Security Object Detection, Surveillance

Cynthia Milan, Edgar Gallegos and Manuel Rodriguez

School of Computer Science and Engineering, CSUSB

Abstract

Computer Neural Networks(CNN) has been a common tool that is defined as image and video recognition. This is commonly used to identify objects, faces and a variety of other items in a frame. Models are used to identify objects through the convolution of connected layers in which each layer contains a single image with a given dimension [8]. Using the data calculated and linear algebra, a computation of the algorithm is made to complete the processed output image and identification. The team plans to implement a detection algorithm using the following: Raspberry Pi, Tensor Flow Lite implemented with COCO, and Python. The results are not expected to be ideal, but enough to work with a 70 to 80 percent accuracy and still remain within the budget. This will be useful for home security detection that enables the software to not only detect objects but also identify them with at least a 60 percent accuracy.

1. Introduction

Security and safety have always been a great issue, making it a great benefit to implement security features in the place of comfort. Anyone would agree that such technologies that can solve security problems could be a great asset to many lives. Using computer vision and neural networks, we can introduce a convenient solution to the lack of safety of any individual's home.

Computer vision and neural network technologies are based on image and video recognition using convoluted connected layers of a variety of dimensions [8]. Using a coded algorithm, the data can be calculated to inform the user of the percentage of identification of the object in front of the camera recording. For example, if you put a coca-cola can in front of the camera it should be able to at least inform the user of it being 60 percent a can, etc. Such designs can cause advantages in security because it allows the user to be warned of the object that may have disrupted the safety of their home. If an ordered item from Amazon Co. was considered "delivered" but it was not found in the descriptive delivery information, the program should be able to identify the situation of it being taken by an animal or a person. This is a great advantage to the user because if the theft was greatly hidden, any slight picture can be identified by the program to ensure the percentages of the object, making the program very beneficial.

Our objective is to develop a computer vision program, embedded in a raspberry pi to use it with a variety of applications. In our case, we will implement it for a security system for homes. The main objective is to layout an image recognition software to tackle this problem embedded in raspberry pi using Python, TensorFlow, Linux, and coco. Tensorflow is an open-sourced software developed by Google, that allows us to use COCO and is available for Linux, another open-source operating system.

2. Related work

Several projects have been tested to represent the idea of image recognition. For example, according to an abstract article from the book Automation in Construction, computer vision has experimented with construction projects in Wuhan, China [7]. They use Convolutional Neural Network (CNN) to record and detect the activities of construction workers. The goal of this observation is for the supervisors to verify that the projects and jobs are done accordingly.

Also, two articles discuss how CNN works, and what computer vision exactly is. They discuss challenges that require taking light, deformity of objects, hidden images, 2 or 3-dimensional figures, and a variety of dynamics into consideration [1]. Although these challenges experiment with computer vision. The articles thoroughly explain how convolutions take place in identifying numerous dimensional layers and set a pool of filters to collect information.

A Tutorial used to assist in the creation of the experiment with the raspberry pi object detection was a blog by Leigh Johnson written by Adrian Rosebrock, "Real-time Object Tracking with TensorFlow, Raspberry Pi, and Pan-Tilt HAT", in which she describes the materials used to create the object detector. She discussed the assembly of a TensorFlow Lite model called MobileNetV3-SSD attached to a Raspberry Pi. She also explained how the tracking was implemented onto the design, by using a proportional-integral-derivative controller(PID) controller.

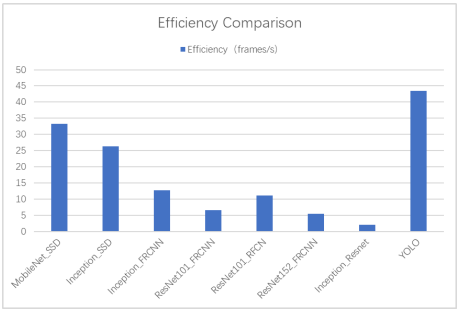

Figure 1. Efficiency Comparison of Eight Groups of Experiments [11]

To accomplish the detector she also implemented an accelerator using Coral's USB Edge TPU Accelerator and the Edge TPU Compiler [6]. Before continuing the tutorial she provides a detailed list of terms that assists the reader in understanding each term by providing definitions, such as MobileNetV3, a computer vision model in which is resourceful for mobile phone processors. As seen in figure 1, Mobilnet is very efficient compared to when compared to others. The setup provided benchmark data of an estimated 7 to 8 frames per second. She added a custom operation TFLite_Detection_PostProcess that implements a technique called Non-Maximum Suppression, which filters multiple box proposals [6]. This although not planned to experiment with presently, is still something the team plans to experiment and implement for further advanced research and testing.

