Real-time Object Detection on a Raspberry Pi

Authors: Abhishek Patoliya
Shivam Saraswat

Supervisor: Dr Abhishek Sharma

Semester and Subject: VI, Internet of things

Authors Email : 18ucc025@lnmiit.ac.in
18ucc167@lnmiit.ac.in

Project Work: https://github.com/AMP4323/Real-Time-Object-Detection-with-Raspberry-Pi

Abstract

With the recent advancement of Artificial Intelligence,object detection techniques has been increased in accuracy as well as speed. It is now possible to run high accurate object detections over the real-time speed on computer systems like a desktop. This project explores the running object detection methods on the Raspberry Pi 3b+, an embedded computer board. Two experiments are conducted where two object detection models YOLO and SSD(Single Shot Detection), are used to test accuracy and speed and according to the results at the end, SSD was outperforming YOLO in terms of speed as well as accuracy. Still, with the low processing power of RaspPi, neither SSD nor YOLO performs well in both speed and accuracy on Raspi.

Keywords: Computer Vision, Object Detection, Raspberry Pi

Contents

1.1 Introduction

Computer vision is a field of Artificial Intelligence that enables machines to see and process digital images and videos and analyse them. One application area of computer vision is object detection, a method to locate and identify objects in a snap. With the recent advancement in deep learning and Neural networks (CNN and RNN), computers can from large image dataset learn to recognise and track objects seen in images and videos with great accuracy. This project implements object detector methods on a single board computer, the Raspberry Pi 3B+. The main problem is low computing power which is required Object Detection. Two object detecting methods evaluated by measuring detection accuracy, inference time and throughput (FPS).

The Raspberry Pi is a small single-board computer, and it lacks the computational power of modern desktops, but they can often be used for specific tasks because of their low cost and size.

1.2 Problem Formulation

Real-
time object detection requires a lot of computational power as well as processing power, and wit h machines like Raspi with limited performance, achieving a low speed can be considered a chall enging task in real-
time. Many different methods can be used to detect objects. Two popularmethods are called SSD , YOLO. These detection methods are implemented on a board,
Raspberry Pi 3 B+ to check whether they are able to run on such low-performance hardware machine. An implemented object detection is only considered if it achieves high enough accurac y and frame rate(fps) to help applications in real time. The test is done by implementing methods on Raspi, measuring how they perform detection accuracy, time and frame rates(fps). with one application, a moving person is frame is chosen to get the tests on accuracy.

1.3 Objectives

O1	Build and set up the environment for the detection
O2	Implement the two detection models
O3	Select the objects, object size, input resolution of the image

O4	Test all implementations of the frames and collect the data

.

1.4 Method

1.3.1 Controlled Experiments

A series of experiments will be implemented to collect quantitative data. Multiple tests will be run on each of the implementations with a few varying variables. One of the variables which are to be measured is detection accuracy. which means how confident the detector is that the detection it made is correct. Another variable that is to be measured is inference time, which is the time taken for the neural networks to calculate and output the predictions for frames. The last dependent variable is FPS; the average frame rate detector achieved while processing the video input. The varying variables used in the experiments are objected size/distance and image input resolution.

1.3.2 Reliability

To increase the experiments' internal validity, the Raspberry Pi is booted without unnecessary services and processes. Pre-recorded videos are used in project, so the duplicate frames are used when measuring the model's accuracy.

1.3.2 Hardware

The main hardware used is a Raspberry Pi Model 3B+ which is the latest and most potent product in the Raspberry Pi range. It costs around $35 and comes with a 64-bit ARMv8 quad-core processor clocked at 1.4GHz and 1GB of LPDDR2 SDRAM.

2.1 Implementation

This project's software consists of scripts written in the Python programming language. Many open-source frameworks of deep learning are used for object detection, such as Caffe, Darknet and Tensorflow-Lite. For this project, the open-source computer vision library OpenCV is chosen for the task. OpenCV includes a large selection of algorithms aimed at real-time computer vision, including a DNN module that enables the use of pre-trained models for inference from the previously mentioned frameworks.

