Seeing the Trees from the Forest:

Using Modern Methods to Identify Individual Objects in a Cluttered Environment for Robots

By

Josh Prow

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the requirements for the degree of
Masters of Science, Electronics and Computer Systems Engineering

Victoria University of Wellington
Supervisor: Will Browne

2019

ABSTRACT

Robotics and computer vision are areas of high growth across both industry and personal usage environments. Robots in industrial situations have been used to work in environments that are hazardous for humans or to perform basic tasks that require fine detail beyond that which human operators can reliably perform. These robotic solutions require a variety of sensors and cameras to navigate and identify objects within their working environment, as well as software and intelligent detection systems. These solutions generally require high definition depth cameras, laser range finders and computer vision algorithms, which are both expensive and require expensive graphics processors to run practically.

This thesis explores the option of a low-cost computer vision enabled robotic solution, which can operate within a forestry environment. Starting with the accuracy of camera technologies, testing two of the main cameras available for robotic vision, and demonstrating the benefits of the RealSense D435 by Intel over the Kinect for X-Box One. Followed by testing common object detection and recognition algorithms on different devices; considering the advantages and weaknesses of the determined models for the intended purpose of forestry.

These tests support other research on finding that the MobileNet Single Shot Detector has the fastest recognition speeds with accurate precision, however, it struggles where multiple objects were present, or the background was complex. In comparison, the Mask R-CNN had high accuracy and was able to identify objects consistently even with large numbers overlaid within a single frame.

A combined method based on the Faster R-CNN architecture with a MobileNet backbone and masking layers is proposed, developed and tested based on these findings. This method utilized the feature extraction and object detection abilities of the faster MobileNet in place of the traditionally ResNet based feature proposal networks, while still capitalizing on the benefits of the region of interest (ROI) align and masking from the Mask R-CNN architecture.

The results from this model did not meet the criteria required to recommend the model as an operational solution for the forestry environment. However, they do show that the model has higher performance and average precision than other models with similar frame rates on the non-CUDA enabled testing device. Demonstrating the technology and methodology has the potential to be the basis for a future solution to the problem of balancing accuracy and performance on a low performance or non GPU-enabled robotic unit.

CONTENTS

1 INTRODUCTION

1.1 CONTEXT

Computer vision is a rapidly growing field utilized in robotics, Internet of Things (IoT), and artificial intelligence, with increasing value to the everyday person. Despite all the research currently invested in this topic, computer vision still struggles with many tasks that, as humans, we do naturally, such as the ability to separate items from each other even in seemingly homogenous environments. Furthermore, high accuracy computer vision methodologies rely on graphical processing units, designed to run these large models. These processors and the kind of computational devices that house them are expensive and can be bulky due to the increased cooling and space required to fit them.

With these developments however, the field of computer vision and machine learning in general has grown to become an important field of future interest. Since the release of AlexNet in 2012, which marked a significant increase in the performance (accuracy and precision) of modern networks, interest has been growing across many industries for machine learning technologies to be integrated into their systems. The New Zealand government is one such interested party, with many intended applications for such technologies.

There are applications for robotics in virtually every industry, from public transport and self-driving vehicles to agriculture and forestry. While this project focuses heavily on the task of navigation through a forestry type environment, ignoring the alternative applications of the technology, or worse, limiting the potential applications of it, would be a mistake. The aim for this project therefore is not limited to the task of identifying trees in the forest, but to create a solution which can provide a basis for computer vision based robotic navigation in a variety of environments and situations.

In forestry the benefits of this technology are due to the inherent dangers that come from working in the industry, and the large task of monitoring and maintaining several hectares of forestry. The applications of this are therefore the collection of environmental information and the monitoring of the forest, and the completion of simple maintenance tasks, such as the clearing of accessways. In precision agriculture this same technology could be easy adapted for the identification of fruit on the trees or bushes, and thus used in aiding in yield predictions and financial planning.

This project is connected to a larger project driven by the New Zealand Government and other industry partners through the SFTI (Science for Technological Innovation) challenge, which aims to "develop world-leading science and technology relevant to New Zealand" [1]. The intention for this project is to be the basis for a robotic implementation for forestry environments. In New Zealand forestry is a wide spread industry covering 29% (78,000 km^2) of country's land area [2], all of which needs to be regularly monitored and surveyed.

Forestry is a difficult environment for computer vision to operate within. Unlike the well-defined and filtered data commonly used to test and train computer vision models, forestry is often made up of a single tree type grown in closely packed rows. In New Zealand these plantations can be as large as 190,000ha and are comprised of predominantly radiata pine (about 90%). Furthermore, the trees grown in New Zealand have been genetically modified to have the preferable features of a single tall trunk with even branch growth. This allows the trees to be closely spaced (distance), and with little variance between individual trees [3].

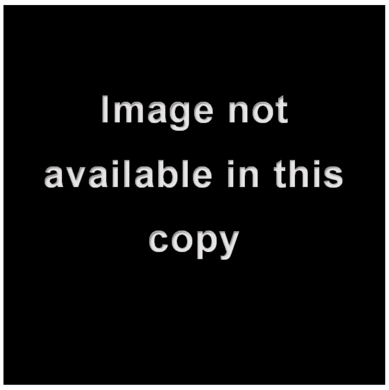


Figure 1 Identifying individual trees in the 190,000ha Kāingaroa Forest is a task beyond modern computer vision algorithms [4]

In February 2018 the New Zealand Government stated their intent to plant a billion trees by 2027, however beyond planting, these trees will need maintenance and care which has resulted in continued government funding beyond planting. This has resulted in a total investment of $6.5 million, however finding workers and trained professionals to supervise and facilitate this process is becoming an apparent issue [5, 6, 7]. This is where automated robotic systems could help, with either Unmanned Aerial Vehicles (UAVs) or Unmanned Ground Vehicles (UGVs) surveying, or even performing essential management tasks, such as pruning, reducing the need for man power and rare expertise [5].

Recent deep learning techniques developed in recent years are relying on CUDA enabled graphics cards with high clock speeds, and other advanced hardware to achieve the best results. This has made applying computer vision to everyday issues in both consumer and industry purposes possible but with a high price for the required computational hardware. This is evident in the recent growth in segmentation and masking technologies, which have the ability to visually and logically separate objects from a scene even where overlapping and non-geometric shapes occur.

The Mask Region Convolutional Neural Network (Mask C-RNN) is an example of such a system (see Section 2.5.2), and its masking methods have been demonstrated to have high accuracy when identifying multiple objects in a frame. Due to its complexity however, this network has yet to be implemented in an effective live environment, instead being used for pre-recorded media (images and video). The implementation of this technology into Simultaneous Localisation and Mapping (SLAM) or other robotic navigation and control processes should enable higher accuracy with robotic interaction with the physical world. Furthermore, configuring it to run on a non-GPU enabled device would provide an example for running higher complexity and accuracy models on the limited hardware these units tend to utilize.

1.2 The Problem

To facilitate the usage of automatic vehicles and robotic units within this type of environment a solution for rapid detection of obstacles and environmental features is essential. The major problem stymying the emergence of real-time advanced agricultural technologies is the required complexity of the hardware and the cost of it. While modern models can theoretically identify and segregate objects with a running time of 5 frames per second (FPS), this requires a minimum of one CUDA enabled graphical processing unit and sufficient computational memory to load the image and model simultaneously [8]. The need to fit all this hardware into a small robotic unit is being researched and solutions such as the Jetson by Nvidia who are trying to solve this by creating embedded computing devices with a GPU styled ARM processor designed for machine learning problems [9].

The second issue with this situation is the homogenous nature of the forestry, which can have low definition between separate items and large amounts of overlap between identifiable objects in a frame. This environment is not optimal for machine learning algorithms, with object separation using masks only recently becoming a trend, previously relying on bounding boxes which are ineffective at accurately defining object borders when there is an overlap between objects, or the objects are irregular in shape. This is exacerbated by the conditions of a forestry environment, with inconsistent lighting, and the potential for obscuring weather effects; features not commonly found in a cleaned and labelled test dataset.

The cost of this solution is an important limitation and ensuring that any device is capable of functioning effectively while being a cost-effective alternative is a challenge for all robotics-based solutions. While many robotic solutions that can navigate in a multitude of complex of environment do exist, these are often expensive and rely on high cost sensors such as LIDAR (see section 2.4.2) or advanced computational units which increase costs. While this cost is balanced against the reduction in staff wages (placed at between $40,000 and $75,000 based on experience by CareersNZ [6]) and other costs related with processing and staff training can be reduced, the cost for a single device must not be significantly higher than the cost of a human worker doing the same task (or set of tasks) over the expected operational lifetime of the unit (while also considering the benefits of a work force that takes no holidays and can function in potentially dangerous environments without risk of death or serious injury). Furthermore, if the cost of the device is too high at the outset, then it could be impossible to finance what could otherwise be a beneficial investment.

1.3 Aims

The aim of this work is to create a machine learning solution able to identify objects in a cluttered and homogenous environment. This solution must be able to run on affordable hardware, identified in this project, while still providing fast and accurate information about the working environment.

This project does not intend to create a complex machine learning algorithm for identifying features within the detected objects, nor does it include the creation of a physical robotic unit. Instead the intention is to create the basis for a complex solution that can better suit the exacting requirements of the industry, and thus can function as the computational core for a robotic unit.

1.4 Objectives

There are two different streams of research, coming together to create the final solution. These are the hardware and the machine learning models. For the hardware, the project seeks to determine the best sensor for this type of project, while the ML models are the software basis for the project. A constraint set by the SFTI project is the need to create a working solution without the need for a high-performance, high cost, device, such that it can be implemented into the industry with the minimum amount of financial investment to the forestry management companies.

The hardware decision process is important to create a solution that can accurately accommodate the limitations of a small (<1m^3) robot as described in section 2.3.2, while still performing its required tasks effectively. This requires the consideration of computational units that could be used in this project, to accurately represent the robot's computational unit. After deciding upon the computational unit, a supported camera or other sensory array is required to collect visual data as well as accurately measure distance to an object in frame, based on the speed of the robot (see Section 2.3.2).

The software focus of this project is the development of an accurate machine learning algorithm that can correctly identify trees in a forest at a practical rate of frames per second. Practical being based on the operational range of the depth sensor and expected speed of the robotic unit (See Section 2.3.2). The model is to be based on a combination of a simple model designed for a lower performance device, such as a smart phone, with no GPU, or other advanced chipset for such purposes, and the new Mask R-CNN technology which aims to provide object segmentation.

Objectives:

1. Identify appropriate hardware including computational device, and sensors required for operating within a dynamic environment, in accordance with the hardware limitations set by SFTI and logical robot design (Section 2.3.2).
    a. Decide upon a computational unit based on cost, potential performance, and the intentions of this project.
    b. Decide upon a series of tests that can ensure the sensors fit the requirements of the project, and the environment the robot will be operating within.
    c. Test a range of cameras commonly used in robotics with the criteria and measurements set in 1.b also noting the availability on cost of the device.
    d. Measure the benefits of the cameras choosing the best performing option, ensuring they are compatible with the decided computational unit – creating a single hardware solution.
2. Identify and test commonly used and preferred Neural Network models to identify features and models, which are likely to assist in the creation of an effective solution.
    a. Identify important and commonly used models with high performance on a range of devices – focussing primarily on models designed to run on minimal processing, or without GPUs.
    b. Test strengths and weaknesses of these identified methods using publicly available frameworks and datasets to analyse these solutions in a fair and verifiable manner.

    c. Evaluate the benefits of the Mask R-CNN in comparison to other R-CNN based methods, using the same frameworks and datasets to confirm the benefit of object segmentation.
    d. Test the best performing models of each category with a custom database to evaluate performance in situations where large numbers of similar objects are present.
3. Final Solution proposal
    a. Create a hybrid model which utilizes the key features of object segmentation with the faster object detection and classification methods of the faster models.
    b. Propose implementation in a live environment with system requirements and future research where required.

1.5 Thesis

This project aims to show that using the masking layers from the new mask regional convolutional neural networks, with current models designed for mobile and other small devices, can increase performance of object detection and recognition in cluttered environments in mobile robotic units and other devices with limited graphical processing units.

1.6 Document Organisation

This chapter has introduced the current problems both from the SFTI industry project and in related academic works, which have motivated this project. It has also provided the objectives and thesis, which guide the project. The next chapter, Background, discusses modern technologies and related works, considering the merits and drawbacks that could influence the results and direction of this endeavour.

The midsection of this document, Chapters 3 and 4, cover the experiments and studies required for investigating the thesis. Chapter 3 discusses the design of the robotic solution and tests used to analyse the hardware and neural network options, while Chapter 4 discusses the proposed solution showing results from tests and discussing the limitations based on the available resources.

Finally, this document concludes by discussing the merits and short-comings of the proposed solution and drawing a conclusion as to the accuracy of the thesis, presenting future works and research avenues that could facilitate the ready availability of this technology.

2 BACKGROUND

Creating faster and more accurate computer vision models has been the goal of many researchers aiming to facilitate the integration of computer vision-based systems with the modern, dynamic, world. Balancing the need for accuracy and the speed with which the models can make predictions is an important feature and consideration.

With faster training times and higher accuracy for image-based tasks than other popular methodologies, Deep Neural Networks (DNNs) have become the preferred method for computer vision. This was largely made possible due to the increase in hardware capabilities, a problem that had been stymying the progress of the technology, alongside the inability to collect the large quantities of data essential for accurately training a complex model [10]. Since their rise to popularity, there have been many improvements to the DNN technology, each new model introducing a new element, or refining a current one to hone the technology towards the ideal models, which can be trained to quickly, and accurately identify and separate objects of interest from a 'live environment' (being an environment where both organic and in-organic actions occur, i.e., outside of a managed test environment).

As well as making large and complex algorithms focused on high precision, there have been many models designed for faster processing or usage on devices with less processing capabilities. This includes models such as the MobileNet [11] and You Only Look Once (YOLO) [12] detectors, which aim to perform accurate image recognition 'on the fly' using mobile (or at least portable) devices. These algorithms are small and compact by design, reducing the amount of calculations and memory (RAM or Graphics) required for predictions, however this ofttimes translates into sacrificing accuracy for simplicity and speed [13].