A better tutorial closer to the experiment we did is "How to perform Object Detection with TensorFlow Lite on Raspberry Pi" by Shawn Hymel, which describes a setup using coco as the MobileNetV1, just as described in the previous tutorial. Since this tutorial actually uses COCO as the team planned to experiment with, this one makes it a closer method almost identical to the one implemented. COCO is described as a collection of a variety of figures labeled by multiple companies like Microsoft and universities specialized in research of computer vision [5]. Just like the previous design this one also uses TensorFlow Lite, Linux, Python, and a Raspberry Pi. However, instead of using MobileNetV3, MobileNetV1 is used. They also describe different versions of the Raspberry Pi used, this is helpful because they explain how each works, but one stands out the most for having a faster processor and more memory implemented on the small computer board. Raspberry Pi 4B is recommended for its processing speed and memory, compared to Raspberry Pi 3B [5]. Raspberry Pi 3B tends to give you a smaller amount of frames per second, compared to the 5 frames per second for the Raspberry Pi 4B, either 4 gigabytes or 8 gigabytes work.

3. Method

Security surveillance is very important in the present time because there tends to be a lot of theft. Leaving people unsteady when it comes to the holidays and every other day. This also comes useful in companies like construction that require constant surveillance to make sure their workers are doing what they should be doing. The problem is to implement a design representing computer vision that allows detecting images live for safety purposes. As explained above there are numerous ways that a Raspberry Pi can be used to create a live image detector. We go forward with using a Raspberry Pi because it allows us to code the algorithm in Linux using python, it also has enough memory and a fast enough processor to detect at least 4 frames per second. It is not as clear since we must stay under budget. Implementing a graphics card like Nvidia, commonly used in PCs, for better graphics, or a 12 MP camera, for clearer pictures would not benefit the project since we are trying to stay under budget. Thus, Raspberry Pi, Linux, a budgeted camera, python, and the required files will have to do. The fact it works allows us to test further research along the way staying under budget and saving money.

TensorFlow Lite was used with COCO, a program installed with over a thousand detected images developed by Google, making the algorithm easier to read the live detection we plan to test. The frames per second aren't planned to be high enough but that is fair based on the board and camera used. There are many boards researched to use, but raspberry pi was the most convenient since it has the camera that works with it, the memory stands enough, and it works with the free open-source operating system Linux. We aren't expecting it to work ideally with a higher resolution or frames per second, but it allows it to detect an image with at least 60 percent accuracy and an estimate of 70 to 80 percent accuracy.

Libraries used in the python algorithm consist of a few like numpy, tflite, and cv2. The coded algorithm defines a video stream class that handles the streaming of the video from the installed USB webcam, or a Picamera. The code will loop indefinitely with a thread that reads the frames per second until the program is stopped. The TensorFlow libraries will then be used to calculate the runtime, and a load_delehgate library is imported if a Coral Edge TPU is to be used. Once the timer starts, a frame is captured and resized ideally (1xHxWx3), to normalize the pixels in the image that allows the algorithm to perform the actual detection and retrieve the results. Which then will be recorded in another file for further analysis and experimentation.

Furthermore, this experiment can be tested multiple times as the budget increases, and different programs are embedded to allow for a cleaner, faster, and more efficient image detector. Adding additional software and libraries

that can help the detection boxes, like the technique Non-Maximum Suppression can be something to advance the computer vision research and test for better working security surveillance that allows for faster detection with higher accuracy. This was used in one of the mentioned tutorials, which is beneficial for us to have because this technique is of interest to implement and embed in the raspberry pi.

The other tutorial also was useful for the COCO file from TensorFlow, developed by Google. This was used to search for data to help identify the object detected on the live camera embedded onto the raspberry pi.

