

import tkinter as tk

import settings

import cv2

from PIL import Image, ImageTk

import numpy as np

import imutils

def start\_video():

settings.start\_video = True

show\_frame()

def stop\_video():

settings.start\_video = False

settings.start\_processing = False

lmain.config(image='')

def start\_process():

settings.start\_processing = True

def stop\_process():

settings.start\_processing = False

def show\_frame():

if not settings.start\_video:

return None

\_, frame = cap.read()

frame = cv2.flip(frame, 1)

frame = imutils.resize(frame, width=400)

if settings.start\_processing:

frame = process\_frame(frame)

cv2image = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

img = Image.fromarray(cv2image)

imgtk = ImageTk.PhotoImage(image=img)

lmain.imgtk = imgtk

lmain.configure(image=imgtk)

lmain.after(10, show\_frame)

def process\_frame(img):

# grab the frame dimensions and convert it to a blob

(h, w) = img.shape[:2]

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

0.007843, (300, 300), 127.5)

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

# predictions

net.setInput(blob)

detections = net.forward()

# loop over the detections

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

# extract the confidence (i.e., probability) associated with

# the prediction

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

# filter out weak detections by ensuring the `confidence` is

# greater than the minimum confidence

if confidence > 0.2:

# extract the index of the class label from the

# `detections`, then compute the (x, y)-coordinates of

# the bounding box for the object

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")

# draw the prediction on the frame

label = "{}: {:.2f}%".format(CLASSES[idx],

confidence \* 100)

cv2.rectangle(img, (startX, startY), (endX, endY),

COLORS[idx], 2)

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

cv2.putText(img, label, (startX, y),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, COLORS[idx], 2)

return img

CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",

"bottle", "bus", "car", "cat", "chair", "cow", "diningtable",

"dog", "horse", "motorbike", "person", "pottedplant", "sheep",

"sofa", "train", "tvmonitor"]

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

# load our serialized model from disk

print("Loading model...")

net = cv2.dnn.readNetFromCaffe('MobileNetSSD\_deploy.prototxt.txt', 'MobileNetSSD\_deploy.caffemodel')

cap = cv2.VideoCapture(0)

window = tk.Tk()

window.title("object detection")

window.geometry('700x420')

lbl = tk.Label(window, text="Object Detection", font=("Arial Bold", 24))

lbl.grid(column=1, row=0)

imageFrame = tk.Frame(window, width=600, height=500)

imageFrame.grid(row=1, column=1, padx=10, pady=2)

lmain = tk.Label(imageFrame, text="Press Start Video")

lmain.grid(row=1, column=1)

startVideoStreamBtn = tk.Button(window, text="Start Video", command=start\_video)

startVideoStreamBtn.grid(column=0, row=2, padx=15)

stopVideoStreamBtn = tk.Button(window, text="Stop Video", command=stop\_video)

stopVideoStreamBtn.grid(column=0, row=3, padx=15)

startProcessBtn = tk.Button(window, text="Start Detection", command=start\_process)

startProcessBtn.grid(column=1, row=2)

stopProcessBtn = tk.Button(window, text="Stop Detection", command=stop\_process)

stopProcessBtn.grid(column=1, row=3)

window.mainloop()

import cv2

import argparse

import numpy as np

ap = argparse.ArgumentParser()

ap.add\_argument('-i', '--image', required=True,

help = 'path to input image')

ap.add\_argument('-c', '--config', required=True,

help = 'path to yolo config file')

ap.add\_argument('-w', '--weights', required=True,

help = 'path to yolo pre-trained weights')

ap.add\_argument('-cl', '--classes', required=True,

help = 'path to text file containing class names')

args = ap.parse\_args()

def get\_output\_layers(net):

layer\_names = net.getLayerNames()

output\_layers = [layer\_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]

return output\_layers

def draw\_prediction(img, class\_id, confidence, x, y, x\_plus\_w, y\_plus\_h):

label = str(classes[class\_id])

color = COLORS[class\_id]

cv2.rectangle(img, (x,y), (x\_plus\_w,y\_plus\_h), color, 2)

cv2.putText(img, label, (x-10,y-10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, color, 2)

image = cv2.imread(args.image)

Width = image.shape[1]

Height = image.shape[0]

scale = 0.00392

classes = None

with open(args.classes, 'r') as f:

classes = [line.strip() for line in f.readlines()]

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

net = cv2.dnn.readNet(args.weights, args.config)

blob = cv2.dnn.blobFromImage(image, scale, (416,416), (0,0,0), True, crop=False)

net.setInput(blob)

outs = net.forward(get\_output\_layers(net))

class\_ids = []

confidences = []

boxes = []

conf\_threshold = 0.5

nms\_threshold = 0.4

for out in outs:

for detection in out:

scores = detection[5:]

class\_id = np.argmax(scores)

confidence = scores[class\_id]

if confidence > 0.5:

center\_x = int(detection[0] \* Width)

center\_y = int(detection[1] \* Height)

w = int(detection[2] \* Width)

h = int(detection[3] \* Height)

x = center\_x - w / 2

y = center\_y - h / 2

class\_ids.append(class\_id)

confidences.append(float(confidence))

boxes.append([x, y, w, h])

indices = cv2.dnn.NMSBoxes(boxes, confidences, conf\_threshold, nms\_threshold)

for i in indices:

i = i[0]

box = boxes[i]

x = box[0]

y = box[1]

w = box[2]

h = box[3]

draw\_prediction(image, class\_ids[i], confidences[i], round(x), round(y), round(x+w), round(y+h))

cv2.imshow("object detection", image)

cv2.waitKey()

cv2.imwrite("object-detection.jpg", image)

cv2.destroyAllWindows()