Bachelor Degree Project

Real time object detection on a Raspberry Pi

Author: Adam Gunnarsson
Supervisor: Mattias Davidsson
Semester. VT 2019
Subject: Computer Science

Abstract

With the recent advancement of deep learning, the performance of object detection techniques has greatly increased in both speed and accuracy. This has made it possible to run highly accurate object detection with real time speed on modern desktop computer systems. Recently, there has been a growing interest in developing smaller and faster deep neural network architectures suited for embedded devices. This thesis explores the suitability of running object detection on the Raspberry Pi 3, a popular embedded computer board. Two controlled experiments are conducted where two state of the art object detection models SSD and YOLO are tested in how they perform in accuracy and speed. The results show that the SSD model slightly outperforms YOLO in both speed and accuracy, but with the low processing power that the current generation of Raspberry Pi has to offer, none of the two performs well enough to be viable in applications where high speed is necessary.

Keywords: computer vision, object detection, Raspberry Pi

Contents

1 Introduction

Computer vision is a field of computer science which enables machines not only to see but also to process and analyze digital images and videos. One big application area of computer vision is object detection [1], the capability of a computer to locate and identify objects in an image. With the recent advances in deep learning and convolutional neural networks (CNN) [2], computers can from large image datasets learn to recognize and track objects seen in images and videos with great accuracy.

This thesis studies the possibility of implementing an object detector on a single board computer, the Raspberry Pi B+, capable of maintaining real-time frame rate while keeping high precision. The problem we face is the lack of computing power that is required in an object detecting system. Two popular object detecting methods have been selected and are evaluated by measuring detection accuracy, inference time and throughput (FPS).

1.1 Background

Object detection is the process of detecting and defining objects of a certain known class in an image. Only a few years back, this was seen as a hard problem to solve. Before Krizhevsky presented the CNN AlexNet [3] at the ImageNet Large Scale Visual Recognition Competition in 2012, researchers were struggling to find a solution to image classification with very low error rate. Since then, many object detecting methods applying CNN has been presented showing great performance and efficiency. However, much computing power is still needed to run visual tasks effectively and on a device with limited hardware resources, running an object detecting system can be challenging.

The Raspberry Pi [4] is a small single-board computer, no larger than a credit card. Single-board computers lack the computing power of traditional desktop systems but due to their low cost and size they can many times be preferable to use for certain tasks.

1.2 Related work

Huang et. al. wrote a paper [5] about speed and accuracy trade-offs for modern object detectors where they discussed some of the main aspects that

influence the speed and accuracy of object detectors. Some new techniques was identified for improving speed without sacrificing much accuracy. They discovered that limiting the number of region proposals in Faster R-CNN could reduce computation significantly without affecting the precision score too much. They also observed how they could decrease input resolution to reduce inference time and still get high performance on large objects. However, on small objects the results were significantly worse.

In a study by Velasco-Montero et al. [6] they investigated the performance of real time DNN inference on a Raspberry Pi. They compared four different frameworks and four popular deep neural network models used for image classification. The authors demonstrated that it was possible to achieve real time inference speed on a Raspberry Pi Model B . This was however done with image classification. Object detection, which is the focus of this paper is a more computationally demanding task.

1.3 Problem formulation

Real-time object detection requires a lot of processing power and on a system with limited performance, achieving a speed that can be considered as real-time is a challenging task. There are many different methods that can be used to detect objects. Two popular methods are called SSD [7] , YOLO[8]. These methods are implemented on a Raspberry Pi 3 B+ to see if they are suitable to run on such low performance hardware. An implemented object detector is considered suitable if it achieves high enough accuracy and frame rate to be useful in practical applications. The evaluation is done by running a few tests on each detector, measuring how they perform in detection accuracy, inference time and frame rate. With home security as an application area in mind, a person is chosen as the object to be detected in the accuracy tests.

1.4 Motivation

There is no optimal go-to choice method to use in a object detecting system. New methods are presented all the time, each claiming to perform better than the other.

As Huang et al. mentions in their paper [5], many methods focus on achieving high accuracy without considering running time or hardware

limitations. In real practical use applications you have to take speed into consideration as well, making it hard to find a suitable method to use.

The aim of this study is to be of use when deciding what hardware or method to use in a object detection system. If the study succeeds and any solution is proven to work, it can be used to create more cost- and space-effective solutions by replacing traditional desktop systems.