2.1.1 Scripts

Detector.py

This script runs the detector using any of the specified models. The first thing it does is load the network. Once the network is loaded, frames are captured from the camera module. Before being passed into the network, the images need to be preprocessed and converted into a blob (Binary Large Object). This preprocessor includes resizing, scaling and normalisation. The images are resized and scaled to fit the chosen network input. The normalisation is done by subtracting RGB values' photos with the mean RGB values of the training set's pictures. This is done to combat illumination changes in the input images. When the blob is ready, a forward pass is run where the blob is propagated through the network, and the predictions are calculated. The time for this process is measured and is what is referred to as inference time. Before showing the final predictions, the detector goes through two steps of filtering to remove imperfect detections. The first one is removing projections with a score lower than the confidence threshold value. Settings this value too low will lead to false detections being included. This value is set to 0.3, which means that any detection with a confidence score lower than 30% is not counted as a detection. The second step of filtering is non-maximum suppression (NMS), which ensures an object is only detected once, removing smaller overlapping bounding boxes. When the filtering is done, bounding boxes are drawn around the detected objects together with a label of the predicted classes and their confidence scores. An overview of the system flow is shown in Figure 1.1.

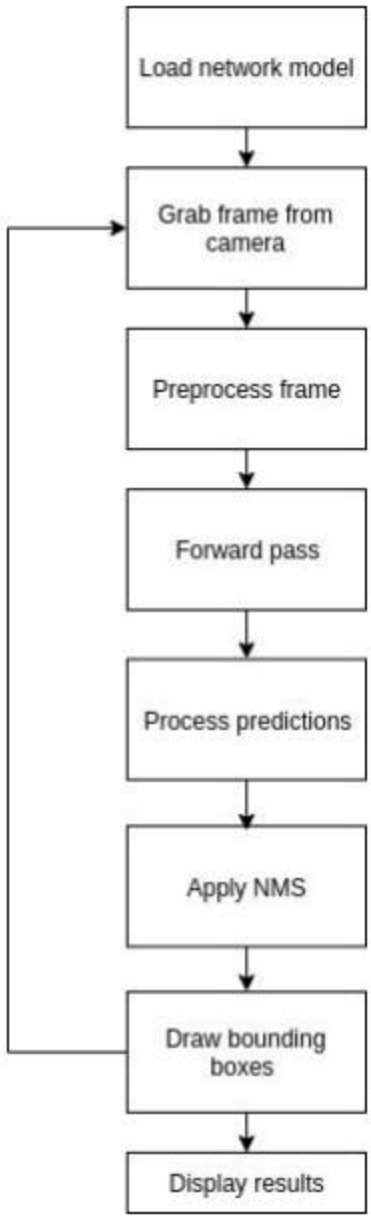

Figure 2.1: Overview of System Flow

models.py

The models.py script contains the functions that are used to load a specific network model. It also includes the configuration variables used when converting the captured frames into input blobs ready to be passed into the network. Table 2.1 shows the details of the models used for the experiments.

Detection Model	SSDLite	YOLOv3-tiny
Classification Network	MobileNetV2	Tiny Darknet
Framework	Tensorflow	Darknet
Config File	ssdlite_mobilenet_v2.pbtext	yolov3-tiny.cfg
Weight File	frozen_interference_graph.pb	yolov3-tiny.weights
Bounding box representation	(left, top, right, bottom)	(center_x, Center_y, width, height)
COCO Labels version	2017	2014

Table 2.1: Object detection models used

2.1.2 Experiment

Both SSD and YOLO are fully convolutional neural networks(CNN).A network where there are no fully connected layers, a layer where all neurons have individual connections with neurons in the previous layer. Being fully convolutional, these networks can run inference on images of varying sizes. This means we can use input size as a parameter to manage a trade-off between speed and detection accuracy.

Average Throughput and Interference time

This experiment aimed to find what rate the models can run at while capturing frames live on a Raspberry Pi. These tests only focused on measuring what speeds they could achieve at different input sizes; how well they could detect objects was measured in experiment two. One hundred frames were captured and processed three times per model at different input sizes: 96x96, 160x160 and 224x224, and the total processing time and the inference time for each frame was recorded. The Raspberry Pi 3 Model B+ CPU clocked is at 1.4GHz by default. While testing the setup, it was noticed that even with the thermal throttle threshold set to 70°C, the detector couldn't run long before overheating and throttling down the CPU. The clock speed was then set to 1.2GHz, at which it ran at a stable frequency and temperature. The memory split was set to allocate 70Mb for the GPU, which is

the minimum required to run the camera and desktop environment. This left as much memory as possible to the CPU for running the models. Another thing to note is the impact the GUI size has on speed. Using a too large resolution can lead to a substantially slower throughput. The GUI resolution used in the experiments was set to 320x240, which doesn't slow down the system too much, and it's still large enough to see what's in the frame.

