LAMPIRAN-1 Sketch Program

```
# 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
import picamera

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 >args["confidence"]:
          # 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)

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-f", "--face", type=str,
   default="face_detector",
   help="path to face detector model directory")
ap.add_argument("-m", "--model", type=str,
   default="mask_detector.model",
   help="path to trained face mask detector model")
ap.add_argument("-c", "--confidence", type=float,
default=0.5,
   help="minimum probability to filter weak detections")
args = vars(ap.parse_args())
```

```
# load our serialized face detector model from disk
print("[INFO] loading face detector model...")
prototxtPath = os.path.sep.join([args["face"],
"deploy.prototxt"])
weightsPath = os.path.sep.join([args["face"],
   "res10_300x300_ssd_iter_140000.caffemodel"])
faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)

# load the face mask detector model from disk
print("[INFO] loading face mask detector model...")
maskNet = load_model(args["model"])

# initialize the video stream and allow the camera sensor
to warm up
print("[INFO] starting video stream...")
vs = VideoStream(usePiCamera=True).start()
time.sleep(2.0)

# loop over the frames from the video stream
while True:
   # grab the frame from the threaded video stream and
resize it
   # to have a maximum width of 400 pixels
   frame = vs.read()
   frame = imutils.resize(frame, width=600)
   frame = imutils.rotate(frame, 180)

   # 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)
     if label != "Mask":
           os.system('omxplayer alarm.mp3 &')
           camera=picamera.PiCamera()
           try:
                 camera.start_preview
                 time.sleep(2)
                 camera.capture('cam.jpg')
```

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

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

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

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

# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()
```

```
# USAGE
# python detect_mask_video.py

# import library
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
import picamera
import Tkinter as tk
from PIL import Image, ImageTk
import os.path, time

#Set up GUI
window = tk.Tk()  #Makes main window
window.wm_title("Sistem pendeteksi masker")
window.config(background="#000000")
window.attributes("-fullscreen",True)

#deklarasi komponen gui
lmain = tk.Label(window)
lmain.place(x=10, y=150)

label2=tk.Label(window)
label2.place(x=700, y=150)
```

```
label3 = tk.Label(window, fg="#ffffff", bg="#f00000", font="15")
label3.place(x=850,y=650)


label4 = tk.Label(window, text="Sistem pendeteksi penggunaaan
masker", fg="#ffffff", bg="#000000", font=(None,20))
label4.place(x=400,y=60)


label5 = tk.Label(window, text="LIVE FEED VIDEO", fg="#ffffff",
bg="#f00000", font="15")
label5.place(x=295,y=650)


def detect_and_predict_mask(frame, faceNet, maskNet):
      # mengambil dimensi dan membentuk ke format BLOB
      # dari ini
      (h, w) = frame.shape[:2]
      blob = cv2.dnn.blobFromImage(frame, 1.0, (300, 300),
            (104.0, 177.0, 123.0))


      # melewatkan gambar ke jaringan model yang sudah di train
      faceNet.setInput(blob)
      detections = faceNet.forward()


      # inisialisasi lokasi wajah
      # dan membentuk ke jaringan pembacaan wajah
      faces = []
      locs = []
      preds = []


      # loop di sekitar lokasi wajah
      for i in range(0, detections.shape[2]):
            # extract deteksi masker
            confidence = detections[0, 0, i, 2]
```

```
            # filter network wajah dan membandingkan antara
memakai masker atau tidak
            if confidence > args["confidence"]:
                  # menghitung koordinat lokasi masker
                  box = detections[0, 0, i, 3:7] * np.array([w, h,
w, h])
                  (startX, startY, endX, endY) = box.astype("int")


                  # memastikan lingkaran lokasi di tempat yang
tepat
                  (startX, startY) = (max(0, startX), max(0,
startY))
                  (endX, endY) = (min(w - 1, endX), min(h - 1,
endY))


                  # extract face ROI, mengubah BGR ke RGB channel
                  # resize dari 224x224, dan mem perprocess
                  face = frame[startY:endY, startX:endX]
```