1.5 Objectives

O1	Build and setup the environment for the detector
O2	Implement the two presented methods
O3	Select the objects, object size, input resolution used for the experiments
O4	Test all implementations and collect data

The object detection methods chosen are two state-of-the-art techniques claiming to be fast and accurate. Object detection systems built with similar hardware has shown poor performance before which is why the results is expected to show that the Raspberry Pi 3 will be struggling to achieve higher speeds without sacrificing a lot of accuracy.

1.6 Scope/Limitation

To extract features of a specific object class a trained CNN model is needed. These models are trained by feeding a collection of pre-annotated images to a neural network. This procedure is often very time consuming and is not done in this project. Instead, the models used will be pre-trained on the Common Objects in Context (COCO) [9] dataset, which includes 80 different object categories in over 300 thousand images.

There are many object detection methods that could be tested in this project, but due to the limited time only two are used. The chosen methods are the two most popular object detection methods designed for real time detection. While there are many other variations of these two where the performance differ slightly, including these in the project would only add extra work without affecting the result significantly.

1.7 Target group

This thesis can be useful for companies looking to replace old hardware with new space- and cost-effective systems in areas such as robotics and surveillance. Because of the Raspberry Pi's low price, it can also be interesting for hobbyists wanting to create their own projects at home, for example a surveillance system in a home security solution. Another group that may be interested is the visually impaired. An object detector implemented on a Raspberry Pi would be easy to carry around and having a text-to-speech voice tell about object it sees could be helpful for a person with low vision. Some projects like this exists today where mobile phones are used. One example is Google Lookout [10], an application for Google Pixel devices that uses object recognition to identify objects through the phone's camera and tell the user information about their surroundings.

1.8 Outline

In chapter 2, the method chosen to solve the thesis problem is discussed. Chapter 3 describes in detail the implementation process. In chapter 4, experiment results are displayed. Chapter 5 contains an analysis of the results. Chapter 6 discusses findings and how the thesis problem was answered. Finally, chapter 7 ends the report with a project conclusion.

2 Method

2.1 Controlled experiments

To answer the thesis problem statement, a series of controlled experiments will be conducted to collect quantitative data. Multiple tests will be run on each of the two implementations with a few varying independent variables. One of the variables to be measured is the detection accuracy. That is how confident the detector is that the detection it made is correct. Another variable that is measured is inference time, which is the time it takes for the neural network to calculate and output the predictions for a frame. The last dependent variable is FPS which is the average frame rate the detector achieved while processing the video stream input.

The varying independent variables used in the experiments are object size/distance and image input resolution. More details about the experiments are explained in section 3.2.

2.2 Reliability and Validity

One problem with the reliability of this project is to get the same setting while conducting the experiments. Using the same models, code implementation and hardware could still lead to different results if variables that are hard to replicate such as ambient lighting, objects placement and angles are different.

In order to increase the internal validity of the experiments, the Raspberry Pi is booted without unnecessary services and processes that could cause disturbances and waste CPU-time for other external tasks. Pre-recorded videos are used in experiment 2 so the same exact frames are used when measuring the models accuracy.

To ensure maximum reproducibility of the experiments, all information and code[1] relevant to this project will be available to the reader.

2.3 Hardware

The hardware used is a Raspberry Pi Model B+ which is the latest and most powerful product in the Raspberry Pi range. It costs merely $35 and comes

[1] https://github.com/adagun/detector

with a 64-bit ARMv8 quad-core processor clocked at 1.4GHz and 1GB of LPDDR2 SDRAM. The camera used is the Raspberry Pi Camera Module v2.

3 Implementation

The software in this project consists of scripts written in the Python programming language which is one of the most popular language used for machine learning tasks. There exist several open source deep learning frameworks that can be used for object detection such as Caffe [11], Darknet [12] and Tensorflow [13]. For this project, the open source computer vision library OpenCV has been chosen to handle the task. OpenCV includes a large selection of algorithms aimed at real-time computer vision, including a DNN module which enables the use of pre-trained models for inference from the previously mentioned frameworks. It has been shown that the models in the CPU implementation of this module perform better than the ones in the frameworks they were ported from. [14]. OpenCV was compiled with NEON and FPV3 support to fully take advantage of hardware optimizations in the Raspberry Pis ARM processor.