Despite the research and new technologies that have emerged recently, the accuracy of these systems is limited, with even the best algorithms are only able to recognise a very limited collection of objects, and with accuracy which is often below that of a human (tested at a top-5 classification error of 5.1% in the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC)) [14].

2.1 Forestry and SFTI

Forestry is one of New Zealand's primary industries, covering over 6% of the total land area of the country. These plantations as large as 190,000ha and are comprised of predominantly radiata pine (about 90%) but also include some native and eucalypts [5]. The radiata pine is an imported species from California, which in the correct conditions, such as those found in many places in New Zealand, grows fast, with an average harvesting age of 28 years, in comparison to the 45-60 years which is more common for other forestry species [2]. By nature, the radiata pine tended to grow with many variate branches and a forked trunk, which created undesirable knots in the wood when harvested. To remedy this, genetic research was undertaken to reduce these features, and trees were planted with smaller spacing to encourage upward growth over the spreading of branches. These studies and practices shape the current industry, with the radiata pines grown in New Zealand's forestry planations still being closely spaced (1 tree per 10 square metres), and with little variance between individual trees due to the genetic research and experiments [3, 15].

2.2 Datasets and Measurement Criteria

2.2.1 Frames per Seconds (FPS)

An important measure for success in this project is the speed at which the system can process the data. A single still image from a video is referred to as a frame, and the number of frames per second or the framerate of a video is a common measurement of the quality of the video. This is due to the smoothness of motion that comes from a higher framerate, especially in situations where objects in the scene are moving at a high velocity.

In Robotics, the processing FPS of the navigation system is important for managing the speed and behaviour of the device. If the output framerate is too low for the speed of the robot, then the robot might not effectively detect objects in its path, especially if these are dynamic objects. This is discussed in greater detail in Section 2.3.2.

For the image recognition system, this measurement is used as one of the primary criteria for evaluating the performance of the network. As the goal of this project is to create a system which can accurately detect objects for the purpose of navigation, it is important that the model has both a high framerate and precision.

2.2.2 Datasets and Organisations

There are several organisations, competitions, and datasets which are related to the measurement of the accuracy of computer vision systems throughout their evolution. The primary datasets are the Pascal VOC [16] (Visual Object Classes), ImageNet [17], and the Microsoft COCO (Common Objects in Context) [18]. Each of these datasets has its own organisation managing the collection and storing of data, and as a part of this they also host competitions. These can be academic events structured with set conditions and time frames, or a rolling scoreboard of models tested with the organisation's toolsets. These groups provide an important standardised platform and measure for machine learning in computer vision. This is provided through toolsets and a dataset large enough to train a quality model on, with a variety of different features and objects.

The first of these, the Pascal VOC, started in 2005 as a method of evaluating and comparing different models, running annual competitions until 2012. This started as a database of 4 classes (motorbikes, bicycles, people, and cars), focussing more on the detection of an item in a frame, with a secondary goal of creating bounding boxes. With the progress in technology, both hardware and software, the level to which networks are tested has increased also. The most common standard for machine learning tests is now the COCO dataset which started in 2015. This dataset has 80 classes, ranging from potted plants, to elephants, with each image taken in a real setting.

By its last competition, the Pascal VOC had 20 classes across 11,530 images (a total of 27,450 annotated objects and 6,929 segmented). While the COCO dataset currently has over 200,000 labelled images and 80 classes, with human key points, annotated and segmented objects, and image captions. This greater range of annotation types and larger image collection shows the advance in the computer vision technology, where more data is both needed and available for training and producing working models in the modern world.

ImageNet is different from the other two solutions where it uses a large database organised into subsets where objects are recognised as in a "x is a y" type relationship i.e. (Pine is a Tree). This means that there are many depths of the ImageNet library, where objects could be recognised as "balls" or delved into deeper as "footballs, and "Rugby Balls". However, while there is much data, very little of it is annotated with only 3000 "synsets" (as they call their organisational hierarchies) being completed and on average 150 images for each. The annotations are in Pascal VOC format, and there is no toolset for the dataset. ImageNet did host competitions, the ILSVRC (ImageNet Large Scale Visual Recognition Competition), however this was based on a small subset of the data contained within the ImageNet database.

Despite the many datasets available, there is no forestry database or even tree section in any of the primary datasets. The Pascal VOC dataset includes the class ‘Tree’ however this is used primarily in the case of a singular tree in a setting – say an indoor office plant, or a tree on a hill. The COCO and ImageNet datasets however have no tree class or data. This means that for this project a new database will be required, and a separate testing environment with these models will be required to achieve a balanced test of the networks.

2.2.3 Measuring the Effectiveness of Object Detection and Classification

When measuring the results of a computer vision system there are several features which are used. Precision and Recall are the first two elements commonly used to describe the accuracy of a model. Both of these measurements are measured against the machine learning algorithm’s results as shown in a confusion matrix (Figure 2), showing the number of True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN). Precision describes the ‘correctness’ of the model, measuring the number of TP out of the number of predicted positives (FP + TP). Recall meanwhile describes the ‘accuracy’ of the model, measuring the number of TPs identified out of the total objects that should have been identified (TPs and FNs).

		Actual	
		Positive	Negative
Predicted	Positive	True Positives	False Positive
	Negative	False Positive	True Negative

Figure 2 Confusion matrix layout

Precision and recall are important functions for all classification type learners, but when working with object detection, the placement of bounding boxes becomes another feature which needs measuring, this is done using the IoU (Intersection over Union). The IoU measures how much of the prediction overlaps with the ground truth (set bounding boxes or masks included in the training data) given in the dataset, against the total area of both ground truth and prediction. This is presented in a binary true or false based on a predetermined percentage, often 50% or 0.5, this is shown as IoU@50 or IoU_{50} (depending on notation).

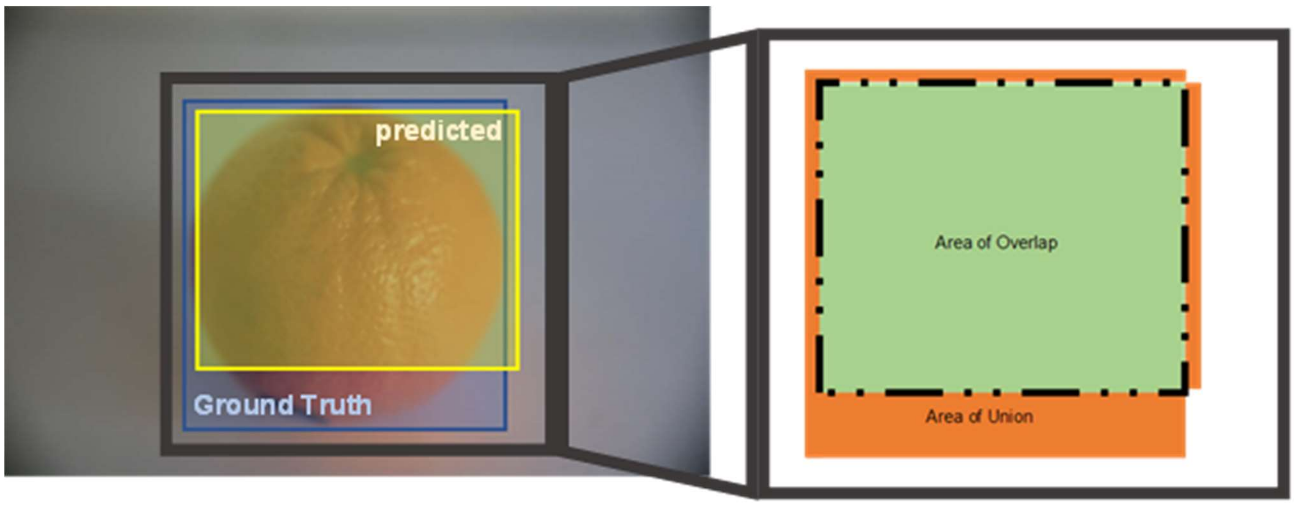


Figure 3 The IoU for the prediction of this orange

The most common measurement for comparing computer vision solutions is the mean Average Precision (mAP). This measurement provides an accurate measure of the abilities of a solution on a certain dataset, based on the precision at set IoU levels weighted by the order of predicted likeliness. For usage in object detection, this method benefits over the confusion matrix as it better represents situations where multiple objects can be detected in a single image. Where in basic classification tasks a prediction must be made for each object, in object detection there can be any number of objects, including none, so the number of possible false predictions is potentially infinite (limited by the model's configuration, if at all).

First average Precision is calculated, which means calculating recall and precision based on the number of true positives for all predictions of a class, with a prediction confidence equal or higher than the current prediction confidence value. In Table 1, the third prediction for example, has a precision of 0.67 as two out of three of the predictions have been correct, whereas the recall is 0.67 because of the 3 instances, only 2 were recognised (the third never being recognised in all predictions). These results are then graphed against the recall, smoothed such that the highest value of precision is mapped against the recall as shown with the blue line in Figure 4 Average precision graph with original line (orange) and smoothed line (blue).

Average precision is the number under the graph, generally calculated with an 11-point interpolated method. The average precision is the value sum of the precision at each of the eleven equally spaced points (0, 0.1, 0.2 … 1), divided by eleven. Thus, in this example; $\frac{4\text{x}1 + 3\text{x}0.66 + 4\text{x}0}{11} = 0.55$.

Table 1 Average Precision for a theoretical model, having made 4 predictions, on an image with 3 objects to be detected

Rank	Prediction Confidence (%)	Correct	Precision	Recall
1	96	True	1.0	0.33
2	95	False	0.5	0.33
3	72	True	0.67	0.67
4	26	False	0.5	0.67

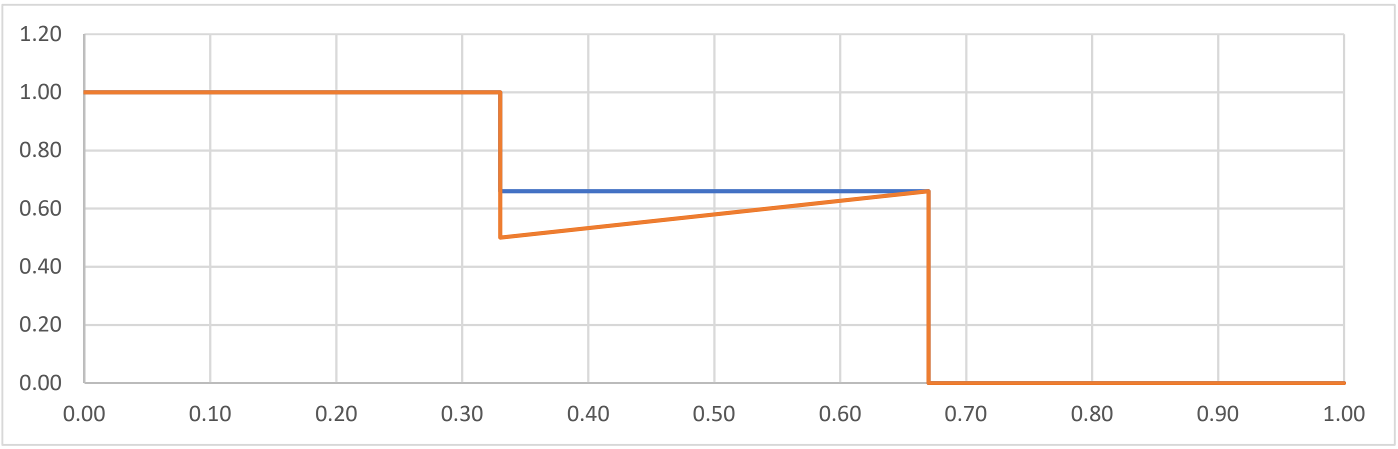


Figure 4 Average precision graph with original line (orange) and smoothed line (blue)

The COCO database however uses a different slightly different definition of mean average precision, which also includes the average across multiple IoUs starting at 0.5 and increasing in increments of 0.05 (up to 0.95). This is done to give weight to models that provide greater accuracy when bounding objects.

The final common measurement is 'top-5' or 'top-1', meaning that the correct result is within the first 5 predictions, or is the first prediction. This is important in situations where there are many classes, especially where different objects are similar or could get confused. This can also be measured in the inverse, where the top-[number] error describes the percentage of times the classifier did not predict the correct class within that number of guesses.

2.3 Hardware

2.3.1 Microcontrollers and Computers

2.3.1.1 *Computational Devices*

There are several types of computational units available for consideration for modern robotic researches, these include conventional computers and PC devices, microcontrollers, and embedded computing boards or single-board computers.

Personal computers (PCs), including both laptops and desktops have the best processing performance (maximum processing speed and complexity of supported calculations) of the commonly accessible computational devices, and are very accessible. However, they are expensive (approx. $1000 for a mid-tier laptop, $750 for a mid-tier desktop) and relatively bulky (2 – 7kg), and although varying dimensions, they are not designed for the purpose of mounting on robotic devices. These devices are often used for testing and prototyping as they can be used for the whole development process without the need to export and compile code for different environments. As well as for ease of debugging as coding on the test device is significantly easier.

Single-board Microcontrollers (SBMs), such as the Arduino boards, as well as other microcontrollers are effective for robotics and other IoT (Internet of Things) based applications due to their small size, low cost (< $100), and simplified system architecture. However, single task nature and small hardware specifications make them very limited for processing complex tasks such as computer vision. Jetson boards are a relatively new (2014) solution to the SBMs' low processing problem, using small chipset size hardware based on Nvidia's GPU technology. This allows them to support CUDA applications and makes them effective at running mathematical processes and simple machine learning applications, for the same reasons as found in GPUs (see below; CUDA and Nvidia). Despite this specialised design, laptops and PCs still have the potential to (and often do) out-perform them for processing speed, due to the larger GPU space available.

Single board computers (SBCs) are becoming popular tools for IoT base projects. These, like SBMs, have a smaller form than PC solutions, however benefit from a multi-threaded core. As such they are capable of running full operating systems such as Raspbian – a custom version of Linux, designed for the Raspberry Pi single board computer. Unfortunately, they still suffer from the requirements of their small size, with limited speed processors and memory space (both RAM, and hard drive) which affects their performance for complex computer vision models.

