Code

# import the necessary packages

import tkinter

import PIL.Image, PIL.ImageTk

from fps import FPS

import numpy as np

import time

import cv2

from args import args

from classes import CLASSES

COLORS = np.random.uniform (0, 255, size= (len (CLASSES), 3))

class App:

def \_\_init\_\_ (self, window, window\_title, video\_source=0):

self.window = window

self.window.resizable(0, 0)

self.window.title(window\_title)

self.video\_source = video\_source

# open video source (by default this will try to open the computer webcam)

self.vid = MyVideoCapture (self.video\_source)

# Create a canvas that can fit the above video source size

self.canvas = tkinter.Canvas (window, width=self.vid.width, height=self.vid.height)

self.canvas.pack ()

# Button that lets the user take a snapshot

self.btn\_snapshot = tkinter.Button (window, text="Snapshot", width=50, command=self.snapshot)

self.btn\_snapshot.pack (anchor=tkinter.CENTER, expand=True)

self.btn\_exit = tkinter.Button(window, text="Exit", width=50, command=self.exit\_window)

self.btn\_exit.pack (anchor=tkinter.CENTER, expand=True)

# After it is called once, the update method will be automatically called every delay milliseconds

self.delay = 15

self.update ()

self.window.mainloop ()

def exit\_window (self):

print("[INFO] closing...")

self.window.destroy ()

cv2.destroyAllWindows () # it is not mandatory in this application

def snapshot (self):

# Get a frame from the video source

ret, frame = self.vid.get\_frame ()

if ret:

cv2.imwrite ("snapshot/frame-" + time.strftime ("%d-%m-%Y-%H-%M-%S") + ".jpg", cv2.cvtColor (frame, cv2.COLOR\_RGB2BGR))

def update (self):

# Get a frame from the video source

ret, frame = self.vid.get\_frame ()

if ret:

self.photo = PIL.ImageTk.PhotoImage (image=PIL.Image.fromarray (frame))

self.canvas.create\_image (0, 0, image=self.photo, anchor=tkinter.NW)

self.window.after (self.delay, self.update)

class MyVideoCapture:

print ("[INFO] loading model...")

net = cv2.dnn.readNetFromCaffe (args["prototxt"], args["model"])

def \_\_init\_\_ (self, video\_source=0):

# Open the video source

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

self.vid = cv2.VideoCapture(video\_source)

time.sleep (2.0)

self.fps = FPS ().start ()

if not self.vid.isOpened ():

raise ValueError ("Unable to open video source", video\_source)

# Get video source width and height

self.width = self.vid.get (cv2.CAP\_PROP\_FRAME\_WIDTH)

self.height = self.vid.get (cv2.CAP\_PROP\_FRAME\_HEIGHT)

def get\_fram (self):

if self.vid.isOpened ():

ret, frame = self.vid. Read ()

(h, w) = frame. Shape [:2]

blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)),

0.007843, (300, 300), 127.5)

self.net.setInput (blob)

detections = self.net.forward ()

for i in np.arange (0, detections.shape[2]):

confidence = detections [0, 0, i, 2]

if confidence > args["confidence"]:

idx = int (detections [0, 0, i, 1])

box = detections [0, 0, i, 3:7] \* np.array w, h, w, h])

(startX, startY, endX, endY) = box.astype ("int")

label = "{}: {:.2f}%".format(CLASSES[idx], confidence \* 100)

cv2.rectangle(frame, (startX, startY), (endX, endY), COLORS[idx], 2)

y = startY - 15 if startY - 15 > 15 else startY + 15

cv2.putText(frame, label, (startX, y), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, COLORS[idx], 2)

if ret:

# Return a boolean success flag and the current frame converted to BGR

return (ret, cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB))

else:

return (ret, None)

else:

return None

# Release the video source when the object is destroyed

def \_\_del\_\_ (self):

if self.vid.isOpened ():

self.vid.release()

self.fps.stop()

# Create a window and pass it to the Application object

App(tkinter.Tk(), "Object Detection App")