4. Experiment

We tested our algorithm to detect humans. Table 1 shows the components used as follows:

Classification Model	**Detection Model**
Operating System	Raspbian OS
CPU	Braodcom 1.5 GHz quad-core A72 64-bit
GPU	Broadcom VideoCore VI
Memory	8GB
Tensorflow	1.15
Camera Sensor	Raspberry Pi Camera Board v2 8MP

Table 1. Experimental Environment

There are many techniques used. However, the mainly used activations are sigmoid, tanh, relu and leaky relu [2]. These techniques, along with COCO, allowed us to train our algorithm to for the detection of humans, regardless of of the type of clothing worn. As demonstrated in figure 2, the algorithm detects a person. The training set was trained using pre-trained models of each classification model. The accuracy of the algorithm depends on the type of camera used. To reduce high variance, regularization techniques and data augmentation techniques can be implemented. Another technique to reduce overfitiing is to train the data on large datasets. We used a Raspberry Pi Camera Board v2 8MP to perform our experiment. We believe that using a higher quality camera will produce better results. The algorithm is able to detect a person, even if there is movement. As demonstrated in figure 3, the algorithm is able to detect the person, regardless of the person's previous position. This is important since the algorithm must be able to detect fast moving objects. Our setup was limited by the hardware we used, nevertheless we predict the algorithm can be trained to be far more efficient. For experimental purposes, we decided that a simple setup was satisfactory enough for our needs.

The accuracy of the sensor also showed is weakness as the light source diminished. This means, the darker the

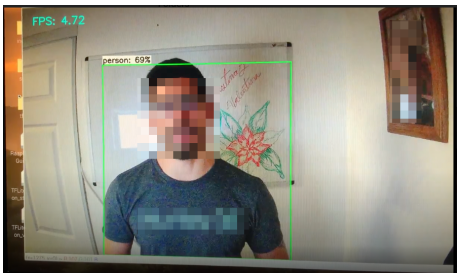

Figure 2. Computer vision detecting a person

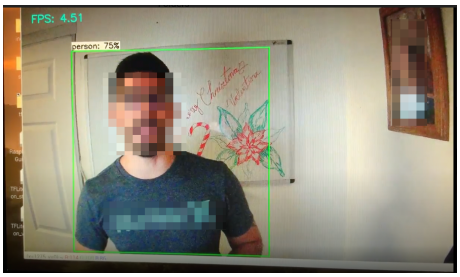

Figure 3. Even if the person moves, the algorithm is able to detect the change

scene, the lesser the accuracy of the algorithm becomes. However, our experiment shows that the algorithm can detect individual that steal packages, which mostly happens during the day, in broad daylight. Most of the limitations we faced were centered around hardware deficiencies. Conversely, conducting the experiment was done in a more ideal environment. This means lighting conditions were ideal, distance from the sensor was 24cm and the test subject was relatively in the line of sight of the sensor, meaning the test object was not at a difficult angle for the sensor to detect. Additionally, we were not able to test a multitude of test subjects, although we are confident that the algorithm would be able to detect them. Our goal was to produce an algorithm that detects a small group of people, since home invasions, burglaries and package snatching typically involves a small group of individuals.

5. Evaluation and Discussion

We used human detection by implementing Convolutional Neural Network (CNN) since is the most widely used deep learning technology, and it performs very well in the fields of image classification and object detection [11]. This allowed us to use existing libraries that have been trained to detect people at a distance without interfering with the other functions of a camera system. CNN is basically basically several layers staged together just like another neural network structure [3]. A layer consists of linear filters which is followed by a non-linear activation function. A pooling layer then followed by convolutional

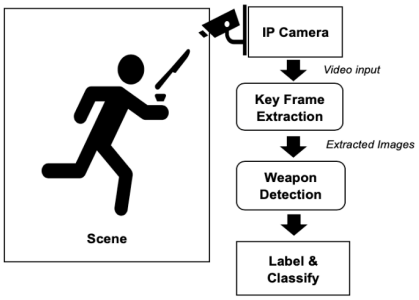

Figure 4. Functional blocks of the proposed system [12]

layer. Max pooling is simply taking the maximum value from predetermined window. Additoinally, CNN is composed of mesh segmentation. From a machine learning point of view, mesh segmentation can be broadly categorised as unsupervised and supervised segmentation [4]. It requires a higher level understanding of the 3D shapes, as composition of an object often relates to shapes and functionality of its parts. This allows for fully connected neural networks, NN. It consist of several fully connected layers followed by a classification layer to produce prediction probabilities.