2.3.1.2 *CUDA and Nvidia*

When working with machine learning, as well as other processes requiring large amounts of numerical processing, programming processes to run directly on the Graphics Processing Unit can provide a much higher performance. This is due to the specialised nature of the microprocessor, which in place of a few (up to 24) complex cores, such as those found in the CPU (Central Processing Unit), is composed of many smaller processors. These processors handle only simple instruction sets, however due to the number of cores, and the abilities of parallel processing, they can perform tasks which require multiple simple processes, or that can be broken down into several small tasks which can run in parallel.

When working with machine learning and big data, GPU processing is an essential tool, reducing training times significantly (as much as 100x faster). When running machine learning models, the GPU allows for faster image manipulation and pre-processing, as well as faster mathematical computation when running the image through the network for predictions.

Using a GPU for programming requires a programming platform to operate optimally. While most modern GPUs can run code written in C based languages (C++ / C) with special libraries, to increase performance and ease of development, Nvidia developed GP-GPU (General Purpose GPU) technology and CUDA, a platform and programming language for creating GPU-accelerated programs. CUDA is not the only solution for programming on GPU devices, with APIs like Direct3D and OpenGL allowing for graphics programming. These APIs[1] however, require advanced programming skills, and specialisation in their usages. In comparison, the CUDA platform provides support for popular programming languages like Python and C++, as well as machine learning and mathematical programming languages like Fortran and MATLAB.

CUDA is a proprietary platform by Nvidia and only supports their graphics cards, which makes them the first choice for any solutions requiring, or that would benefit from, GPU acceleration. This means that a computer using a non-Nvidia graphics card like AMD cannot run CUDA, and as such is sub-optimal for any machine learning process.

[1] Application Programming Interface

2.3.1.3 *Intel NUC*

The Intel NUC is a compact desktop computer or ‘mini pc’ designed for usage in a multitude of situations where a full-sized tower is inconvenient, but a full operating system is required or of benefit. In the case of this project there are three major benefits. First is the availability and price of this device, the NUC used is cheaper than modern Jetson models by 15%. Furthermore, due to their small and re-usable nature several were available for testing without further expense from previous projects requiring small processing units. The Jetson, however, is an integrated board and as such it is often integrated into a single robotics solution and thus cannot be moved between projects as needed.

The second is, as mentioned in the previous section, the ease of programming and debugging on a computer or laptop. While it is possible to reprogram and edit SMBs, and SMCs do technically run full operating systems, the ability to run the whole process from programming to testing within a single Linux environment with complete IDEs is a massive assistance to the process. It allows rapid testing of code, and effective result and error logging within a contained environment, which better helps the understanding of issues and features which are important to the tests and technologies being used.

Finally, this project includes the desire to create a reusable methodology and solution which can be used in a multitude of situations. And, as on average, computational units for robotic devices do not have graphic processing units, creating a solution which relies on CUDA technology is limiting to the hardware and application of the methodology, forcing developers and engineers to include Nvidia computational units into their designs to support the technology. Therefore, the basic NUC model’s lack of GPU beyond the included emulated driver in the Intel ‘i’ series CPU is an advantage; better allowing it to demonstrate the limitations of an embedded robotic controller (although it is not a perfect comparison as single embedded processor will almost certainly be inferior in processing capabilities due to requirements in size, cost, and technologies.

The NUC by Intel is a compromise between the need for an effective example device, showing the features of a final solution, and a reusable testing device which can be quickly reconfigured for a range of tasks. It runs full desktop hardware and operating systems while only having the basic specifications for completing its tasks in a small frame. Using a NUC in a final robotic solution is not a recommended solution and an SBC likely will achieve similar results for less financial expense, as well as be better fitted to the size constraints. However, as a demonstration device, the convenience of the NUC makes it a clear choice.

2.3.2 SFTI Robot Design

This project is focussed on researching and proposing options for the computer vision element of the SFTI robotics solution, however in tandem to this is a physical robotic design project. This is being created and tested by James Barrett at Victoria University of Wellington, based on specifications set out by the SFTI industry partners, and his line of research. The estimated specifications of this device are as follows, although final results will be assessed upon the completion of that project.

Due to its working environment, the robot is required to fit and traverse narrow paths at high inclines. According to the New Zealand Forest Road Engineering Manual published by the NZ Forest Owners Association Inc. [19] the smallest tracks in forestry should be an “establishment track” which is a low speed ‘road’ with a width of about 2.5m and a maximum gradient of 20%. However, as there may be situations which require the robot to leave the standard vehicular path, the unit must be able to fit between closely planted trees (as close as 3m of trunk separation) and easily navigate between and around obstacles. Due to this, the recommended size of the robot is no wider than 1m, so to easily fit between the planted trees even with low branches untrimmed, and also to allow navigational options around terrestrial obstructions (rocks, surface roots, etc.).

An important feature, with significant effect on the physical and software design of the robot, is the speed at which it traverses the environment. This is particularly important as many success criteria are based on this. Primarily among them being the frame rates of the depth sensor, camera, object and detection system. A study in 2011 identified the average human walking pace as 1.25 metres per second [20], which will be the basis for a reasonable speed for the robotic unit. This means that with a frame rate of 5 FPS the system needs to accurately detect a stationary object 0.25m away. In a dynamic environment it must be able to accurately detect significant objects that are 0.5m away, as well as manage to identify objects approaching on intersecting angles.

The length of the robot should also be limited to fit in the environment. However, the turning arc is more important with a recommended maximum radius of 1.5m; this should allow the robot to navigate between the trees should such a manoeuvre be required. Achieving a tight turning arc will require the robot to not be too long or require the implementation of alternate solutions in which case a maximum length of 1m should be implemented for the same reasons as the maximum width set in the previous paragraph.

A significant limiting factor in the size of the robotic platform and materials used in the creation of it, is the weight. This is important as the unit needs to be easily translocated between different plantations and furthermore in the case of failure may need to be moved manually by human operators. As such, the device should be light enough, to facilitate collection by a single human user in optimal situations or two in cases where complex lifting, or manipulation is required. Therefore, a recommended maximum weight of 20kg is set for this solution.

The remaining features are the cost of the device and the conditions under which it can operate. These are soft features which need to be analysed during the design and development phases of the project. Ultimately keeping the cost low and allowing the device to work in all environments and conditions is preferred, however it is unlikely that a low-cost camera option will be able to function in adverse conditions, and thus a benefit analysis should be made for the available technologies.

2.4 Camera Hardware and Technology

2.4.1 Data Formats

There are many different formats for storing and working with images and other sensory data. Methods can differ in how they store and represent the data, as well as compression techniques utilized in the collection and storage processes. When using multi-sensory devices this expands to include methods of merging the data formats such that higher context might be gleaned.

The data collected on a camera with depth sensors will naturally include two separate frames from the two sensors. In stereo vision type devices like the ZED (See Section 2.4.8.5), both images are colour while with IR based cameras one image is a standard RGB colour image, while the other will be a depth stream of values. In either case, these frames will have different points of origin, and thus a slightly different image will be captured, and furthermore the lenses are likely not the same quality or shape, which can cause the FOV or even zoom factor to not match up either, making it difficult to combine the two data streams into a combined format.

With devices like the D435 (See section 2.4.8.3) solving this is made significantly easier by the included SDK, which has inbuilt tools for ‘aligning’ the video outputs. However, these methods compensate for, rather than solve the problem. With stereovision type cameras, this is less of an issue, as the lenses should be reasonably similar – with differences being imperfections and damages rather than differences in the base technologies. They also benefit as images are individually showing the same information and depth context can be calculated using parallax-based algorithms.

With an IR sensor however the situation is different, as the IR data collected is often without reliable context without the supporting colour image, and as the two collected data “images” show different data types, aligning them is based on fixed values and calculations. However, doing this causes ‘shadowing’ where objects are obscured from the depth camera, but visible to the colour (meaning that depth cannot be aligned), or mis-alignments where objects measured by the depth sensor are obfuscated on the colour, so the obfuscating object is given the depth of the detected object instead.

Instead of storing the data as two separate files it is possible to convert them into a single file type. The primary formats being RGB-D, point clouds, and voxels (see below sections). This adds increased contextual data to the information being collected. However, depending on the file type, it can also require more processing to read, write, and represent in a human-viewable way.

2.4.1.1 *RGB-D*

RGB-D is a 4 channel image data input, which requires the depth image and colour image to be of equivalent resolution, or to be scaled as such. This process saves the Depth data as a channel within the colour data’s Red, Green, and Blue channels. This method is very simple and can quickly provide depth data along with a colour image in a form that a machine learning algorithm can process, making it advantageous for time-based processing. It is, however, limited in the quality and versatility of the information it provides. By mixing the depth data as a channel it becomes difficult to visualise and separate objects from one another without parsing the data directly to extract the depth values.

2.4.1.2 *Point Clouds*

In point clouds, depth data is saved as points within a 3D space, these points can then be mapped and overlaid with the colour data. This provides for a very clear image of the environment, and where objects sit within it. However, the data is not evenly distributed, and any colour data that does not have a corresponding depth point (because they were out of range, or other cause) is lost. This system is well suited for separating objects in 3D space, but because the points are unconnected analysing material and colour is less effective.

2.4.1.3 *Voxels*

Voxels are a method of creating 3D digital images, by placing 3D ‘pixels’ in an environment, the name voxel coming from “volumetric pixel”. Voxels allow for the easy representation and visual separation of 3D objects in a drawn space, while also allowing better retention of colour data than point cloud data. Voxels however, suffer from over simplification as they try to represent the real world in only cubes, which, due to size, are generally low in resolution (a box voxel of resolution 100x100x100 cubes is the same size as an image at 1000 pixels, which is approximately the largest size image commonly used for machine learning due to memory and pre-processing limitations).

Another issue is lack of standardisation, as there is no recognized single format for voxel data storage. Often voxels are stored in .STL format, which is a structure for triangle based meshes. This format is suited for describing complex shapes and meshes. However, it is based around edges and faces, features which are irrelevant for the single tone, equally sized cubes which voxel data is composed of. Effectively using this format requires a transformation to reduce the size of the data, as each pixel would otherwise be converted into its own separately packaged object comprised of 8 points and 12 faces (as faces are triangular). This would need to be later reversed to get the tensor format which would be expected in a machine learning application. While it is possible to convert the image directly from the input source to the 3D tensor format, this makes human visualization of the process very difficult as tensors are very poor for human reading and analysis.

2.4.2 Active Sensors

The term active sensor is used for any sensor which uses an external broadcast to operate. This is commonly based on the idea of sending a signal and measuring the time required for the signal to be returned in the form of an ‘echo’. This technology is well established in many fields; however, they all suffer from issues caused by interfering signals, other devices and technologies broadcasting a similar signal can have an effect on the measurements collected by these devices. The exception to this are structured light based systems, which, in place of measuring the time of reflection, they project a known image and measure the distortions in this pattern to calculate depth. These are covered in section 2.4.4.

There are many different methods for sensing and visualizing an environment in a digital format, where each method has different requirements and strengths. The most common are radar, lidar, sonar, each using different wave types to measure distance – these are collectively known as “time of flight” sensors. **Ra**dio **d**etection **a**nd **r**anging (Radar), **li**ght **d**etection **a**nd **r**anging (Lidar), and **so**und **na**vigation **a**nd **r**anging (Sonar) all utilise an emitted signal and the resulting echoes to collect data on the working environment.

Radar is most common in aerial based situations, for both UAVs and aircraft detection and management, from both ground stations and the vehicles. Radar uses radio waves to detect objects in an area surrounding the sensor with a range dependent on the sensor size and type; air traffic control towers use large radar towers to detect aircraft over kilometres, whereas land based autonomous vehicles using radar for navigation tend to only have ranges of only a few metres, which the longest range being about 300m. Radar has a large range, and can penetrate many objects, allowing for detection of objects otherwise obfuscated. However, this is not possible if the object is conductive. The major drawbacks to this technology are a lack of contextual information as radar can only measure distance from the sensor, “smearing”, which is an increasing inaccuracy which causes blurring of data at range, and its high cost in smaller implementations. [21]

Sonar is very much the same as radar, except that it uses sound in the place of radio waves, this allows for greater range in denser environments and is cheaper when used in smaller devices and for smaller ranges. Ultrasonic and sonar technology is most commonly known for its application underwater, where the sound waves can travel for larger distances and with greater accuracy than other technologies. However, they are not limited to this one environment, with ultrasound being used for a vision aids for the visually impaired, and as a low-cost option for object detection on robotic platforms.

Ultrasound and sonic sensors suffer heavily from inaccuracy when working with certain shapes and materials which can render environments invisible or cause unexpected results – a curved wall for example may be able to reflect the sound in such a manner that the curve is imperceivable to the sensor – instead objects at the end of the curve are detected to be directly ahead.

Lidar uses fine lasers as opposed to waves and therefore is limited to line of sight only. This technology is the most expensive but is significantly higher in accuracy and fidelity. This is due to the fine nature of the laser beam and fixed behaviour of light, where sonar is affected by the shape of objects and how they reflect sound waves. The tight beam of light has significantly less dispersion and will better reflect a single point. Likewise, light reflects off most objects with even results, whereas both sound and radio waves can be disrupted by different materials and environmental interferences. The main disadvantage of lidar is operational requirements and its inability to detect through obfuscating objects, while radar and sonar can be used in most environments, high temperatures or water in the air can cause light to diffract and cause errors, especially over large distances. A similar technology, laser range finders (LRF) solve this using triangulation techniques to achieve a higher accuracy, but with a lower effective range and field of view.

2.4.3 Passive Sensors

The second option is using passive sensors, which capture information using the environment around them, this includes camera technologies, as well as microphones, thermometers, and other common place sensors. The biggest advantage of passive technologies is that they cause no interference with other devices, which is beneficial in environments where multiple devices or systems can be working within an environment. However, this makes their usage entirely dependent upon the environment, for example; a camera cannot collect colour video at night without a light source, whereas an active sensor can accurately capture 'images' regardless.

Cameras are the best example of multi-purpose passive sensors, using the atmospheric lighting conditions to capture and image the environment.