3.1 Scripts

3.1.1 detector.py

This script runs the detector, using any of the specified models. The first thing it does is to load the network. Once the network is loaded, frames are captured from the camera module. Before being passed in to the network, the images need to be preprocessed and converted into a blob (Binary Large Object). This preprocess includes resizing, scaling and normalisation. The images are resized and scaled to fit the chosen network input. The normalisation is done by subtracting the images RGB values with the mean RGB values of the images in the training set. 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 poor detections. The first one is removing predictions 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) that is used to ensure 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 3.1.

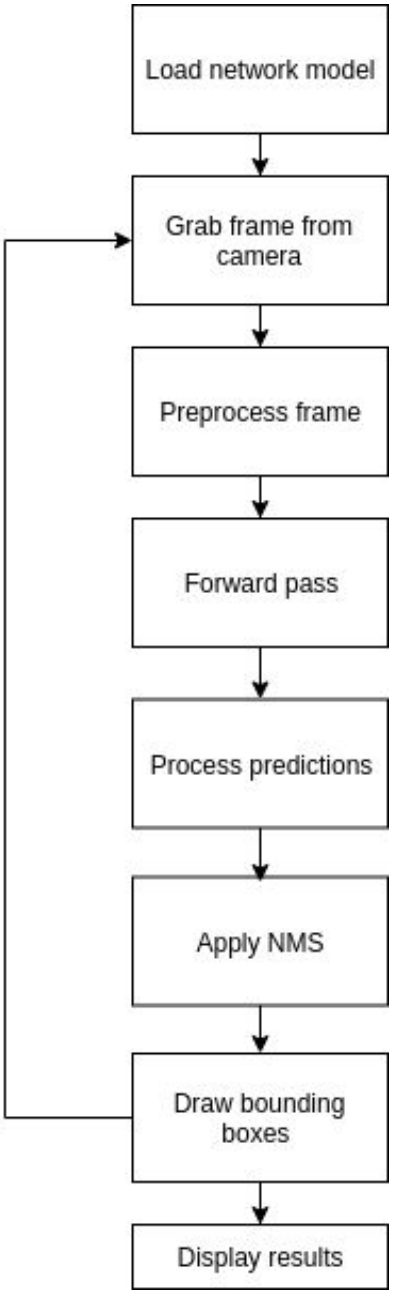

Figure 3.1 Overview of system flow.

3.1.2 models.py

The models.py script contains the functions that is used to load a certain network model. It also contains the configuration variables used when converting the captured frames into input blobs ready to be passed in to the network. Table 3.1 shows 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.pbtxt [17]	yolov3-tiny.cfg [16]
Weight File	frozen_inference_graph.pb [15]	yolov3-tiny.weights [16]
Bounding box representation	(left, top, right, bottom)	(center_x, center_y, width, height)
COCO labels version	2017	2014

Table 3.1 Object detection models used in the experiments.

3.2 The experiments

The goal of this project was to evaluate the suitability of running real time object detection on a Raspberry Pi. To do this, there were two experiments conducted for each object detection model. One to measure the average throughput and inference time, the other to measure detection accuracy.

Both SSD and YOLO are fully convolutional neural networks. That is, a network where there is 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.

3.2.1 Average throughput and inference time

The purpose of this experiment was to find what rate the models are able to run at while capturing frames live on a Raspberry Pi. This tests only focused on measuring what speeds they could achieve at different input sizes, how

well they were able to detect objects was measured in experiment two. 100 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 for 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.

3.2.2 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 in 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 the this experiment. Figure 3.2 shows an example of the person being detected at three meters distance, using SSD with 160x160 as input size.

Figure 3.2: SSD detecting a person at three meters distance with image an input size of 160x160 pixels

4 Results

The data presented in this chapter is the results of the experiments described in section 3.2.

4.1 Mean throughput and inference

The results displayed here shows at what speed the models could run with different input sizes. The numbers show the average over 100 captured frames. In Figure 4.1 we can see the throughput in frames per second. Figure 4.2 shows inference time in milliseconds.