COCO object detection was used as shown in figure 6. Fine tuning can result in a more efficient model. The COCO dataset consists of 328 000 images, of which there are 2.5 million labelled instances, and about 90 categories [9]. After having satisfactory test results, it is possible to to export a photo file or live video to use for object detection. Furthermore, CNN can also be used to detect potentially dangerous intruders equipped with firearms. Figure 4 shows the functional blocks of the proposed system. After the video is captured by the surveillance camera, it is passed to the key frame extraction subsystem, which reduces data size by selecting key frames for feasible real-time running of the subsequent steps [12]. Additionally, techniques can be developed to increase the efficiency of human detection, such as at night. For example, an algorithm can be based on the fact that thermal images produced by thermal cameras make it possible to see any environment with or without light. The amount of radiation emitted by an object increases with the temperature and the variations in temperature can be visualized for detecting a person [10].

Figure 5 demonstrates how a night vision algorithm can be implemented. This can be useful to add extra functionally to our existing algorithm. This is what is fascinating about CNN, it can be expanded upon and improved to meet different use cases. Now, instead of having just a regular daytime surveillance system, a person can have a system where it does multiple functions, such as detecting a person and determining if a weapon is being carried. Additionally, the system can be setup to alert the user if it detects certain parameters. Our algorithm is more modest, focusing on the detection of humans with expandability in mind for future updates.

Figure 5. (a) is the original image and (b) is binary with human detected [10]

Figure 6. Detection result of training algorithm [3]

6. Conclusion

Computer vision and neural networks are mainly used as security surveillance for the safety of homes and companies. This goes from at-home safety to labor worker safety in construction sites. The image detector uses a variety of layers and boxes with different pixels, using linear algebra, the data is then calculated to identify the image with a percentage of certainty and the used frames per second based on the materials used. There are many tutorials used to help create the surveillance device embedded in raspberry pi, using Linux, TensorFlow, and COCO. One uses MobileNetV3, while the other uses MobileNetV1, but due to staying under budget, the second tutorial was the best way to go since the first one required more materials. It uses a better camera and software system that can also be used for a more efficient algorithm.

We also looked at different boards and found the most efficient way to go, that being a raspberry pi since it works with Linux, the operating system, and python, the programming language. Overall, we were successful in creating an algorithm that detects humans using existing COCO datasets and Tensorflow Lite to accomplish our goal. Furthermore, the algorithm can be modified and improved upon to increase its functionality and efficiency. the algorithm is able to detect humans with 70% accuracy or better. Combined with different hardware, it has the potential of replacing different equipment in addition in aiding in home surveillance. We have also demonstrated how CNN can be used to detect other objects, not just humans.

References

[1] Florian Bansac. Computer vision and convolutional neural networks.

[2] R. Chauhan, K. K. Ghanshala, and R. C. Joshi. Convolutional neural network (cnn) for image detection and recognition. In *2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC)*, pages 278–282, 2018.

[3] D. M. Dinama, Q. A'yun, A. D. Syahroni, I. Adji Sulistijono, and A. Risnumawan. Human detection and tracking on surveillance video footage using convolutional neural networks. In *2019 International Electronics Symposium (IES)*, pages 534–538, 2019.

[4] David George, Xianghua Xie, and Gary KL Tam. 3d mesh segmentation via multi-branch 1d convolutional neural networks. *Graphical Models*, 96:1 – 10, 2018.

[5] Shawn Hymel. Real-time object tracking with tensorflow, raspberry pi, and pan-tilt hat.

[6] Leigh Johnson. How to perform object detection with tensorflow lite on raspberry pi.

[7] Hanbin Luo, Chaohua Xiong, Weili Fang, Peter E.D. Love, Bowen Zhang, and Xi Ouyang. How to perform object detection with tensorflow lite on raspberry pi. *Automation in Construction*, 94:282 – 289, 2018.

[8] KISHAN MALADKAR. Overview of convolutional neural network in image classification.

[9] S. W. Pienaar and R. Malekian. Human activity recognition using visual object detection. In *2019 IEEE 2nd Wireless Africa Conference (WAC)*, pages 1–5, 2019.

[10] S. K. Sharma, R. Agrawal, S. Srivastava, and D. K. Singh. Review of human detection techniques in night vision. In *2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, pages 2216–2220, 2017.

[11] L. Xie and X. Guo. Object detection and analysis of human body postures based on tensorflow. In *2019 IEEE International Conference on Smart Internet of Things (SmartIoT)*, pages 397–401, 2019.