2.4.4 IR Depth Sensors and Structured Light

The biggest issue with using passive sensors is measuring distance, as a single image without the augmentation of some form of active known-emission sensor, does not contain enough information to measure the depth of object in the frame. To address this issue there are several solutions, including cameras such as the ZED (see chapter 2.3.7.5) which use the parallax effect between two lenses to calculate distance, or multi-frame object tracking, such as monocular SLAM [16]. However, these methods require processor consuming algorithms which can reduce the performance of the computer vision task, as well as other tasks the robot should be performing.

As such, a simple active sensor, for example a time-of-flight (active) sensor, is usually coupled with the camera. This active sensor, such as an infra-red (IR) camera which uses an IR emitter and sensor to measure depth, suffers from the disadvantages of all active sensors, but is generally designed to be low power, and intermittent with its broadcasting, meaning it is less likely to cause serious interference with other devices working within the environment. For example; the IR light often used by these depth sensors cannot pass through non-transparent objects, and when used in a large open area, the low power of the emitters is used to limit the range. Devices working in close proximity can also be synchronised to alternate their emitters and as such several such devices can be used in a single environment. Doing this however, requires the cameras to have a similar phase timer (when the emitter is broadcasting or not), which could be an issue with devices from different brands or even just different models.

Another method used by IR cameras for measuring depth is using a structured light process. This is done by projecting a known pattern usually of a grid pattern, bars, or a dot array, onto objects, then measuring the distortion when recorded from an IR camera (this can also be done with visible light, although, this is not a common case outside of demonstrations of the technology). This method allows for high accuracy of depth measurement across the entirety of the image's frame, at the expense of slightly increased processing time per captured depth image.

2.4.5 Camera Technology and Features

Modern digital cameras are based on very similar concepts to their analogue counter-parts using digital sensors in place of the photosensitive film. Because of this legacy, many terms & features of digital cameras are the same or based on the analogue version. Concepts such as 'field of view' and focal length are the same and controlled by the same lens technologies, while features such as sensor-size and resolution are new terms referring to the digital sensor & file output.

Field of view, focal length, and aperture are lens features; effected by the type design and feature of the camera's lens. The field of view (FOV) describes the angle of vision that can be captured by the camera, with a wide-angle lens or panoramic lens the FOV is increased such that a large area can be captured in a single image, however in doing so the lens causes distortion known as 'fish eye' on the image.

Focal length is the distance, measured in millimetres, from the exit of the lens to the camera sensor. This affects the field of view of a camera, with a shorter distance having a wider field of view. This is useful as the tighter the field of view the larger items seem when captured on a sensor of the same size (the sensor being the photosensitive element of the camera which 'captures' the image), thus providing a zoom feature.

The aperture is the opening which controls the amount of light allowed onto the sensor for a single image or still frame of video, thus influences brightness of the image. It also has an inverse effect on the depth of field, which determines the range of distance at which objects are held in focus. Aperture is measured in f/stops, with high f/stops such as f/4 meaning a wide aperture, thus a bright image but with low depth of field where as a low f/stop such as f/22 will be darker but with a large depth of field.

The counter part of this is the shutter speed, which measures in seconds how long the aperture is held open. Like the f/stop this controls the light allowed onto the sensor, but instead of negatively effecting the depth of field, it will result in a motion blur in any objects should they move during the time the aperture is open.

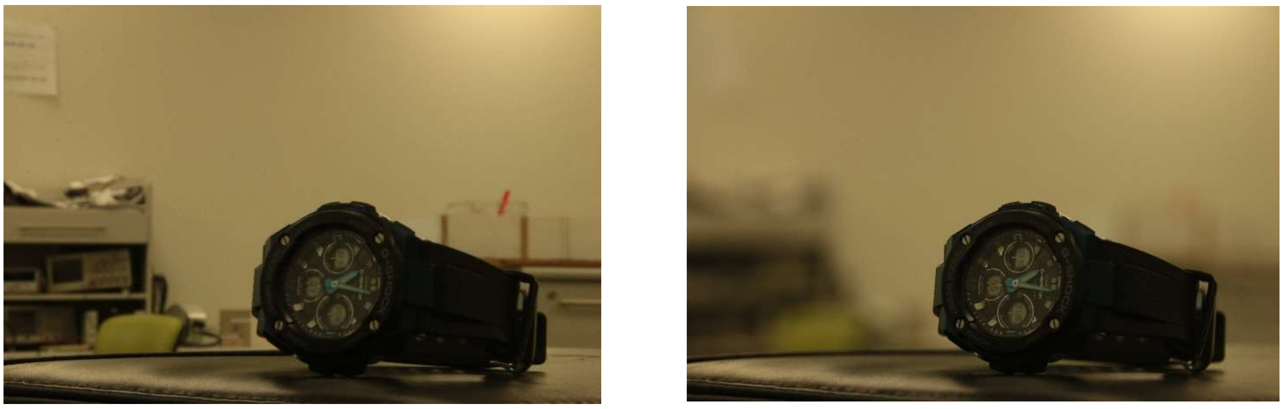


Figure 5 Comparing f-stops 32 on the left and 5.6 on the right. (Shutter speed has been changed to balance exposure)

When describing lenses, it is common to refer to them by their focal length (i.e. a 35mm lens) as this feature is usually immutable, or scales within a range (for zoom lenses, which are referred to by their range i.e. a 38 – 280mm lens). The focal length of a lens dictates the lens's general purpose; lenses with higher focal lengths having smaller FOVs, and make images in frame larger, so are better for long distance images, however they cannot focus on images in proximity. Whereas cameras with smaller focal distances have larger FOVs and as such can fit more or larger items at closer ranges into a frame.

When comparing cameras, it is a common misconception that the quality is measured in megapixels; that cameras with higher mega pixels can take better images. The megapixel value attributed to a camera is instead related to the resolution of the image and the number of pixels saved in the output file. The megapixel count is important when using the images on large or high-quality displays, or when they are being printed. When the image is being used for digital processing, low to medium quality digital displaying (e.g. website graphics), or other non-photographical-publication (such as posters, art, etc.) purposes, such as this project, the megapixel count is significantly less important than the other quality features.

2.4.7 Colour

There are also different methods for storing and representing colour data, apart from the ubiquitous RGB colour code used in digital purposes. The RGB paradigm is based on three colour channels (Red, Green, and Blue) which, when mixed as light on a black background, can form the multitude of colours used in modern computing.

However, this is not representative of how the human eye works which is the basis of the Hue, Saturation, Value (HSV) and Hue, Saturation, Lightness (HSL) colour systems shown below. The HSV system is based on the mixing of paints while HSL is more representative of natural light perception. Both are based on a cylinder with the extremis being the colours at saturation, however the HSV places white at the point where saturation is 0, but that value is at its highest, whereas HSL places white at the point where lightness is at its maximum, regardless of hue or saturation, although a saturation of 0 still provides a tone of grey.

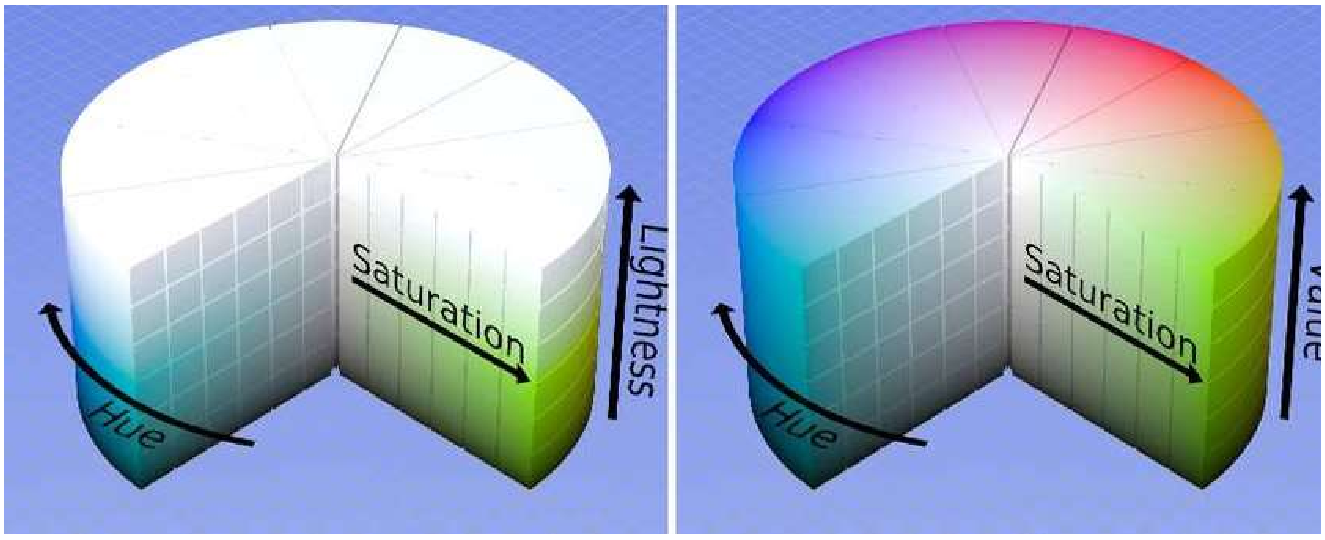


Figure 6 Comparison of the HSL (left) and HSV (right) colour spaces [22]

There are several other colour spaces, such as the analytically focussed YCbCr colour space, which are designed to represent the human eye's colour detection or to provide a more mathematical approach to colours. However, current methods for machine learning tend towards the RGB colour space that is popular for bitmap image files due to the availability of images, and the extra time required to reencode images. Relying on such a process may cause data loss in transformation, and the increase in pre-training processing time, especially when live cameras are in use. However, reasons for this preference have not been well documented, and the lack of research in this could be driven by the high accuracy already found using the RGB colour space as shown in the results of ImageNet and COCO competitions. Experimental results from this year

(2019), currently awaiting peer-reviewed publishing, have shown that the L*a*b*[2] colour space can perform well in machine learning, however that the SVM pretrained with its more common RGB colour space still outperformed the tested alternatives [23].

2.4.8 Commonly used RGB-D Cameras

There are many different camera and sensory devices currently available for usage in computer vision and robotic purposes. Over the years these devices have been used by researchers to collect data, drive autonomous vehicles, and collect 3D representations of real-world objects. Of the devices used however, the Xbox Kinect camera has been a favourite, with many researchers using it for their works [24, 25]. As such, it would follow that this is a leading device in the field. However, it is not alone, and recent changes in the environment have had Microsoft, the creator of the Kinect range stop producing the camera for research purposes, and instead start recommending the Intel RealSense product line [26].

The RealSense line is not unfamiliar to the academic and robotic communities with versions being used in several works since its release [27, 28]. The R200 series camera had similar specifications to the original Xbox Kinect camera, however benefitted from a significantly smaller body (only 101.6mm x 9.6mm x 3.8mm in comparison to the Xbox's 249mm x 66mm x 67mm) and a lower power requirement – running over USB by default, rather than a wall socket. The D435 and 415 are the newer models, theoretically performing above the Xbox One Kinect camera. Like the R200, these cameras are smaller and better suited for mounting on mobile units than the Kinect equivalent.

The ZED camera is different from the other cameras in that it uses a passive sensor only, instead of the active infrared used by the Kinect and RealSense cameras. To collect depth data, the ZED uses two HD colour cameras and the parallax effect to calculate the distance of an object in its field of vision. This benefits the technology as it can run in environments where IR lighting could be influenced or disrupted by interfering sources, however, hamstrings it in conditions where lighting is bad or heavily inconsistent. This technique also requires graphical processing, and thus a GPU, which can put a strain on a system relying already on a GPU for computer vision tasks or rule out the camera (as in this case) when a GPU is not available by design.

[2] The CIE l*a*b (Commission Internationale de l'Eclairage) colour space models colours with colour-opponent theory, that a colour cannot be red and green or yellow and blue at the same time. As such these colours are used as the measures creating the colour code. l* is the lightness, a* is the red/green component, and b* is the yellow/blue component.

2.4.8.1 *Intel RealSense*

The Intel RealSense range were first released in 2016 with a range of cameras which were primarily embedded in phone and laptops, with the exception of the R200. This device was competing with the Xbox Kinect as well as other robotically focussed solutions to provide a method for collecting image and depth data. RealSense has been used in several works in the fields of computer vision and robotics, however the higher price has made it a less popular option to the Xbox Kinects.

2.4.8.2 *R200*

The R200 is one of the earlier series of dual sensor cameras released by Intel in its RealSense series line. It is endorsed by Microsoft as the camera series which should replace the Kinect for Windows in computer vision applications, as seen in the redirect when one expresses interest in the Kinect for Windows range. The R200 is fully supported by many prebuilt frameworks including ROS, however it has been replaced by the newer series of cameras which have higher specifications and better supporting software. This means that getting the official drivers and SDK from Intel is not easy or possible (in the case of the SDK). There is however, a library librealsense [29](LRS) which is an old project by Intel which provides an interface with the device for C based languages, as well as including wrappers for other languages such as python.

2.4.8.3 *D435 & D415*

The D400 are the newest in the RealSense series, they have a similar camera array to the R200 with two IR cameras, an IR laser projector, and a colour sensor. However, the shutter and camera technology has been updated for higher quality. The D435 is the larger option of the 400 series, the main difference being the size and type of the sensors. The 415 uses a decrease Field of View and rolling shutter to provide higher accuracy images, both for depth and colour images, while the D435 uses a larger FOV and global shutter which collects more light, thus making it more effective in dark or inconsistent lighting but increases artifacting. Like the R200, the D435 and D415 have a small body size, with the larger of the two measuring only 90mm x 25mm x 25mm, which is beneficial when mounting the camera on a small platform.

The D400 series is the updated version of the R200 series and is supported with the RealSense2 SDK. The newer environment is designed to be more efficient and have more options than the original version of the SDK. Unfortunately, the new system is neither backwards nor forwards compatible (code for the R200 will not work on the D400 series, or vice-versa). When making a business decision this loss of support should be considered, as the robotic units will be expected to have a long lifetime.

2.4.8.4 *Xbox Kinect*

The Kinect camera was originally designed for the Xbox gaming console, as a competitive product to other physically interactive systems such as Nintendo's Wii Console and PlayStation's EyeToy and later Move camera & controllers. The Xbox system removed the need for a controller with IR emitter and/or accelerometers by using a dual camera array and IR 'Blaster' (array of IR emitters) which can be used to calculate depth of object within the field of view. The "for Windows" adaptor was released soon afterwards as it became apparent that the technology behind this camera could also be beneficial to computer visions tasks. This adapter consisted of a controller box which handled the device, USB cable, and wall socket for power (as the camera requires more than the 5V maximum provided over USB).