Detection Accuracy

This experiment was conducted to find out how well the models were able to detect objects. It was done by capturing 80 frames of video with a person at four different distances from the camera and measuring the average confidence the detector achieved in those frames. One application for object detection is home security. Implementing object detection in a surveillance system would enable features such as automatic intruder detection. This is why a person is chosen as the object to be detected in this experiment. Figure 2.2 shows an example of the person being seen at approx—three meters distance, using SSD with 160x160 as input size.

Figure 2.2: SSD detecting a person at approx. three meters distance with the image an input size of 160x160 pixels

2.1.3 Results

Mean throughput and inference

The results displayed here shows at what speed the models could run with different input sizes. The numbers indicate the average of over 100 captured frames. In Figure 2.3, we can see the throughput in frames per second. Figure 2.4 shows inference time in milliseconds.

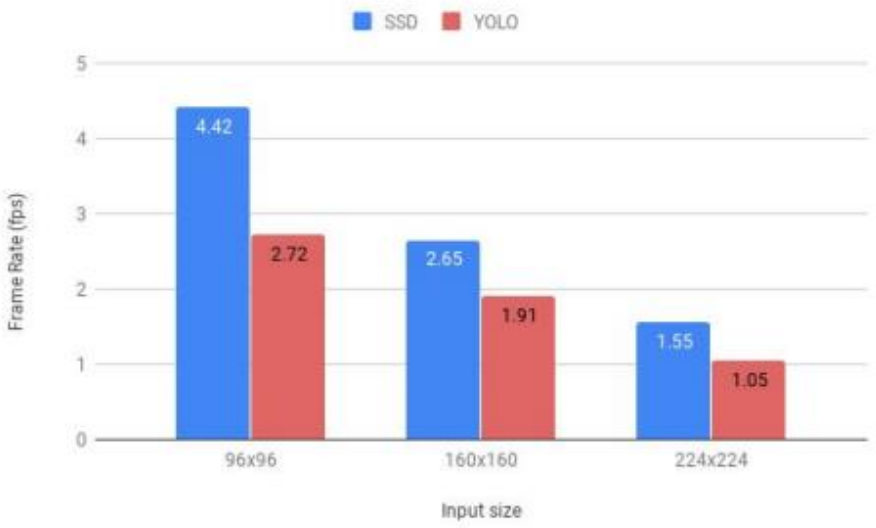

Figure 2.3: Throughput for each model and input size

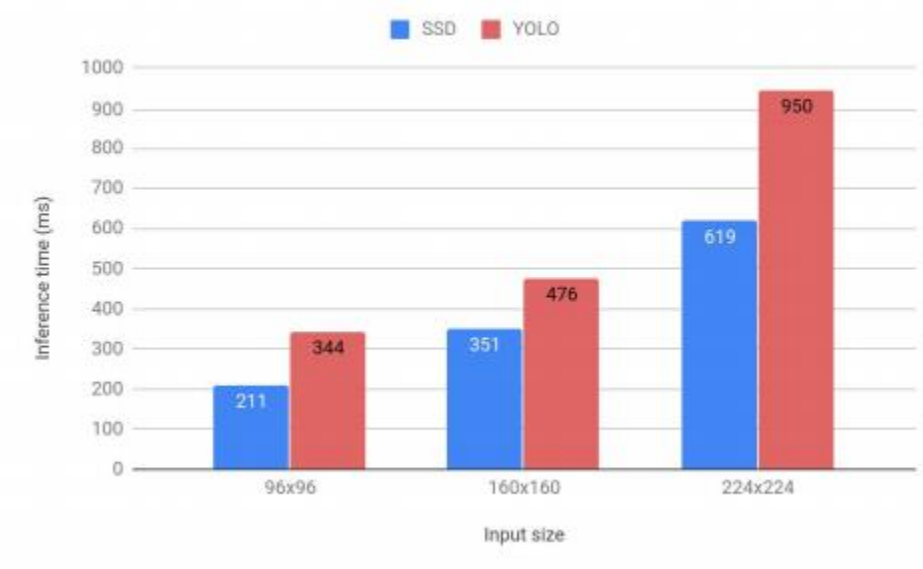

Figure 2.4: Mean inference time for each model and input size

2.1.4 Accuracy

Table 2.2 shows the average classification score of the two object detection methods achieved with varied input size for the person at a different distance for the camera

