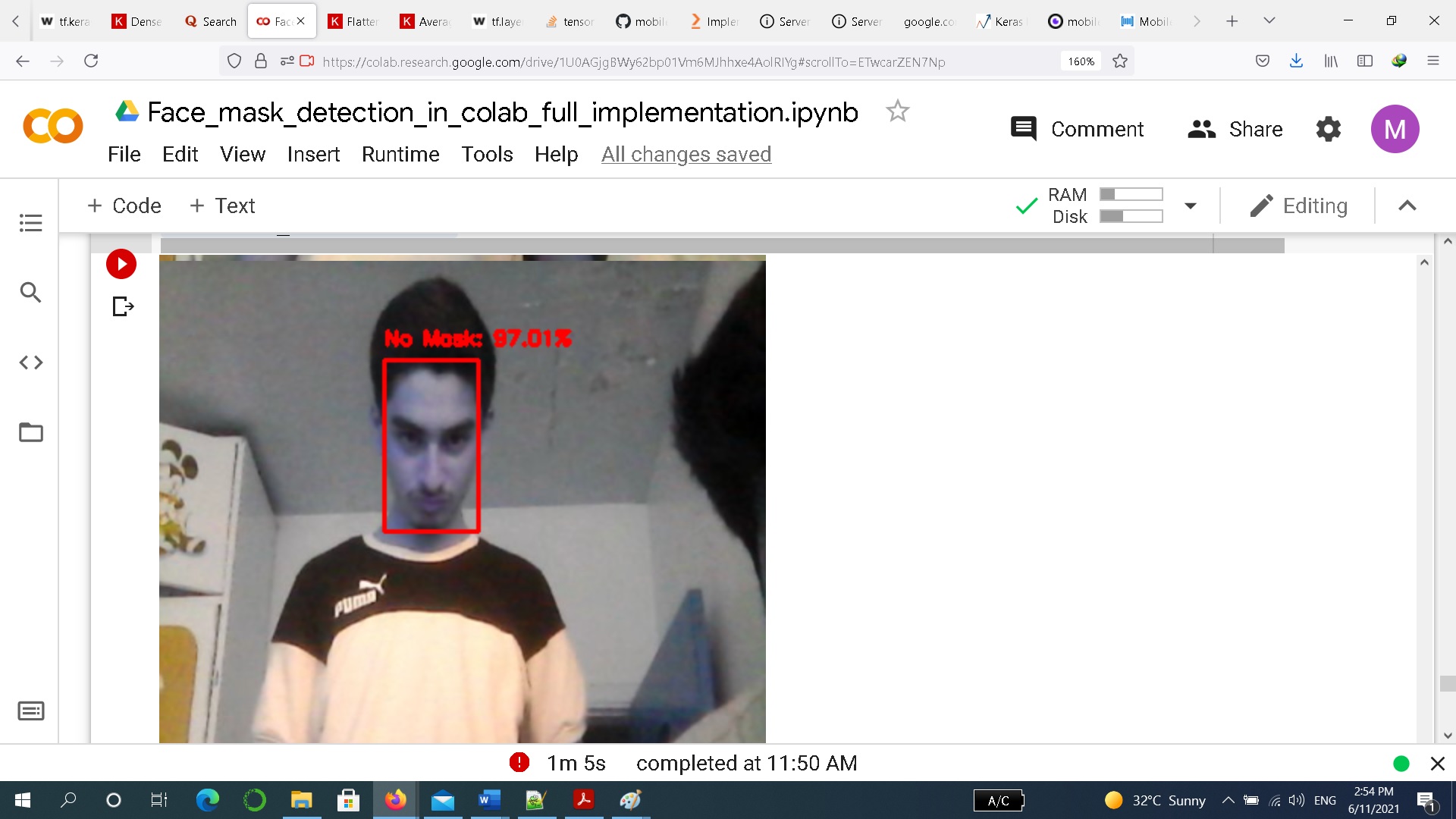
****

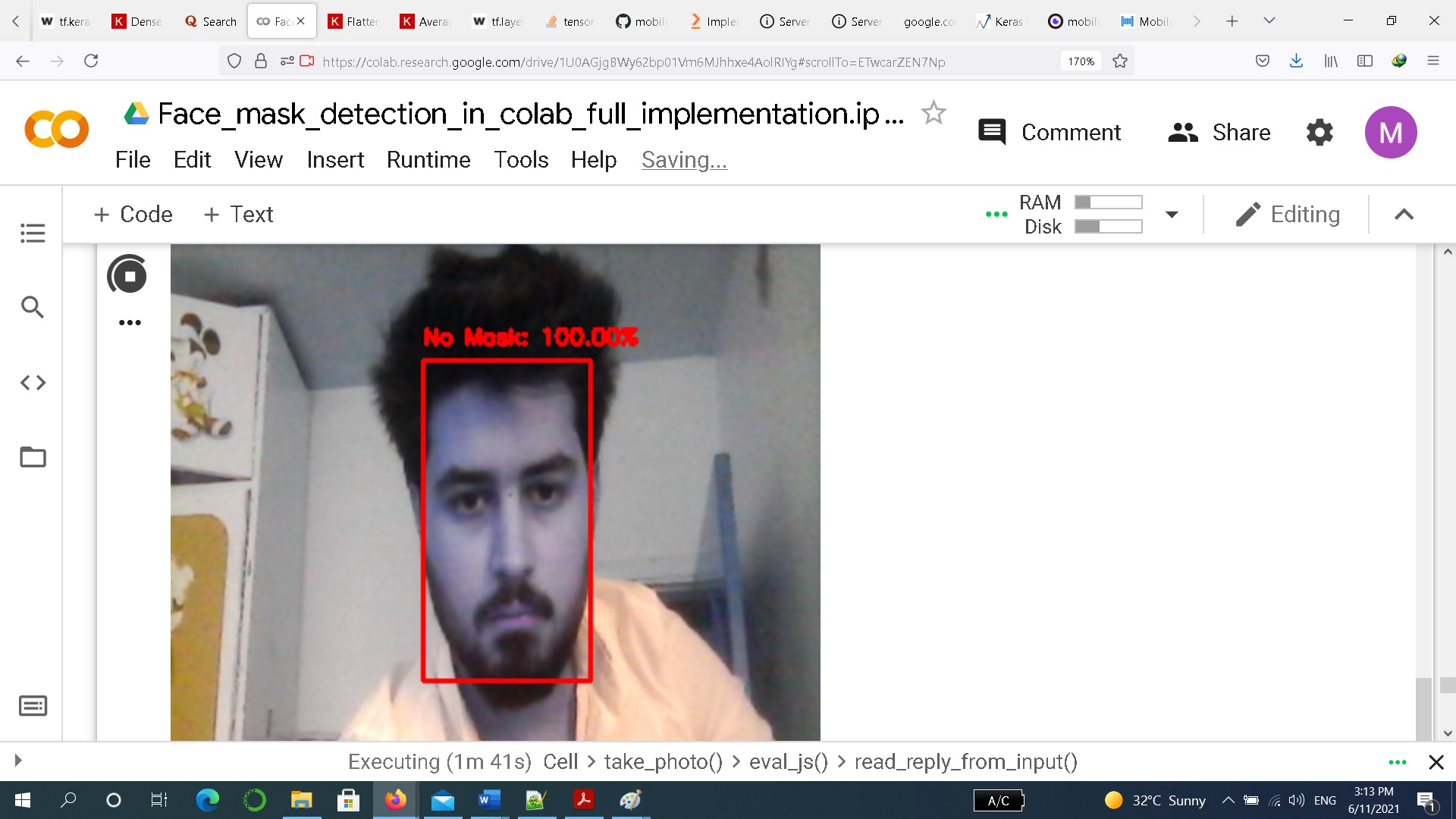
****

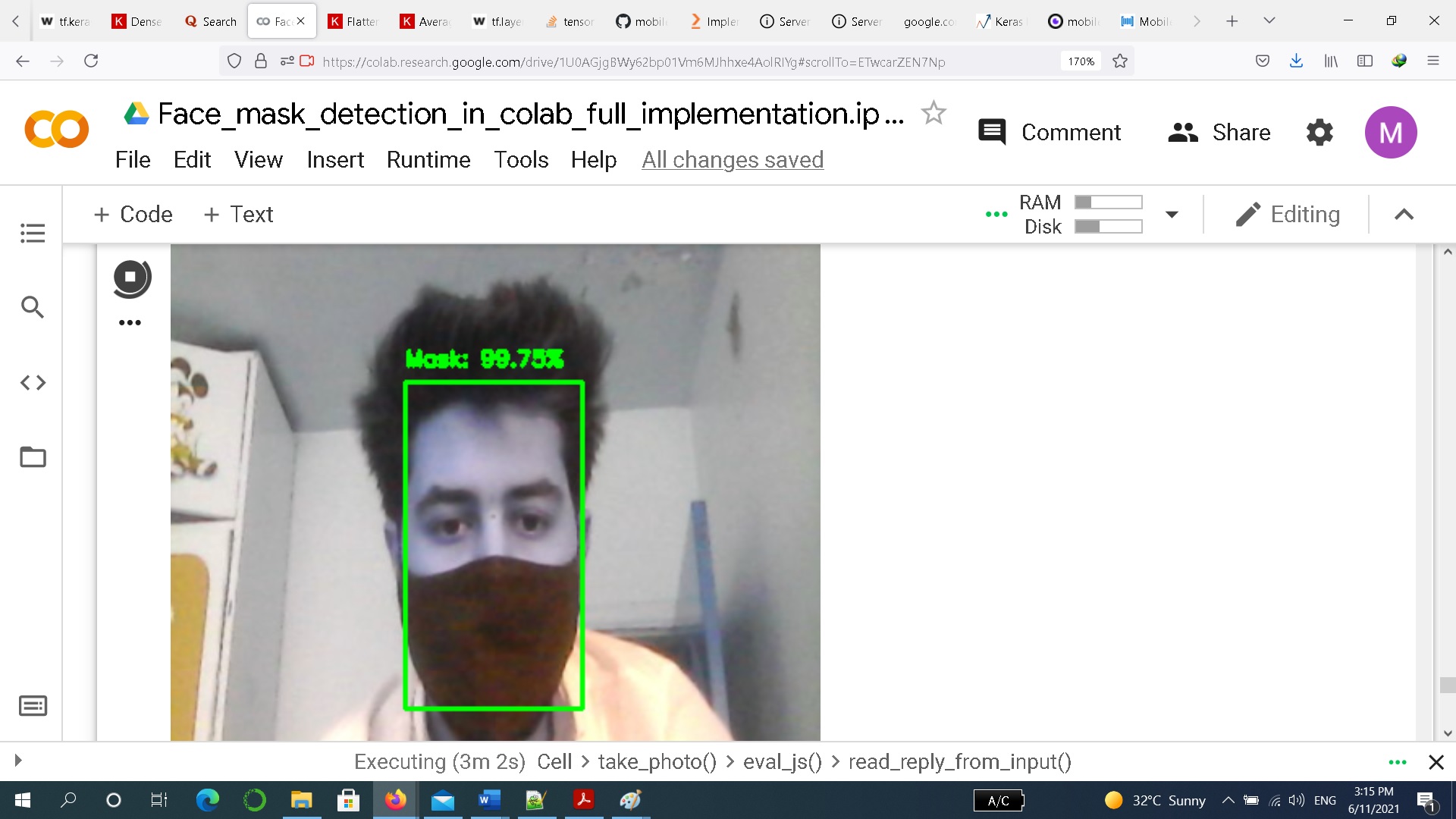
****

****

****

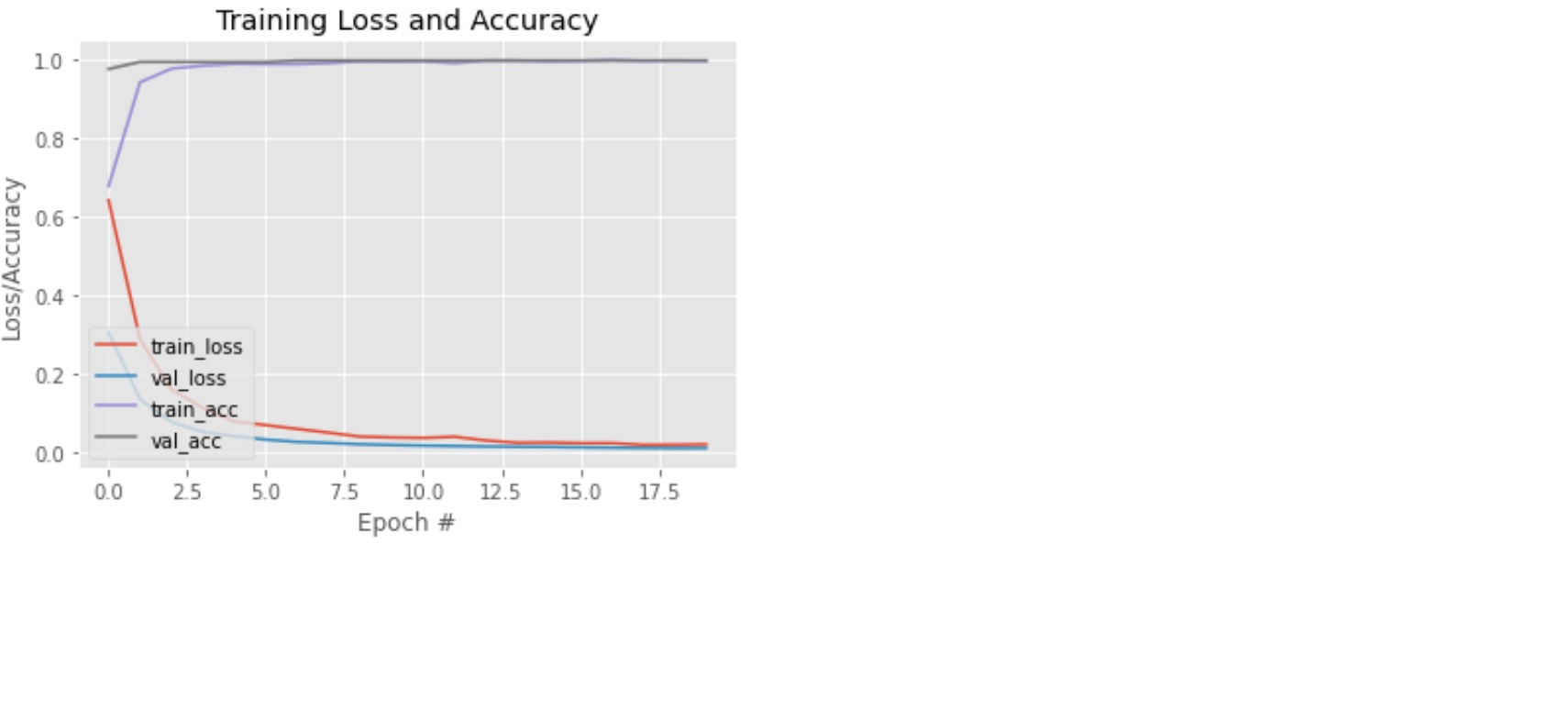
****

****

****

## **Result And Analysis:**

The model is trained, validated and tested upon two datasets



* **Environment**:

We used anaconda for implementing the pre-processing and data science section and implementing non-time-expensive algorithms then when we wanted to train the time-expensive algorithms we used Google Colab platform, it really helped us and saved a much time and give us cpu and gpu speed to train than local training.

## **Conclusions**

In this paper, we briefly explained the motivation of the work at first. Then, we illustrated the learning and performance task of the model. Using basic ML tools and simplified techniques the method has achieved reasonably high accuracy. It can be used for a variety of applications. Wearing a mask may be obligatory in the near future, considering the Covid-19 crisis. Many public service providers will ask the customers to wear masks correctly to avail of their services. The deployed model will contribute immensely to the public health care system. In future it can be extended to detect if a person is wearing the mask properly or not. The model can be further improved to detect if the mask is virus prone or not i.e. the type of the mask is surgical, N95 or not we use 20 epochs to train because no electricity.

**Submission Students:**

**MOHAMMAD ALSHAHROR**