The Xbox One Kinect is the second major version of the camera, and uses a higher resolution camera array, and a new 3 light IR emission system which is touted to increase framerate and depth accuracy.

The Kinect has been a favourite for research into computer vision since its release for Windows (and subsequently Linux, with the correct SDK). Its high specifications, inbuilt depth sensor and IR emitter, and low cost, made it a great choice for SLAM (Simultaneous Localization and Mapping) or object detection projects.

2.4.8.5 *ZED*

The ZED is a 3D camera by Stereo Labs, designed for 3D depth sensing, motion tracking, and Simultaneous Localisation and Mapping (SLAM). It uses a twin colour camera array to measure depth without the need for IR. The benefits of this being that it is an emission free method, which means multiple devices will not cause issues, and cannot interfere with each other. However, it does mean it is totally dependent on environmental lighting, and suffers when lighting is low, or inconsistent.

The ZED has a very wide image format as both cameras are capable of 1080p recording. The 'main' image captured is in the area where the sensors' FOVs overlap. In this area the depth of objects can be measured, and due to the capturing methods extremely high-quality images are ensured. The technical framerate for each camera is 60FPS at WVGA (Wide VGA; 800 × 480px), however due to the ability to control the sensors' framerate separately, it can effectively capture video at 100FPS for the central section which is a resolution area just above WVGA at 800 x 480px.

2.4.8.6 *Pixy*

The Pixy camera is a small (43mm x 39mm x 16mm) camera designed for object tracking and detection created in collaboration by Charmed Labs and Carnegie Mellon University. It uses an onboard computer to recognise objects in its field of vision and track them as they move. This is an interesting idea as it aims to separate the object detection task from the robot's processing. This camera is designed to run on an Arduino board, and as such is connected by a ribbon cable connector, rather than a traditional USB cable. However, adapters for USB can be purchased, although it is noted the increased protocols can cause performance issues on devices with limited RAM and flash memory.

The Pixy, however, does not have a depth sensor and as such cannot perform many of the navigational tasks that are expected from a robot. This could be augmented using a second sensor set, however as these sensors are included in the other cameras, this is a lacking feature which requires consideration. Furthermore, the onboard image detection system is built into the camera and the algorithm being used is not editable which severely limits the usages of the camera to those which the developer planned for, and unfortunately, made it non-viable for this project.

2.5 Neural Networks and Machine Learning

Machine learning is becoming a common trend in modern systems with many information technology solutions implementing a form of machine learning system somewhere within their processes. Géron defines machine learning as "programming computers so they can learn from data" [30] which is a concise statement aligning with other researchers in the field such as Pal and Gulli [31]. To achieve this task, there are many different models, architectures, and techniques, with simple solutions like binary trees with deterministic logic, and the fuzzy logic based naive Bayes classifiers.

The two major types of machine learning tasks are regression and classification; the prediction of a number on a scale, and the classifying of data. Regression tasks such as predicting weather and the stereotypical use-case of house prices are common and are an area in which computers excel over humans, as it is often based on large data correlations with many discrete features within a dataset. Classification is used for image recognition, grouping of search and news results, with spam detectors being the go-to use-case for explanatory purposes. Unlike regression, classification tasks are usually based on contextual data from both within and without the dataset, which can limit the effectiveness of machine learning implementations over human capabilities.

How the machine learning models learn and run, is another defining feature with another two major categories; supervised and unsupervised. In supervised learning, data is labelled and results are provided during the training process and the accuracy of the model is based on correct predictions. Whereas unsupervised learning is based on less deterministic data, in which there may not be a correct answer, and the results are subjective. Often used for tasks like clustering, unsupervised learning draws correlations between data objects and organises them based on the predicted strength of these correlations.

With the goal of creating a computer which can think like humans, to complete tasks a human can, and answer real world questions like a human could; creating a model which represents and functions like the human brain and neural network was an obvious solution. And this is the basis for the artificial neural network, which in theory represents the thought process of a human, but which in reality fails in its simulation completely. Advanced Neural Networks (ANNs), more commonly referred to as just neural networks, are based on the multiconnected neurons present in a living brain. Where each neuron runs a single process, but it is connected in some way to the neurons around it.

Neural Networks are configured to have a certain number of layers and for each of those layers a number of neurons. The first layer is known as the input layer, which has one neuron for each data point being passed into the network, and the last layer is the output layer which outputs the prediction or predictions. The layers in between are 'hidden layers' as the weights, inputs, and outputs of these layers are hidden within the network, and even if viewed are incomprehensible to a human viewer as they lack context.

The number of layers and neurons in a neural network are described as the model's complexity, where models with a large number of neurons but a few layers are deemed 'wide', while having a large number of layers is 'deep'. Until in the early 2000s, deep neural networks (DNNs) were highly impractical with long training times even on multi-GPU devices. However, with technology and hardware increases, this has been reduced to a few days of training on such devices.

2.5.1 Pre-Neural Network Methodologies

Before the emergence of effective Deep Neural Networks in 2012, the field of object detection was very different from what it is now. The main focus of research was feature descriptors, methods that can be used to describe the shape, key points, or other defining features of objects within a scene. These feature descriptors can be described as human-driven approaches to computer vision, where the important features for detection and identification are identified by human programmers. In comparison, CNNs and modern machine learning approaches focus on machine-driven approaches where the features are decided by the learning algorithm without human input.

There are many solutions and approaches with three common recognised paradigms; corner detection, blob detectors, and feature point detection. In their survey [32] Tuytelaars and Mikolajczyk recognise corner detectors and blob detectors in their standard context, however also identify region detectors as methods based on intensity, colour, or other identifiers that create homogenous regions.

Corner detectors are models which identify locations of high curvature in two dimensions. This method is effective when the goal is the identification of well-defined objects with a set of repeatable and expected edges and contours. These corners do not necessarily correspond to corners in three dimensions, and as such the effectiveness of these models is heavily dependent on the objects and the background upon which they are taken. A popular example of a corner detection model is the Harris Detector developed by Chris Harris and Mike Stephens in 1988 [33].

Blob detectors use Gaussian equations and filters to analyse the image as a three-dimensional volume, with the intensity of pixels and their surroundings forming the third dimension. In this representation, areas of high intensity are represented by strong negative regions, whereas areas of low intensity are represented as positive regions. These methods are the basis for the popular SIFT (Scale-Invariant Feature Transform) [34] and SURF (Speeded-Up Robust Features) [35] technologies, which are still used in situations where a CNN is too complex or computationally demanding.

Region detectors identify homogenous regions within their assigned feature’s range or spectrum, usually intensity. The most commonly used form is Maximally Stable Extremal Regions (MSER) [36], which identifies and separates the image into regions of high and low intensity. This method creates well separated regions of tonal extremes but is dependant heavily on the clarity of the image, as blurring can smooth the difference between the high and low tonal areas, thus rendering the regions too similar to identify as separate regions.

Once the detectors have identified the key points or areas of interest, the information is fed into a classification system. This system could be a Support Vector Machine (SVM), decision tree, or other machine learning algorithm.

2.5.1.1 *SIFT, SURF, BRIEF, and Fast*

Of these 'traditional' computer vision methodologies, the SIFT (Scale-Invariant Feature Transform) [34] and SURF (Speeded-Up Robust Features) [35] methods are two of the most important. The scale-invariant feature of SIFT describes its ability to standardise the scale of objects using Gaussian functions to identify corners and regions of interest regardless of the scale. This was important as it solved the biggest issue with the Harris corner detector which could struggle to detect edges if the scale was too large. (This occurred when the corner's border was larger than the threshold for corner detection).

SIFT however, suffered from slow processing times due to the series of Gaussian equations which were required to be run over the image. SURF improved this, using a box blur method which takes the average value over windows and in doing so decreases the number of computations required to detect objects.

Features from Accelerated Segment Test (FAST) [37] is a different solution for object detection, designed for computational efficiency. This approach takes each pixel in the image and identifies pixels in a circumference of 16 pixels around it (this is a Bresenham circle[3] with a radius of three), and based on the measured intensity of each of these pixels decides whether the central pixel is a corner. This method is generally fast, with the system being able to quickly eliminate potential options with very little processing. However, the quality of its results depends heavily on the quality of the data, with blurred or unfocussed images often not having as many well-defined corners.

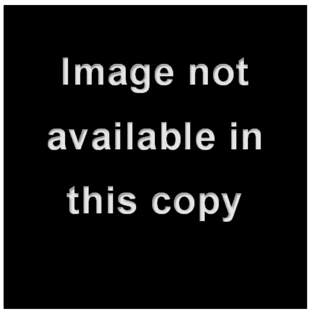


Figure 7 Example of different feature extraction methods (a) FAST, (b) SIFT, (c) SURF [38]

A study in 2016 [39] showed that the traditional computer vision methods are capable, with very little training or data, to identify objects in close proximity to the robot with 70% accuracy. This level of accuracy was lower than the CNN methodologies tested and the range of objects able to

[3] A Bresenham Circle is a rasterized pixel circle based around a mid-point.

be consistently detected was smaller, with the traditional methods finding difficulties with smaller objects such as keyboards and balls.

This study was concluded that for "on-the-fly mobile robotic applications" traditional computer vision algorithms are possible solutions. However, they also demonstrate that the accuracy of DNNs makes them a better option for most situations, if the time and data to train them is available.

2.5.2 Convolutional Neural Networks

CNNs are not a new technology, with their roots stemming as far back as the Neocognitron presented by Fukushima in 1980 [40]. This model was aimed at recognizing "stimulus patterns based on the geometrical similarity (Gestalt) of their shapes without affected by their positions" [40]. This was based on the concept of simple cells and complex cells ("S-cells" and "C-cells"), where the S-cells are trained without supervision on a set of stimulus patterns to elicit a response from a single C-cell regardless of positioning. This was then further trained to provide context from the stimulus patterns identified in the C-cells. The research and concept for this model was heavily influenced by the hierarchy model by Hubel and Wiesel who proposed the two classes of visual neurons in their papers in 1962 and 1965 [41, 42].

While Fukushima was able to run successful simulations on a "digital computer", since the 1980s technology has progressed such that modelling and compiling complex neurologically inspired models is a fairly simple task, and modern hardware is able to perform tasks beyond the limitations with which researchers at the time were forced to work with. In 1998 the first 'recognisable' version of the modern CNN was presented, the Le-Net [43]. This paper showed that the CNN were effective at recognising images of handwritten characters and was reported to "outperform all other techniques". Despite its accuracy however, training the Le-Net was slow and inefficient, and with other technologies such as the Support Vector Machine (SVM) becoming popular due to their simpler training methods, the CNN and Deep Neural Network architecture in general declined in popularity.

Krizhevsky, Sutskever, and Hinton [44] started the shift back to CNN methods in 2012 when their model (dubbed AlexNet after Krizhevsky; first name Alex) showed significantly higher accuracy in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). They used the CNN's large learning capacity and ability to "make strong and mostly correct assumptions about the nature of images" [44] to win the competition with a top-5 error rate of only 15.3%; 10.9% better than the second-best performing network. Despite its strength in image recognition, research had progressed, and the forefront was in the area of object detection, something the CNN could not yet accomplish.

The R-CNN was the solution to this problem, using a region proposal network (RPN) it selects 2000 category-independent regions for each image. Each region is then computed using a CNN to extract features, these are then classified using an SVM. This method provided better results than other models at the time, however, in testing it takes as much as a minute on a standard frame of video (scaled to 500 x 500 pixels) without a GPU to process each image for object detection, and even with a GPU that time is cut down to a quarter – significantly slower than the goal of real-time object detection [45].

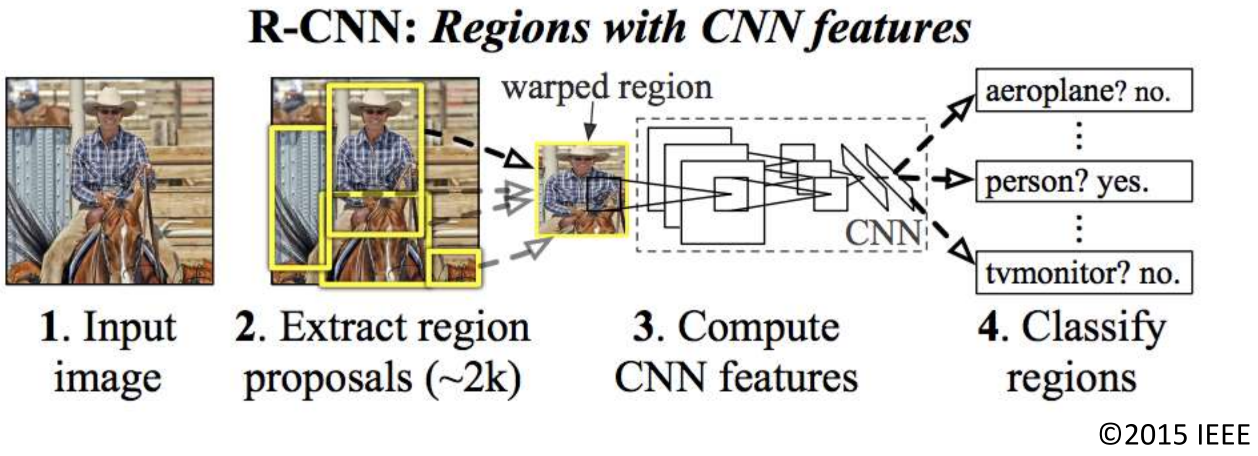


Figure 8 R-CNN model process sourced from the Fast R-CNN paper; [46]

2.5.3 Fast R-CNNs

ConvNets and R-CNNs were effective methods for object detection and identification, however they have a few notable problems aside from the slow detection speed as mentioned in the previous section. Training is a multi-stage process, requiring; log loss algorithm on object proposal layers, SVM fitting for object detection which also function as the classifiers, and finally the bounding box regressor layers are trained. This means that training and evaluation is a slow process with multiple different processes which need to be run and evaluated separately and sequentially. Furthermore, the memory and storage requirements of training this system are very expensive, as each object proposal from the SVM and bounding box regressor is written to disk which could consume hundreds of gigabytes of storage [46].

