```
# USAGE
# python yolo_video_live.py --yolo yolo-coco

# import the necessary packages
import numpy as np
import argparse
import imutils
import time
import cv2
import os
from datetime import datetime
import json

#from gtts import gTTS
import os
#from playsound import playsound

language = 'en'

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-y", "--yolo", required=True, help="base path to YOLO
directory")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
help="minimum probability to filter weak detections")
ap.add_argument("-t", "--threshold", type=float, default=0.3,
help="threshold when applyong non-maxima suppression")
args = vars(ap.parse_args())

# load the COCO class labels our YOLO model was trained on
labelsPath = os.path.sep.join([args["yolo"], "obj.names"])
LABELS = open(labelsPath).read().strip().split("\n")

# initialize a list of colors to represent each possible class label
np.random.seed(42)
COLORS = np.random.randint(0, 255, size=(len(LABELS), 3),
      dtype="uint8")

# derive the paths to the YOLO weights and model configuration
weightsPath = os.path.sep.join([args["yolo"], "yolo.weights"])
configPath = os.path.sep.join([args["yolo"], "yolo.cfg"])

# load our YOLO object detector trained on COCO dataset (80 classes)
# and determine only the *output* layer names that we need from YOLO
print("[INFO] loading YOLO from disk...")
net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)
ln = net.getLayerNames()
ln = [ln[i - 1] for i in net.getUnconnectedOutLayers()]

# initialize the video stream, pointer to output video file, and
# frame dimensions
vs = cv2.VideoCapture(0)
(W, H) = (None, None)
```

```
# try to determine the total number of frames in the video file
try:
	prop = cv2.cv.CV_CAP_PROP_FRAME_COUNT if imutils.is_cv2() \
		else cv2.CAP_PROP_FRAME_COUNT
	total = int(vs.get(prop))
	print("[INFO] {} total frames in video".format(total))

# an error occurred while trying to determine the total
# number of frames in the video file
except:
	print("[INFO] could not determine # of frames in video")
	print("[INFO] no approx. completion time can be provided")
	total = -1

# loop over frames from the video file stream
while True:

	# read the next frame from the file
	(grabbed, frame) = vs.read()

	# if the frame was not grabbed, then we have reached the end
	# of the stream
	if not grabbed:
		break

	# if the frame dimensions are empty, grab them
	if W is None or H is None:
		(H, W) = frame.shape[:2]

	# construct a blob from the input frame and then perform a forward
	# pass of the YOLO object detector, giving us our bounding boxes
	# and associated probabilities
	blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416),
		swapRB=True, crop=False)
	net.setInput(blob)
	start = time.time()
	layerOutputs = net.forward(ln)
	end = time.time()

	# initialize our lists of detected bounding boxes, confidences,
	# and class IDs, respectively
	boxes = []
	confidences = []
	classIDs = []

	# loop over each of the layer outputs
	for output in layerOutputs:
		# loop over each of the detections
		for detection in output:
			# extract the class ID and confidence (i.e.,
probability)
			# of the current object detection
			scores = detection[5:]
			classID = np.argmax(scores)
```

```
                confidence = scores[classID]

                # filter out weak predictions by ensuring the detected
                # probability is greater than the minimum probability
                if confidence > args["confidence"]:
                      # scale the bounding box coordinates back relative
to
                      # the size of the image, keeping in mind that YOLO
                      # actually returns the center (x, y)-coordinates
of
                      # the bounding box followed by the boxes' width
and
                      # height
                      box = detection[0:4] * np.array([W, H, W, H])
                      (centerX, centerY, width, height) =
box.astype("int")

                      # use the center (x, y)-coordinates to derive the
top
                      # and and left corner of the bounding box
                      x = int(centerX - (width / 2))
                      y = int(centerY - (height / 2))

                      # update our list of bounding box coordinates,
                      # confidences, and class IDs
                      boxes.append([x, y, int(width), int(height)])
                      confidences.append(float(confidence))
                      classIDs.append(classID)

      # apply non-maxima suppression to suppress weak, overlapping
      # bounding boxes
      print(confidences)
      print(["threshold"])
      idxs = cv2.dnn.NMSBoxes(boxes, confidences,
args["confidence"],args["threshold"])

      # ensure at least one detection exists
      if len(idxs) > 0:
            # loop over the indexes we are keeping
            for i in idxs.flatten():
                  # extract the bounding box coordinates
                  (x, y) = (boxes[i][0], boxes[i][1])
                  (w, h) = (boxes[i][2], boxes[i][3])

                  # draw a bounding box rectangle and label on the frame
                  color = [int(c) for c in COLORS[classIDs[i]]]
                  cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
                  text = "{}:
{:.4f}".format(LABELS[classIDs[i]],confidences[i])
                  cv2.putText(frame, text, (x, y -
5),cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

                  #myobj = gTTS(text=LABELS[classIDs[i]], lang=language,
slow=False)
```

```
            #date_string = datetime.now().strftime("%d%m%Y%H%M%S")
            #filename = "voice"+date_string+".mp3"
            #myobj.save(filename)
            #playsound(filename)
            #os.remove(filename)

# display frame
cv2.imshow('frame', frame)
if cv2.waitKey(1) & 0xFF == ord('q'):
      break
```