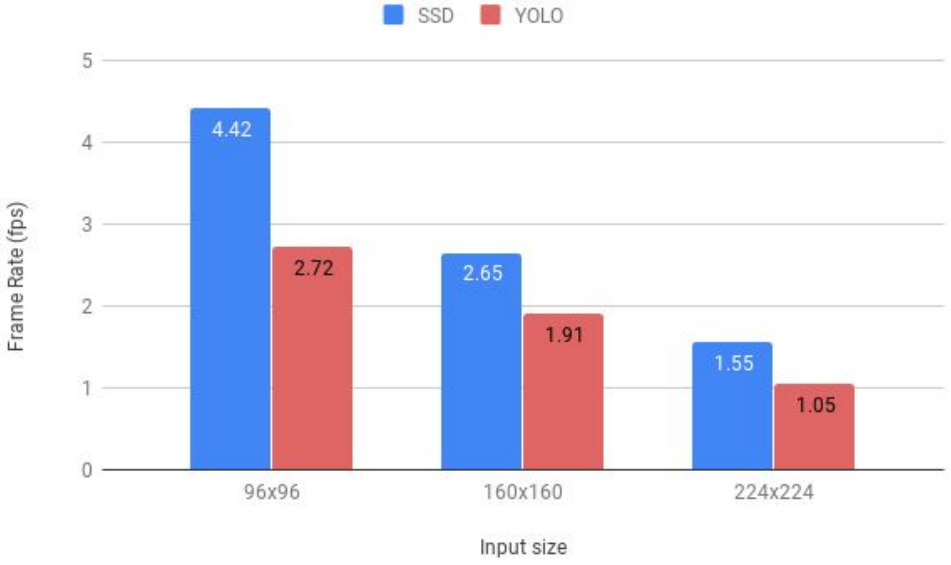

Figure 4.1 Throughput for each model and input size

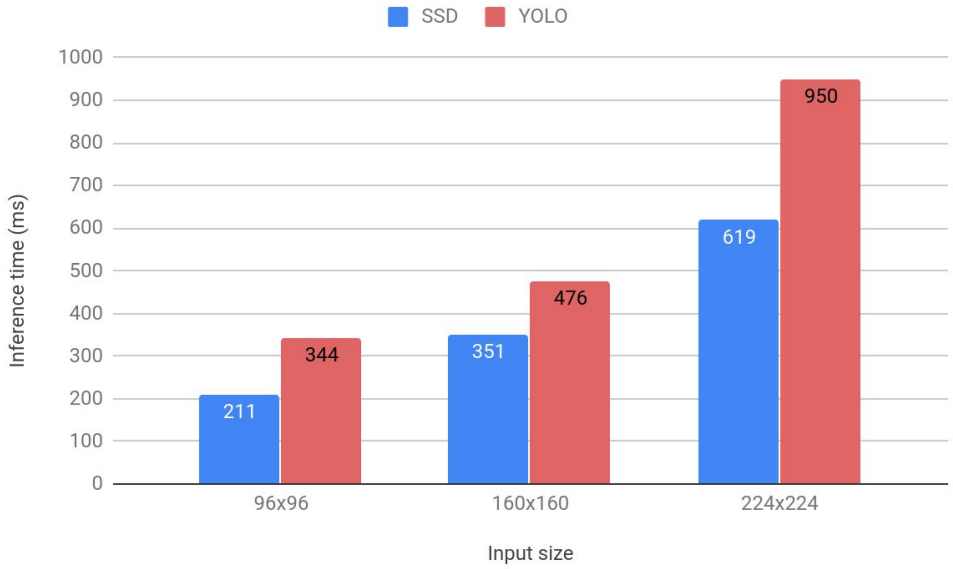

Figure 4.2: Mean inference time for each model and input size

4.2 Accuracy

Table 4.1 shows the average classification score the detector achieved with the different input sizes, while detecting a person at various distances from the camera.

Distance (meters)	**Input size**	**YOLO**	**SSD**
1	96x96	48%	90%
1	160x160	94%	96%
1	224x224	97%	97%
2	96x96	66%	50%
2	160x160	82%	97%
2	224x224	90%	96%
3	96x96	0%	0%
3	160x160	84%	95%
3	224x224	85%	90%
4	96x96	0%	0%
4	160x160	37%	38%
4	224x224	67%	93%

Table 4.1: Detection accuracy at different input sizes and distance

5 Analysis

The data displayed in chapter 4 shows how YOLO and SSD performs at different network input sizes. It shows that by using smaller input sizes, speed is gained at the cost of accuracy. In Figure 4.1 and 4.2 we see that SSD outperforms YOLO in speed and Table 4.1 shows that SSD also achieves same or higher accuracy in 11 of 12 instances.