[12] S. Xu and K. Hung. Development of an ai-based system for automatic detection and recognition of weapons in surveillance videos. In *2020 IEEE 10th Symposium on Computer Applications Industrial Electronics (ISCAIE)*, pages 48–52, 2020.

Appendix

Main Code

```
# Webcam Object Detection Using Tensorflow-trained Classifier
#
# This program uses a TensorFlow Lite model to perform object detection on a live webcam
# feed. It draws boxes and scores around the objects of interest in each frame from the
# webcam. To improve FPS, the webcam object runs in a separate thread from the main program.
# This script will work with either a Picamera or regular USB webcam.
#
#
#
#
# Import packages
import os
import argparse
import cv2
import numpy as np
import sys
import time
from threading import Thread
import importlib.util


# Define VideoStream class to handle streaming of video from webcam in separate processing thread
class VideoStream:
"""Camera object that controls video streaming from the Picamera"""
def __init__(self,resolution=(640,480),framerate=30):
# Initialize the PiCamera and the camera image stream
self.stream = cv2.VideoCapture(0)
ret = self.stream.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc(*'MJPG'))
ret = self.stream.set(3,resolution[0])
ret = self.stream.set(4,resolution[1])

# Read first frame from the stream
(self.grabbed, self.frame) = self.stream.read()

# Variable to control when the camera is stopped
self.stopped = False

def start(self):
# Start the thread that reads frames from the video stream
Thread(target=self.update,args=()).start()
return self

def update(self):
# Keep looping indefinitely until the thread is stopped
while True:
# If the camera is stopped, stop the thread
if self.stopped:
# Close camera resources
self.stream.release()
return

# Otherwise, grab the next frame from the stream
(self.grabbed, self.frame) = self.stream.read()

def read(self):
```

```
# Return the most recent frame
return self.frame

    def stop(self):
# Indicate that the camera and thread should be stopped
self.stopped = True

    # Define and parse input arguments
parser = argparse.ArgumentParser()
parser.add_argument('–modeldir', help='Folder the .tflite file is located in',
required=True)
parser.add_argument('–graph', help='Name of the .tflite file, if different than detect.tflite',
default='detect.tflite')
parser.add_argument('–labels', help='Name of the labelmap file, if different than labelmap.txt',
default='labelmap.txt')
parser.add_argument('–threshold', help='Minimum confidence threshold for displaying detected objects',
default=0.5)
parser.add_argument('–resolution', help='Desired webcam resolution in WxH. If the webcam does not support the resolution entered, errors may occur.',
default='1280x720')
parser.add_argument('–edgetpu', help='Use Coral Edge TPU Accelerator to speed up detection',
action='store_true')

    args = parser.parse_args()

    MODEL_NAME = args.modeldir
GRAPH_NAME = args.graph
LABELMAP_NAME = args.labels
min_conf_threshold = float(args.threshold)
resW, resH = args.resolution.split('x')
imW, imH = int(resW), int(resH)
use_TPU = args.edgetpu

    # Import TensorFlow libraries
# If tflite_runtime is installed, import interpreter from tflite_runtime, else import from regular tensorflow
# If using Coral Edge TPU, import the load_delegate library
pkg = importlib.util.find_spec('tflite_runtime')
if pkg:
from tflite_runtime.interpreter import Interpreter
if use_TPU:
from tflite_runtime.interpreter import load_delegate
else:
from tensorflow.lite.python.interpreter import Interpreter
if use_TPU:
from tensorflow.lite.python.interpreter import load_delegate

    # If using Edge TPU, assign filename for Edge TPU model
if use_TPU:
# If user has specified the name of the .tflite file, use that name, otherwise use default 'edgetpu.tflite'
if (GRAPH_NAME == 'detect.tflite'):
GRAPH_NAME = 'edgetpu.tflite'

    # Get path to current working directory
CWD_PATH = os.getcwd()

    # Path to .tflite file, which contains the model that is used for object detection
PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,GRAPH_NAME)

    # Path to label map file
PATH_TO_LABELS = os.path.join(CWD_PATH,MODEL_NAME,LABELMAP_NAME)