Distance (meters)	Input size	YOLO	SSD
1	96*96	48%	90%
1	160*160	94%	96%
1	224*224	97%	97%
2	96*96	66%	50%
2	160*160	82%	97%
2	224*224	90%	96%
3	96*96	0%	0%
3	160*160	84%	95%
3	224*224	85%	90%
4	96*96	0%	0%
4	160*160	37%	38%
4	224*224	67%	93%

Table 2.2: Detection accuracy at different input sizes

The data displayed in the table shows how YOLO and SSD perform at different network input sizes. It shows that by using smaller input sizes, speed is gained at the cost of accuracy. In Figure 2.3 and 2.4, we see that SSD outperforms YOLO in speed and Table 2.2 shows that SSD also achieves the same or higher accuracy in 11 of 12 instances. In Figure 2.3, we see at what rate the models could operate. Using 96x96 as input size, the SSD model reached as High as 4.42 frames per second while YOLO only achieved 2.72, which is just a tiny bit faster than SSD at 16 0x160. Both models got very low results when the input size was 224x224. YOLO ran at only 1.05 fps and SSD at 1.55, which is an almost 50% increase and a noticeable difference at these speeds. By analysing Table 2.2, we see how object size impacts the models ability to detect. As the person gets farther away, it becomes smaller in the image, andthe detection gets weaker. Wit

h224x224 as input size, both models could detect the person Robustly simultaneously; only YOLO performed slightly worse at 4 meters. Both models also did well using 160x160 as input size. The person was seen with high accuracy at 1 to 3 meter,but there was a notable drop off at 4 When the input size was 96x96, the results were significantly worse. Both models were able to d etect at 1 and 2 meters, but already at 3 meters, the person was too small to be seen.

2.2 Conclusion

This project aimed to investigate the suitability of running a real-time object detection system on a Raspberry Pi. SSD and YOLO were implemented and tested in accuracy and speed at different input sizes. The results showed that both models are very slow and that only in applications that don't require high speed would it be viable to use the Raspberry Pi as hardware. There is a tradeoff with accuracy to be made if higher\rates are to be achieved since there is not enough computational power to have both.
This leads to the conclusion that it is important to\choose a proper input size to obtain the right balance of speed and accuracy needed for a particular application. This study couldbe of help for others that are looking to implement object detection on similar hardware\to find that balance. Due to the lack of a proper system and pi camera, many things left out could have been done to strengthen this study’s results, such as more test ing of different objects,distances, and input sizes. It would be interesting to train own model on other datasets with lower resolutions and see if it could improve accuracy for smaller objects. Another thing left out in this project was looking at the impact that lighting has on a model's ability to detect objects.

Bibliography

[1] Y. Amit and P. Felzenszwalb, "Object Detection", Computer Vision, pp. 537-542, 2014.
[2] "Convolutional neural network", En.wikipedia.org, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Convolutional_neural_network. [Accessed: 10- Feb- 2019]

[3] A. Krizhevsky, I. Sutskever and G. Hinton, "ImageNet classification with deep convolutional neural networks", Communications of the ACM, vol. 60, no. 6, pp. 84-90, 017.

[4] "What is a Raspberry Pi?", Raspberry Pi, 2019. [Online]. Available: https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/. [Accessed: 06- Mar- 2019]

[5] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, “Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors,” 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[6] D. Velasco-Montero, J. Fernández-Berni, R. Carmona-Galán and Á. Rodríguez-Vázquez, "Performance analysis of real-time DNN inference on Raspberry Pi", Real-Time Image and Video Processing 2018, 2018.

[7] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu and A. Berg, "SSD: Singlehot MultiBox Detector", Computer Vision – ECCV 2016, pp. 21-37, 2016.

[8] Redmon, J., & Farhadi, A. (2017). YOLO9000: Better, Faster, Stronger. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). doi:10.1109/cvpr.2017.690

[9] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft COCO: Common Objects in Context,” Computer Vision – ECCV 2014 Lecture Notes in Computer Science, pp. 740–755, 2014.

[10] "With Lookout, discover your surroundings with the help of AI", Google, 2019. [Online].