In Figure 4.1 we see at what rate the models could operate at. Using 96x96 as input size, the SSD model reached as high as 4.42 frames per second while YOLO only achieved 2.72, which just a tiny bit faster than SSD at 160x160. 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 analyzing Table 4.1 we see how object size impact the models ability to detect. As the person gets farther away it becomes smaller in the image and the detection gets weaker. With 224x224 as input size, both models could detect the person robustly at all distances, only YOLO performed slightly worse at 4 meters. Both models also did well using 160x160 as input size. The person was detected with high accuracy at 1 to 3 meters 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 detect at 1 and 2 meters but already at 3 meters the person was to small to be detected.

6 Discussion

The results from the experiments show that running object detection on low end CPU devices is very slow. They also show that we can increase the speed by lowering input sizes at the cost of accuracy. The purpose of this thesis was to find out if a Raspberry Pi is suitable to use as hardware in a real time object detection system. Before answering this, we need to realize that different applications have different requirements in speed and accuracy. In one instance you might need fast detection but don't care about small or distant objects. In another situation, speed might be less important but better detection is. When implementing an object detector on a low end device, this speed and accuracy trade-off most likely must be done and the results shown in chapter 4 can be of help in selecting appropriate model and input size.

The object used in the experiments was a person, they are generally large and easy to detect even at distance. The reason a person was chosen instead of some random object was to see how well the detector can perform in a practical application such as surveillance. For this use, high speed is not essential so SSD with input sizes of 224x224 and 160x160 would both work well for that purpose.

One thing to note is that the models used in the experiment are trained on larger images. The SSD model was trained on 300x300 images and the YOLO model on 416x416. For best results, you would have the input size to be close the training size. This might have been why YOLO performed worse than SSD in accuracy.

In the article by Huang et. al. mentioned in section 1.2, [5] the authors showed how input size affects detection accuracy. The results corresponds to theirs as the SSD model still performed well on large objects at lower input resolution while struggling to detect smaller objects.

7 Conclusion

The aim of this thesis was to investigate the suitability of running a real time object detection system on a Raspberry Pi. Two models, 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 doesn't require high speed would it be viable to use the Raspberry Pi as hardware.

There is a trade-off with accuracy to be made if higher speeds are to be achieved since there is not enough computational power to have both. This lead to the conclusion that it is important to choose a proper input size to obtain the right balance of speed and accuracy that is needed for a particular application. This study could be of help for others that are looking to implement object detection on similar hardware to find that balance.

7.1 Future work

Due to time constraints there was a lot of things left out that could have been done to strengthen the results of this study such as more testing of different objects, distances and input sizes. One thing that would be interesting is to train own models on different datasets with lower resolutions and see if it could improve accuracy for smaller objects. Another thing that was left out in this project was looking at the impact that lightning has on a models ability to detect objects, this is something that could be explored in future work.

References

[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]. Available: https://www.blog.google/outreach-initiatives/accessibility/lookout-discover-your-surroundings-help-ai/. [Accessed: 05- Sep- 2019].

[11] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama and T. Darrell, "Caffe", *Proceedings of the ACM International Conference on Multimedia - MM '14*, 2014.

[12] J. Redmon, "Darknet: Open Source Neural Networks in C", *Pjreddie.com*, 2013-2016. [Online]. Available: https://pjreddie.com/darknet/. [Accessed: 22- Apr- 2019]

[13] Abadi, M. et al., "Tensorflow: A system for large-scale machine learning," in [12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)], 265–283 (2016).

[14] S. Mallick, "CPU Performance Comparison of OpenCV and other Deep Learning frameworks | Learn OpenCV", *Learnopencv.com*, 2019. [Online]. Available: https://www.learnopencv.com/cpu-performance-comparison-of-opencv-and-other-deep-learning-frameworks/#object-detection. [Accessed: 08- Sep- 2019].

[15] GitHub. (2019). *Tensorflow detection model zoo*. [online] Available at: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md#coco-trained-models [Accessed 20 May 2019].

[16] Redmon, J. (2019). YOLO: Real-Time Object Detection. [online] Pjreddie.com. Available at: https://pjreddie.com/darknet/yolo/ [Accessed 20 May 2019].

[17] *Raw.githubusercontent.com*, 2019. [Online]. Available: https://raw.githubusercontent.com/adagun/detector/master/models/ssdlite_mobilenet_v2.pbtxt. [Accessed: 08- Sep- 2019].