A faster and more effective model was achieved by Girshick in 2015 who introduced the Fast R-CN, which focussed on improving the cohesion between the layers and reducing the storage usage when training. This was achieved by replacing the final max pooling layer with a RoI (region of interest) pooling layer; which uses a modified variation of the ‘special-case’ of the spatial pooling layer used in SPPNets where the is only a single pyramid level. This allows it to effectively generate a fixed sized region of interest window for each object proposed by the convolutional and max pooling layers, scaled to fit.

Furthermore the Fast R-CNN replaces the R-CNN’s last fully connected layer with a separated fully connected softmax layer and a category specific bounding box regressors working in parallel (termed ‘sibling’ layers by Girshick [45]). This configuration runs the tasks together and calculates loss as a multitask loss between the first layer, outputting the possibility distribution per RoI, and the second layer, outputting the bounding boxes. In the Fast R-CNN’s accompanying paper [46], it is demonstrated that this multi-tasked training method is advantageous as it avoids the management of sequentially trained tasks, but also increases the overall accuracy of the trained models. This is further supported by the usage of similar techniques, and the continued usage of this method in Faster [47] and Mask R-CNN [8] models.

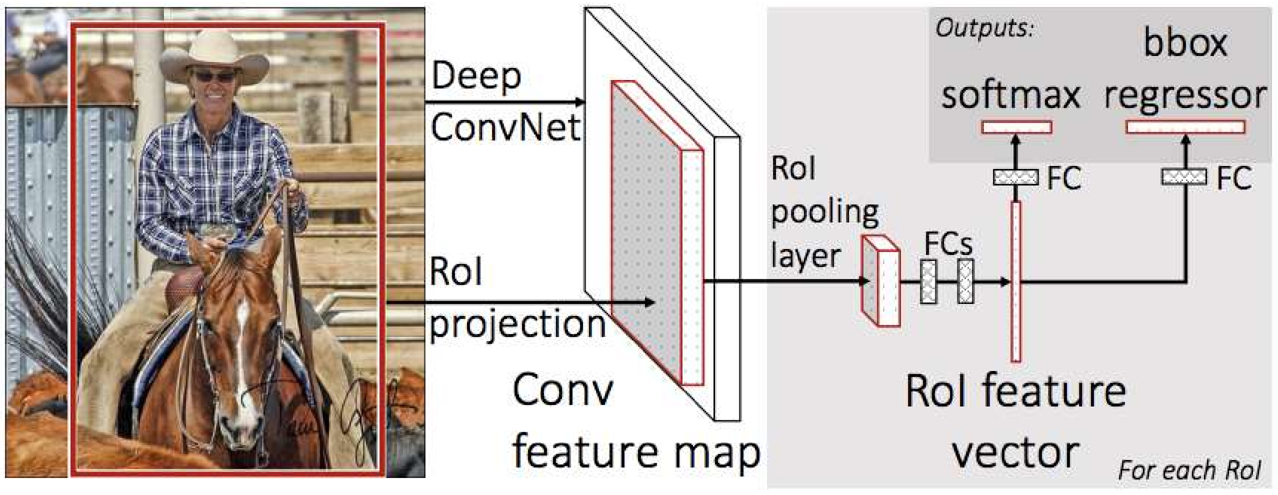


Figure 9 Fast R-CNN architecture model. sourced from the Fast R-CNN paper; [46]

The Faster R-CNN [47] model architecture integrates the region proposal system into the convolutional network’s computations so as to benefit from the shared layers and the creation of a single process object detection and classification system. This Region Proposal Network (RPN) is a deep convolutional network which identifies areas of interest from a sliding window passing over the image. This network utilized modified versions of predesigned CNN models such as the ResNet [48] and VGG[4] [49] models, to identify objects, however exporting the results directly into

[4] Named after its creating group; the Visual Geometry Group, from Oxford

the feature map with proposal ratings before handing it off to the Fast R-CNN classifier in the image data and RoI data format expected.

Aside from the convenience of combining the technologies, this further increases the accuracy of the model, as the predictions and outputs from the convolutions within the RPN can be used as direct input for the classification network with known biases and result features from training. A Non-Maximum Suppression (NMS) was used to limit repeated proposals. This works by only allowing the top prediction of objects within a certain cell based on a measure – in this case the bounding box accuracy.

Despite its name, the Faster R-CNN is not faster than the Fast R-CNN, instead it includes the RPN as a part of its processing, which is optimised to also provide higher accuracy. This inclusion is demonstrated to be "nearly cost-free" [47] with the additional layers only adding 10ms of computing time. This however, results in a faster object recognition process as times for saving and loading the region proposals on the separated classifier and significant overhead. (In the Faster R-CNN paper the Fast R-CNN and VGG-16 take a total 320ms to compute where a Faster R-CNN based on the VGG-16 and Fast R-CNN takes only 198ms) [47]

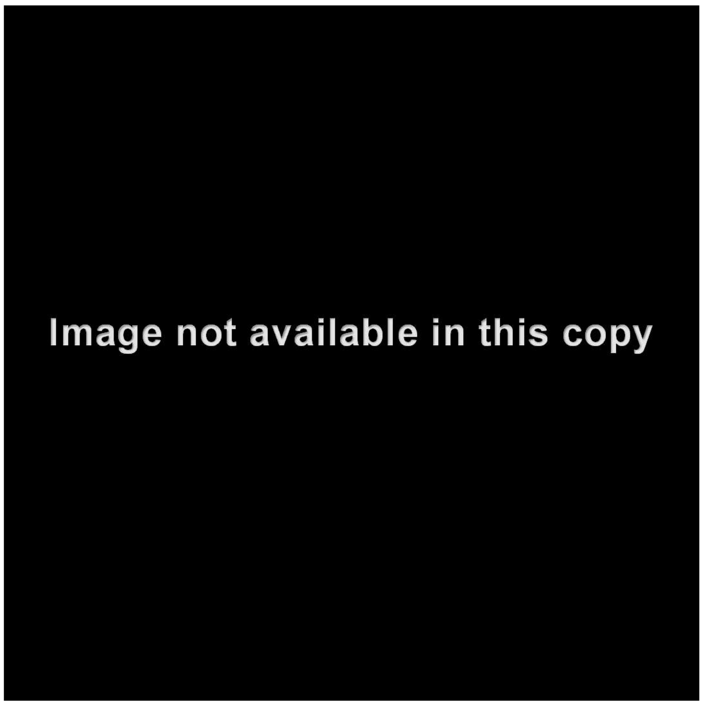


Figure 10 Faster R-CNN architecture model. sourced from the Faster R-CNN paper [30]

2.5.3.1 *ResNets*

ResNets were developed to overcome an increasingly important problem in the training of deep neural networks, degradation. Degradation is a problem that occurs as neural networks get deeper, resulting in first accuracy saturation, and eventually a decrease in accuracy (degradation).

This problem is counter-intuitive to the training methods and expectations which were correlating increased model depth to an increase of data fitting.

This expectation of a deeper model better fitting the data it is trained on is reasonable; much as a higher order polynomial will be much more likely to accurately map all the points in a mapped data array, the layers of a network work as variables, tailoring the process to identify and map the commonalities between the training data. This line of thought results in eventual overfitting, the point where the network has been trained to identify the training data, however therefore is ineffective with testing data, as its algorithms are too specialised – as shown in Figure 11.

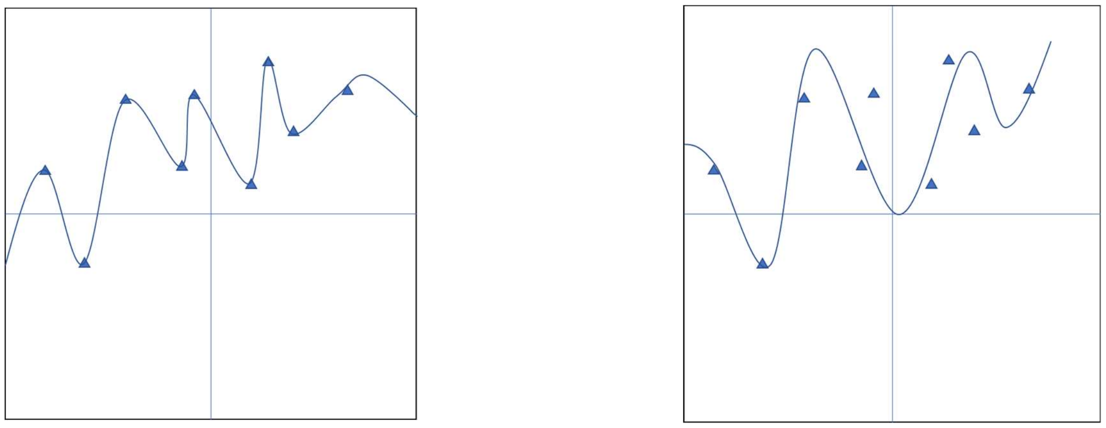


Figure 11 simplified example of overfitting. The left graph is likely over fitted to the data, with each point being directly accounted for by the curve. The right graph however is loosely fitted to the data, this means that it is more likely to represent the natural distribution – although will have a higher error on training data.

This however is not the case; degradation is an increase in *training error* with the addition of deeper layers. This suggests that the optimization of deeper models is more complex than shallower models, and the solvers used for training these models are unable to effectively train these deeper networks. A test was created to identify the solvers as the cause of this problem in which a constructed model was made. Using a shallower model as its top level and using identity layers below that (layers where output equals input), it was shown that a deep model can produced to output a training error that is no higher or lower than its shallow counterpart.

ResNet aims to solve this problem by creating 'shortcut connections' around stacks of one or more layers. This shortcut connection carries the identity (input) value, denoted as x, to the end of the stack, which it then uses to create a residual mapping. Using the same notation as in the original document, the unknown desired mapping $H(x)$, is calculated as a residual using the equation $F(x) \coloneqq H(x) - x$. The optimising code therefore aims to decrease the residual $(F(x) + x)$ in the place of aiming to fit each stack to a desired outcome.

Image not available in this copy

Figure 12 Training error of ResNet models (right) and 'plain' model based on the VGG philosophy [49]

The benefit of this solution is immediately identifiable when the effects of increased layers is compared with other models. As shown in Figure 12, the ResNets show increased training accuracy – and even overfitting on models with increased layers, whereas the ‘plain’ model without this has a decrease in accuracy between 18 and 34 layers. This benefit allows the ResNets to be deeper than other networks, with the most common models having 50 and 101 layers – in comparison to the 16 layers of the VGG.

2.5.4 Mask R-CNN

Mask R-CNN is a model proposed in January 2018 [8] which separates itself from the other models currently in use with its “masking layer” which segregates objects visibly on the screen. This model is based on the Faster R-CNN technology, and has the same primary architecture, with a more effective Region Proposal Network (RPN) and the second masking branch of the model.

The largest change to the RPN is in the ROI pooling layer which was introduced in the Fast R-CNN. This change fixes a rounding error created by the floating-point division process used to create a fixed size co-ordinate feature map. This fix uses bilinear interpolation of sampled points to better align the feature map to the region proposal. This decreases the loss of quality due to the transformation of data sizes throughout the multiple layers of the network, resulting in better fitting region proposals and thus bounding boxes and masks.

In its accompanying paper, the Mask-RCNN is proven to be accurate and fast when a ResNet-FPN backbone is used and performs well with the ResNet-50 (mAP of 30.3 on the COCO dataset). The standard version of the model however, uses a ResNet-101-FPN backbone and when tested achieved a mean average precision of 35.7% on the COCO dataset. Training on the ResNet-101-

FPN however, is a little over 35% slower at 44 hours for the entire COCO trainval135k database (135,000 images) in comparison to the 32 hours on the ResNet-50-FPN.

The Mask R-CNN's primary contribution are the masking layers which allow segmentation of objects as shown in Figure 13. These are calculated separate from the bounding boxes and object classifications and have the ability to add context to environments beyond the bounding box approach, especially where objects overlay within the image and other situations involving dense populations.

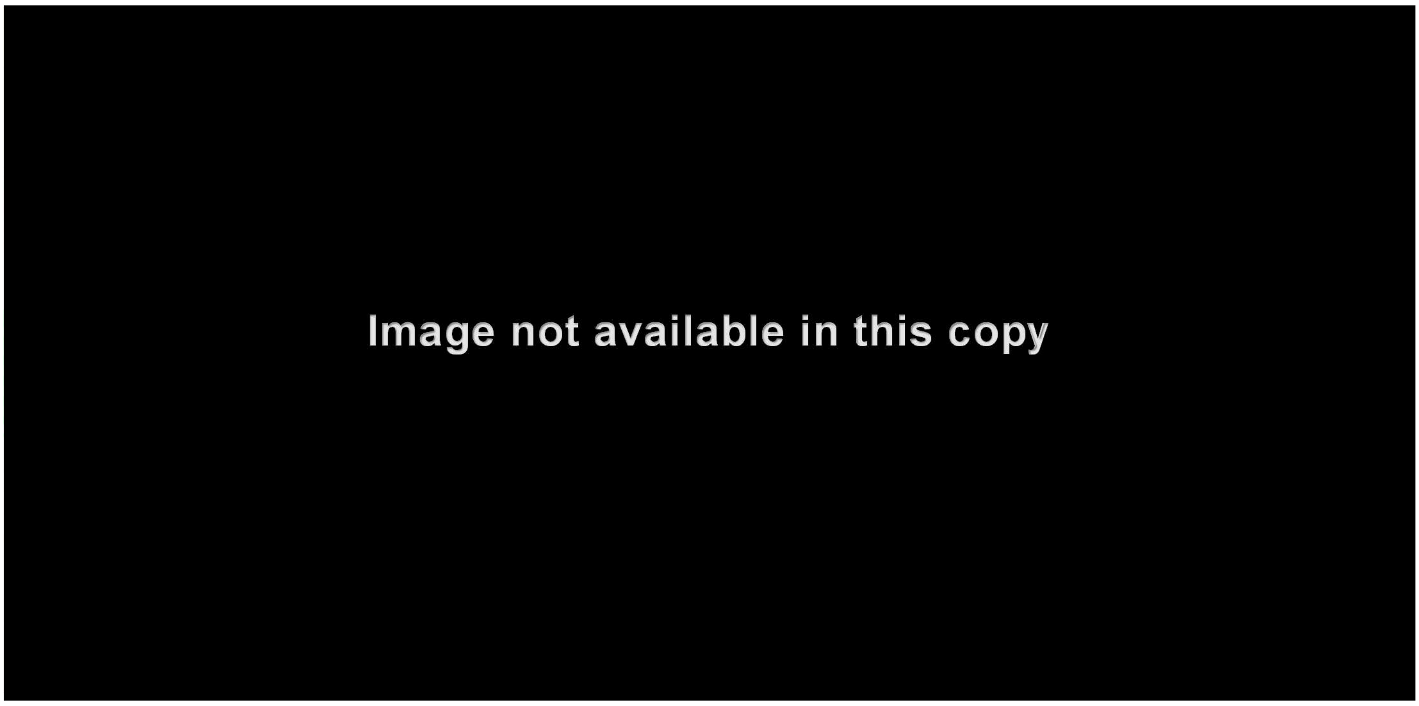
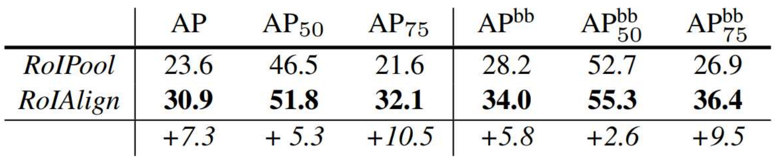


Figure 13 COCO Mask R-CNN example from the Matterport GitHub repository *[50]*

Masking is not the only addition to the faster R-CNN that the Mask R-CNN uses to increase its performance, especially in situations which require accuracy. The RoIAlign process used in place of a RoIPool increases the mAP by about three points, however at higher IoU levels this number increases with a five-point increase at IoU_{75}. This is shown in **Error! Reference source not found.** from the model's supporting paper [8] using a large-stride feature detector to create severe misalignments, resulting in big accuracy gaps between the technologies.