    # Load the label map
with open(PATH_TO_LABELS, 'r') as f:
labels = [line.strip() for line in f.readlines()]

    # Have to do a weird fix for label map if using the COCO "starter model" from
# https://www.tensorflow.org/lite/models/object_detection/overview
# First label is '???', which has to be removed.
if labels[0] == '???':
del(labels[0])

    # Load the Tensorflow Lite model.
# If using Edge TPU, use special load_delegate argument
if use_TPU:
interpreter = Interpreter(model_path=PATH_TO_CKPT,
experimental_delegates=[load_delegate('libedgetpu.so.1.0')])
print(PATH_TO_CKPT)
else:
interpreter = Interpreter(model_path=PATH_TO_CKPT)

    interpreter.allocate_tensors()

    # Get model details
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
height = input_details[0]['shape'][1]
width = input_details[0]['shape'][2]

    floating_model = (input_details[0]['dtype'] == np.float32)

    input_mean = 127.5
```

```
    input_std = 127.5

    # Initialize frame rate calculation
frame_rate_calc = 1
freq = cv2.getTickFrequency()

    # Initialize video stream
videostream = VideoStream(resolution=(imW,imH), framerate=30).start()time.sleep(1)

    #for frame1 in camera.capture_continuous(rawCapture, format="bgr",use_video_port=True):
while True:

    # Start timer (for calculating frame rate)
t1 = cv2.getTickCount()

    # Grab frame from video stream
frame1 = videostream.read()

    # Acquire frame and resize to expected shape [1xHxWx3]
frame = frame1.copy()
frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
frame_resized = cv2.resize(frame_rgb, (width, height))
input_data = np.expand_dims(frame_resized, axis=0)

    # Normalize pixel values if using a floating model (i.e. if model is non-quantized)
if floating_model:
input_data = (np.float32(input_data) - input_mean) / input_std

    # Perform the actual detection by running the model with the image as input
interpreter.set_tensor(input_details[0]['index'],input_data)
interpreter.invoke()

    # Retrieve detection results
boxes = interpreter.get_tensor(output_details[0]['index'])[0]
# Bounding box coordinates of detected objects
classes = interpreter.get_tensor(output_details[1]['index'])[0]
# Class index of detected objects
scores = interpreter.get_tensor(output_details[2]['index'])[0]
# Confidence of detected objects
#num = interpreter.get_tensor(output_details[3]['index'])[0]
# Total number of detected objects (inaccurate and not needed)

    # Loop over all detections and draw detection box if confidence is above minimum threshold
for i in range(len(scores)):
if ((scores[i] ¿ min_conf_threshold) and (scores[i] ¡= 1.0)):

    # Get bounding box coordinates and draw box
# Interpreter can return coordinates that are outside of image dimensions, need to force them to be within image using max() and min()
ymin = int(max(1,(boxes[i][0] * imH)))
xmin = int(max(1,(boxes[i][1] * imW)))
ymax = int(min(imH,(boxes[i][2] * imH)))
xmax = int(min(imW,(boxes[i][3] * imW)))

    cv2.rectangle(frame, (xmin,ymin), (xmax,ymax), (10, 255, 0), 2)

    # Draw label
object_name = labels[int(classes[i])] # Look up object name from "labels" array using class index
label = '%s: %d%%' % (object_name, int(scores[i]*100))
# Example: 'person: 72%'
labelSize, baseLine = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 0.7, 2) # Get font size
label_ymin = max(ymin, labelSize[1] + 10) # Make sure not to draw label too close to top of window
cv2.rectangle(frame, (xmin, label_ymin-labelSize[1]-10), (xmin+labelSize[0], label_ymin+baseLine-10), (255, 255, 255), cv2.FILLED) # Draw white box to put label text in
cv2.putText(frame, label, (xmin, label_ymin-7), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 2) # Draw label text

    # Draw framerate in corner of frame
cv2.putText(frame,'FPS: 0:.2f'.format(frame_rate_calc),(30,50),cv2.FONT_HERSHEY_SIMPLEX,1,(255,255,0),2,cv2.LINE_AA)

    # All the results have been drawn on the frame, so it's time to display it.
cv2.imshow('Object detector', frame)

    # Calculate framerate
t2 = cv2.getTickCount()
time1 = (t2-t1)/freq
frame_rate_calc= 1/time1

    # Press 'q' to quit
if cv2.waitKey(1) == ord('q'):
break

    # Clean up
cv2.destroyAllWindows()
videostream.stop()
```