Table 2 RoI Pooling methods vs RoI Align methods on the Mask R-CNN with a ResNet-50-C5 stride of 32. Trained on the COCO trainval35k dataset and tested on minival dataset. (bb is Bounding Box data) *sourced from the Mask R-CNN Paper [8]*



	AP	AP_{50}	AP_{75}	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}
RoIPool	23.6	46.5	21.6	28.2	52.7	26.9
RoIAlign	**30.9**	**51.8**	**32.1**	**34.0**	**55.3**	**36.4**
	+7.3	*+ 5.3*	*+10.5*	*+5.8*	*+2.6*	*+9.5*

3 DESIGN

This thesis aims to test the possibility of using modern methodologies to achieve accurate recognition of objects in a cluttered environment. This chapter discusses the methodologies used to determine the best fitting hardware and software to base the tests on.

Before creating a software solution could take place, specifying a pseudo-robot which the tests could take place on was important. This included identifying possible visual sensors which would be fit for the purpose of navigating a robot through a dynamic environment, and a computational unit which could function on such a device also. This serves as a proof of concept “robot” and functions as an example of the inputs and processing power available for this task.

3.1 COMPUTATIONAL DEVICE SELECTION

Selection of a computational device had been completed before the start of this thesis, with the main two options being the Jetson TX2 and the Intel NUC (BOXNUC7i7). Both computational devices have strengths and weaknesses as discussed in section 2.3.1. However, the focus on performing machine learning without a graphics processor eliminated the Jetson from the selection pool.

Were the constraint on GPU processors removed, then the Jetson Xavier released in August of 2018 would have also been tested as a computational device. The Xavier, like the TX2 is an SBC with inbuilt GPU cores for advanced processing of image data. This board is designed to be small and compact, while providing all the necessary processing power for running machine learning and computer vision on an IoT or robotic unit.

3.2 Camera Testing

Deciding which camera to use is one of the first design choices in this project, as there are several considerations and requirements in the selection. The quality and accuracy of the images produced by the camera and depth sensor could have a serious effect on the results, therefore determining the most accurate sensor is key.

The key features being tested are:

- The image quality, as measured on the ISO[5] 12233 testing board
- The field of view of both colour and depth
- The accuracy of the depth camera and
- The range of the depth camera

Other features being measured are the resolution of the outputs, and the frame rate. These two features are not as important in this case as the speed of the system will be limited by the machine learning algorithm, and the size of the image in pixels is also determined by the machine learning algorithm and will not surpass 500 pixels in any dimension.

Field of view was decided to be of importance as a wide view allows for greater contextual data collection, and better navigation due to peripheral vision – especially in live or dynamic environments where other obstacles may come in from the sides. This must be tempered with a high resolution, otherwise the detail of the frame's contents may be lost due to zoom factor.

Frames per second is also an important feature, while a very high FPS is redundant, as the machine learning algorithm will be unlikely to perform at high speeds, creating a bottleneck, regardless of the camera's framerate. As such, the FPS needs to be high enough to collect a constant stream of data such that the robot can navigate at a reasonable pace without colliding with yet unseen obstacles. Again, this is increasingly important in dynamic environments, where the speed of other objects as well as the robot is a consideration.

Finally, the depth qualities, which are the minimum and maximum range, as well as the level of error within the captured frame. As depth is the primary advantage of these multi-sensory devices over simpler (and less expensive) camera-based sensors, and likely to be an important part of the navigation & classification processes, the quality of these must be high enough that the data

[5] ISO is the shorted name of the International Organization for Standardization, derived from the Greek word isos meaning equal – this was done for the purpose of internationalisation. [60]

can be trusted to be accurate, as well as be able to capture information in a task appropriate range.

To be fit for the purpose of this project, the camera is expected to have:

- A field of view of approaching 120 degrees; the field of view of a human eye
- A framerate higher than 50 FPS; resulting in about 3cm of forward motion between each frame at average human walking pace
- An average depth error at 1m less than 10cm; the depth data is worthless if it is unreliable or inaccurate
- A maximum depth of at least 5m and minimum of at most 0.5m; both should be easily obtainable for the depth sensor, and if a device cannot achieve this it is not suitable for the task.

The camera is also expected to have a large focal range with high f-stop; to allow accurate collection of data from both close and distant ranges. However, this is not something that can be accurately measured with the equipment made available to this thesis, so must be considered as a qualitive requirement. The resolution of both the colour and depth images will also be measured, however as the image will be formatted to a fixed size square for the machine learning, between 300 and 600 pixels, this feature is mostly relevant as a tie-breaker or minor consideration.

Three cameras were tested using the methodologies set out in section 4.2, these were the Intel RealSense D435, and the Xbox One Kinect for Windows, also considered was the ZED camera however due to hardware requirements it was removed from the tests. The Kinect was chosen due to its popularity and usage in many robotic and computer vision researches as described in section 2.4. The RealSense is the logical comparison for the Kinect, as its intended usage is for robotics and computer vision tasks, although its usage in the field is not as prolific.

3.2.1 Image Quality

The official method to test the pixel quality of a photographic device is to use the ISO 12233 testing board [51], which has multiple tests which when used alongside the supporting software can analyse many different features of the camera, such as lens distortion, post-processing flaws, and the pixel quality of the camera.

However, the old (2000) version of the board alone costs at $450 from the official source [52] and after reviewing, it was decided that this level of detail would not be beneficial to the thesis proposed. To ensure the cameras used were of reasonable standard however, a free version of the ISO12233:2000 board re-created by Stephen H. Westin within limitations of the copyright held by ISO was printed onto A3 paper at a high DPI (Figure 14) [53].

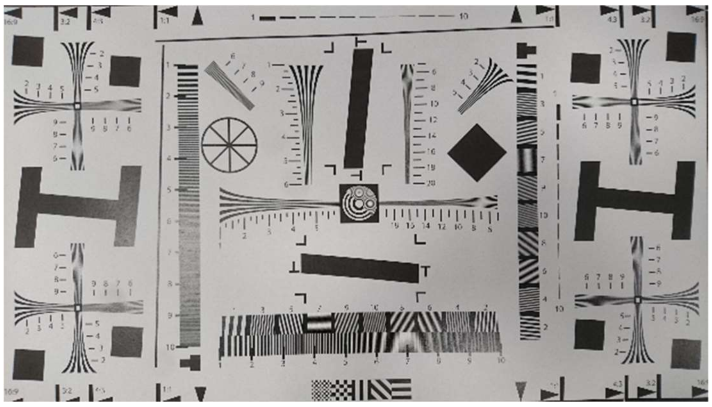


Figure 14 Unofficial ISO12233:2000 Camera Testing Board

This test is not suitable for official standards as the printer, paper, and lighting amongst other features can affect the outcome of the test, and without the accompanying software, accurate analysis of the information collected could not be ascertained. However, for proving that the cameras are suitable, this basic version of the test is functional and cost effective.

The testing board is to be lit evenly and placed at a 90-degree angle (standing vertically) on a desk, the cameras being tested, will then be placed on a tripod, or stand if tripod support is not available and moved such that the frames match the testing board's guide ratio markers.

Without the software required for testing, the only reliable test that can be conducted with the board are the horizontal and vertical quality tests. These tests (horizontal version shown below in Figure 15) work by measuring the number of lines the camera can capture on each axis, identifying the point where a selection of converging lines become indistinguishable in the camera's output. The numbers describe how many lines fit within the frame, measured in magnitudes of 100.

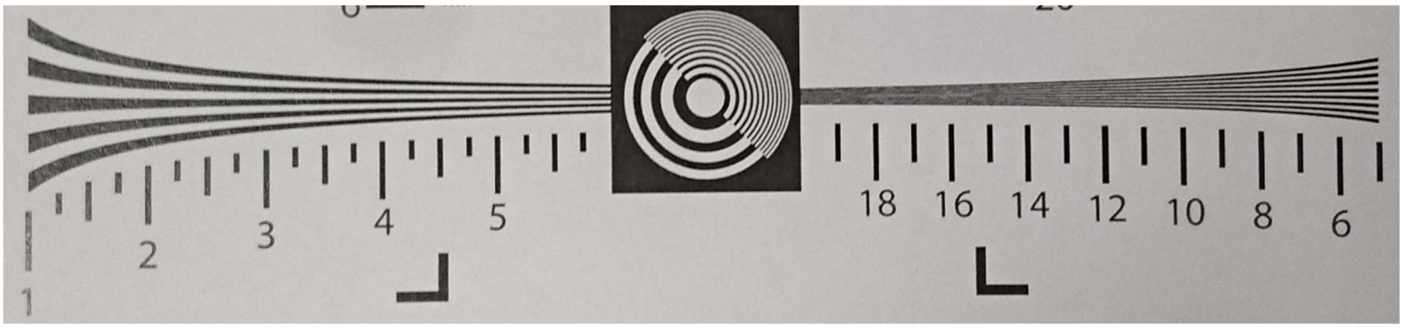


Figure 15 The vertical line test of the ISO12233 captured on the 27 mega-pixel camera of a Sony Xperia Z5. Without the proper analysis tool, this camera manages to score around 17 – which means the camera and lens

can capture around 1700 lines vertically in a frame. However, the designed analysis tool may be able to extract more detail from the image than the human eye reading the data.

Both of the tested cameras managed to score similar results for vertical pixels and horizontal, however the images collected on the Xbox One Kinect were generally in worse focus, which meant that the text was washed out and faded, also marring accuracy on the tests. Despite several attempts to prevent the issue, the results from the Xbox were of significantly lower quality. This could suggest there was an issue with the hardware or controlling software that drives the camera and could mean that the results should have been higher than recorded. However, without being able to produce better results the Kinect must be judged equivalent in this view to the D435, if only due to its incapability to accurately focus on an object.

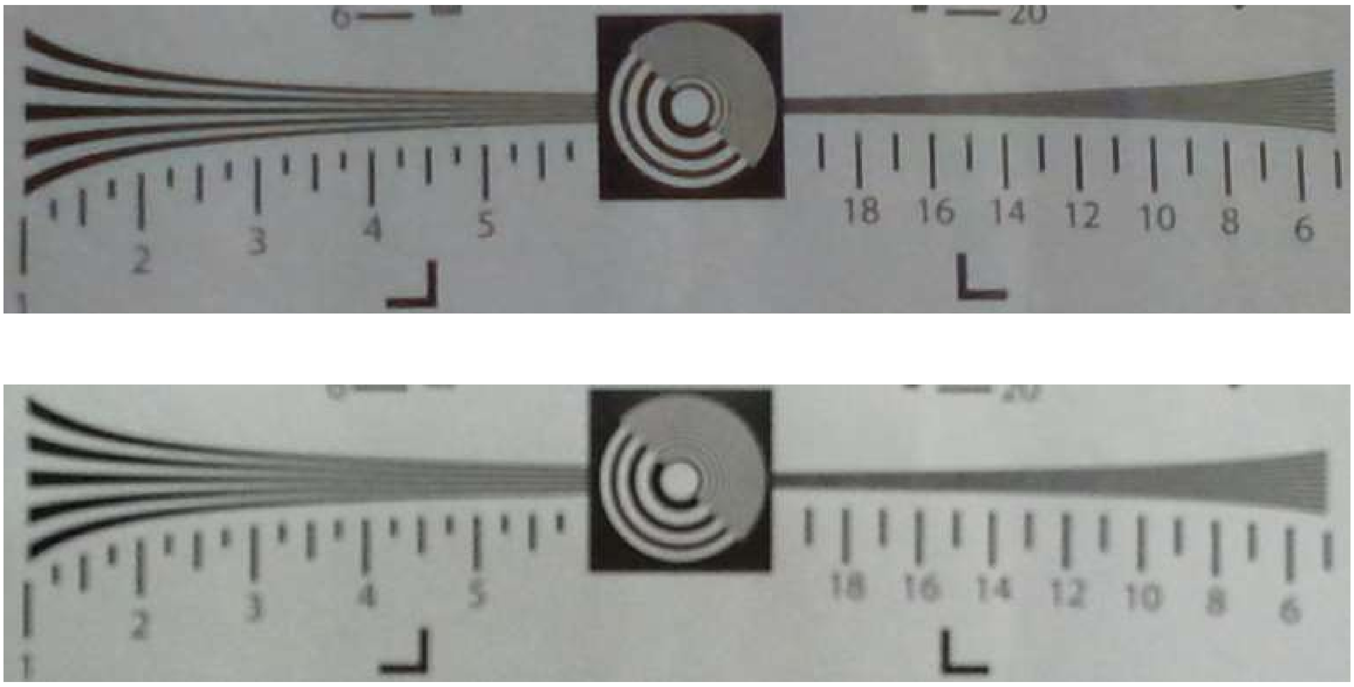


Figure 16 Vertical pixel density measures for the ISO 12233. The D435 (top) clearly shows a pixel accuracy of 600 horizonal lines in an image. Whereas, the Kinect (bottom) was unable to focus on the testing board when placed at the required distance, but also resulted in a score of 600 horizontal lines.

3.2.2 Field of View

Field of view was decided to be of importance as a wide view allows for greater contextual data collection, and better navigation due to peripheral vision – especially in live or dynamic environments where other obstacles may come in from the sides. This must be tempered with a high resolution, otherwise the detail of the frame's contents may be lost due to zoom factor.

There are two common methods for measuring the field of view of a camera; the first method is to literally measure the camera's field of view, using a protractor (or other such device) and guide lines, while the second is to mathematically calculate the field using the known size of an object and the size it appears in the frame at a series of set distances. Due to the human requirements of collecting accurate measurements, it is more reliable to use a calculated method which can be easily repeated and regulated.

The method for calculating the field of view of a camera uses an object of known dimensions measured at two known distances. The field of view can then be calculated by measuring the fraction of the screen filled by the known objects and using that ratio to measure the total size of the frame, using the known distance of the object from the frame and the size of the frame (at that location) it is possible using trigonometry to calculate the angle of the frame.

I = size of Image in Pixels
measurement (m)
P = Size of Object in Pixels
α = FOV
	S = Size of object in real-world
	
	d = distance from camera to object
	C = lens features (zoom, recession, etc.)

1	$$\text{Opposite} = \frac{IS}{2P}$$
2	$$\alpha = 2 \times \tan^{-1}\left(\frac{Opposite \times 0.5}{d + C}\right)$$
3	$$\tan^{-1}\left(\frac{Opposite_1}{(d_1 + C)}\right) = \tan^{-1}\left(\frac{Opposite_2}{(d_2 + C)}\right)$$
4	$$C = \frac{Opposite_1 \times d_2 - Opposite_2 \times d_1}{Opposite_2 - Opposite_1}$$

Equation 1 Field of View Equations

The first part of the equation works out the opposite edge of the right-angled triangle formed by the centre of the frame and distance from measured object. A scale for working out the size of the frame in measurable units at the distance can be created by working out the ratio of the screen filled by the object. Using trigonometry, in the second part, the FOV angle can be found easily.

For this test, a cork board of 40cm x 40cm is to be captured centre frame at several distances ranging between 0.50m and 2.00m. The size of the board in the frame will then be measured in pixels and the field of view calculated. This process is repeated 30 times, with the board reset between images, to reduce any error due to inaccuracy of board placement and angle.

Table 3 Camera fields of view

Camera	Mean FOV Width (degrees)	Std. Dev. Width	Mean FOV Height (degrees)	Std. Dev. Height
Kinect - Depth	68.47	0.60	52.25	0.54
D435 - Depth	89.62	0.93	57.07	0.67
Kinect - Colour	82.32	0.58	52.36	0.50
D435 - Colour	70.21	0.62	41.97	0.54

The results from these tests (above) show that neither of the cameras approach the field of vision attainable by a human eye (about 120 degrees). The results from these tests show that the difference in the sensor fields of view are statistically significant with a P value less than 0.05 for all 4 categories (both cameras, width and height) when compared with an unpaired t test.

However, the D435 has a better field of view on its depth camera, while the Kinect in its colour.

3.2.3 Depth Quality

The depth qualities, which are the minimum and maximum range, as well as the level of error within the captured frame are important to measure as the depth data collected are likely to be an important part of the navigation & classification processes. The quality of these must be high enough that the data can be trusted to be accurate, as well as be able to capture information in the specified range as described in the SFTI Robot Design section (2.3.2).

To test the depth sensor, the camera was placed at set distances between 0.5 and 5m from a testing board[6], held at a 90-degree angle, as in the field of view test. In front of this board was placed a box creating a 5cm difference in the surface of the board. The cameras were then moved between the set distances, and the ability to separate the box from the board measured using the numerical and visual outputs from the depth sensors.

[6] The depth qualities were captured at 0.5, 0.75, 1, 1.5, 2, 2.5, 3, 4, and 5 metres

Both cameras measure distance on their camera in non-measurements, instead using a value which must be converted into a metric for real-world usage. With the D435 this can be done using the librealsense library, and is implemented to allow for a fluid scale, such that greater distance can be shown, or higher detail where that is not required. This means while the maximum range for the camera is around 10m, however it can “express” distances up to 65m, which is useful when an external light source is present (rather than the low power emitter included with the device).

The accuracy of the RealSense camera was very high at close range, with the distance of the board measured at 1.01m when the camera was 1m away, the box being measured at 0.96m. However, the visual representation of this data was very ineffective, with the distances smoothed together to create a clearer image. This however, meant that the box was identifiable in the numerical measurements until 2.5 metres but was lost in the smoothing at only 1m. This is due to the ‘dynamic visualisation’ setting which Intel have built into the depth stream reader. This setting can be modified to ‘near’ or ‘far’ visualisation, but doing so has an inverse effect on the visualisation of objects in the other category. Detection of the 5cm box was visible in the depth measurements until 2.5 metres, and the board was detected at the end of the test at 5m.

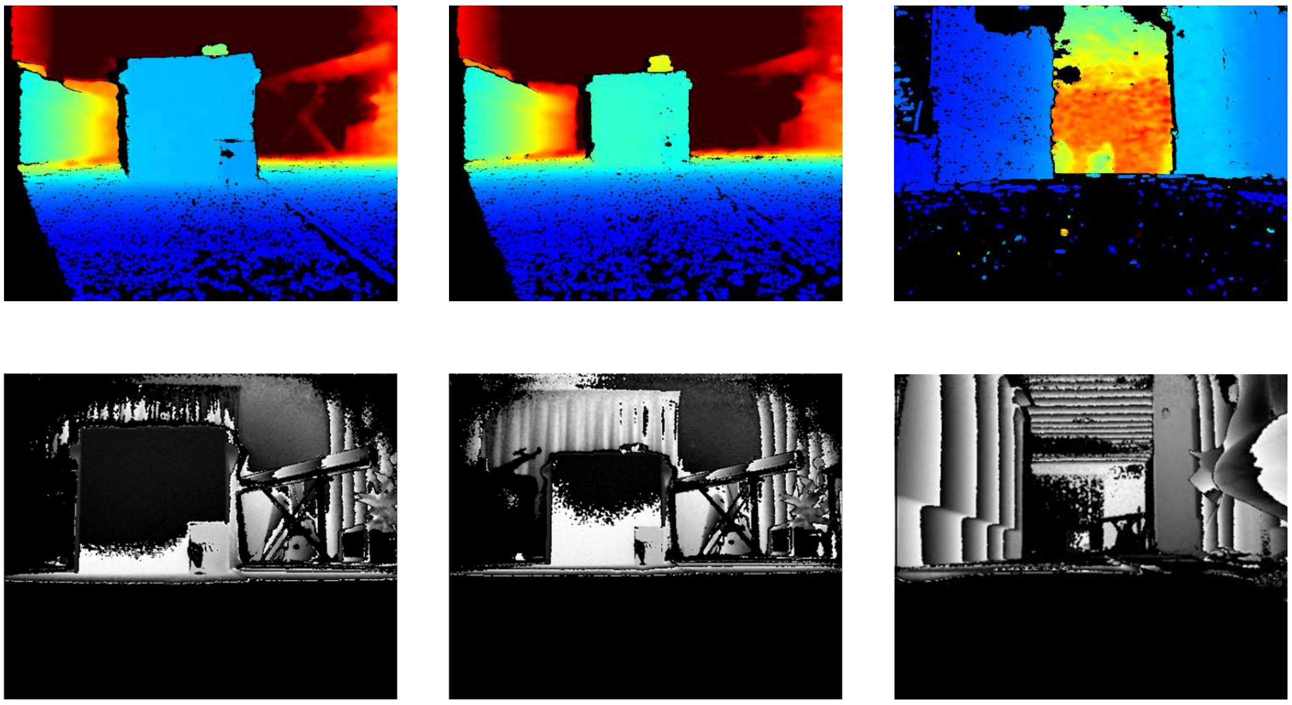


Figure 17 visualisation of the RealSense D435 (top) and Kinect V2 depth data (bottom). At 0.75m (left) the RealSense clearly displays the board, and box with little noise on the image, whereas the Kinect seems to struggle to measure the entirety of the board at this distance. At 1m (centre) the D435 barely is able to display the box on its visualiser, although it is still measured in the numerical data, whereas the Kinect clearly displays and measures both the board and box, with a significant decrease in noise from the closer ranges. At 5m (right) the RealSense needs to switch to far more to display the board. At this point the box can no longer be detected. However, the Kinect still clearly shows both board and box.

The Kinect V2 in comparison had the best long-range performance, clearly identifying both the board and the box at 5m. However, while at long distances this camera retained its accuracy, the camera was inaccurate and noisy at closer ranges, with the measurements collected at distances below two metres being rough and high in variance.

This behaviour was also noted by Halmetschlager-Funek et al. [27] in their study of depth cameras published March of this year (2019). Their study focussed on empirically evaluating popular depth camera technologies. Two of the metrics they used were the precision, and bias of the image, precision being the stability of the measurements, while bias is the relation between the measurements and the ground truth.

In their study they found that the range of the D435 was large, between 0.2 and 7m and support the findings of this test that the camera has high precision and low bias at close ranges (less than 1m). Their assessment of the Kinect also matches the results from this test, demonstrating superior results when compared to all other tested devices long ranges, but the data it collects is "less smooth" and has "inferior surface representations for midrange depths" [27].

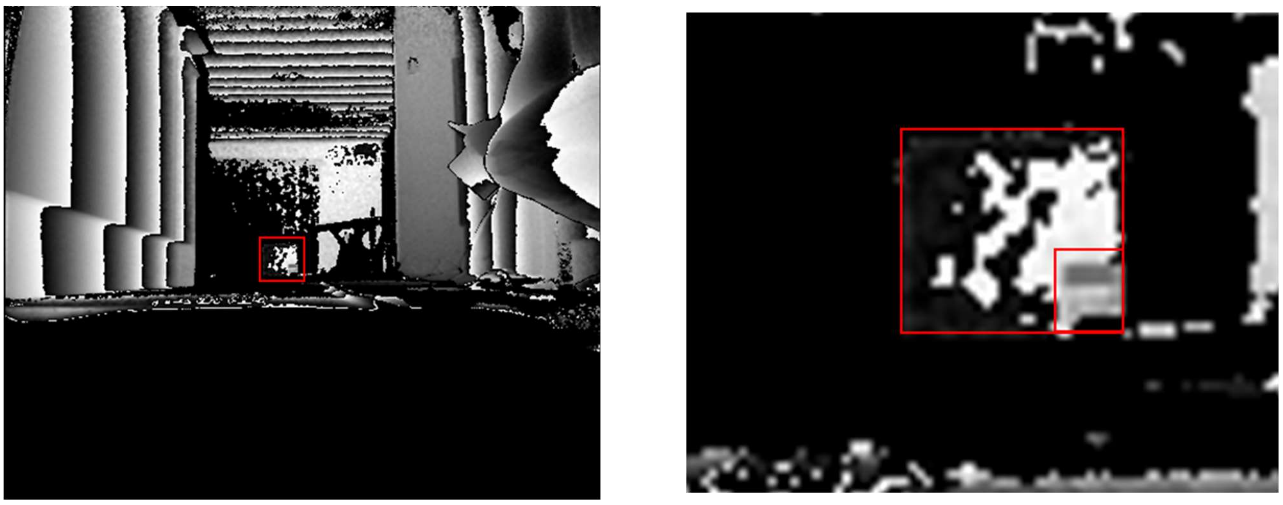


Figure 18 Kinect recorded image at 5m the board in the centre of the image is clearly visible and when zoomed in (right) the test box place 5cm in front of the board is also visible. Noise is evident in the image, and the representation of the narrow hallway and ceiling is uneven. This will likely be due to the structured light pattern which is unable to accurately measure the near 90 degree angle the walls have from the camera.

That this study was released within the duration of this project shows the importance of these devices in the field. With the recent (last half-decade) increase in RGB-D and stereoscopic cameras for Internet of Things (IoT) and robotics applications, the need to identify devices which can be relied on to accurately measure and understand the world in three dimensions is growing with the increase in robotics and automation.

3.2.4 Overview

3.2.4.1 *Kinect*

The Xbox One Kinect for Windows benefits from a well-developed and established official SDK and community of developers and researchers which means that it has many tools and applications which can expedite the development of solutions for this device. Built on the reputation of the first Kinect, which is commonly considered the first affordable depth sensor [27, 24], the Kinect V2 is a versatile device with a high quality colour camera and effective depth sensor.

The colour camera on this device is high quality with a maximum resolution of 1920 x 1080 pixels, capable of 600 lines horizontally within a frame. The lens allows for a large field of range covering 82 degrees horizontally, and 52 vertically. The depth camera however has significantly lower performance with a max resolution of 512 x 424 pixels and a field of view of 68 degrees horizontally and 52 on the vertical axis. The image produced by this sensor was generally noisy, however, it had high accuracy for distant objects, identifying a 5cm difference in a surface from 5m.

Unfortunately, the Kinect V2 has an issue with its power consumption, cable design, and size. While most RGB-D cameras are small and compact, so to fit into the robotic solutions they are designed for, the Kinect is a large bulky camera, with specialised cables and adapters that are needed to operate the device. This includes an adapter, to convert the Kinect connector cable into a USB connection and auxiliary DC power cable, and a DC transformer, which takes as standard 240V AC power socket and outputs the 12V 2.67A which the Kinect runs off. (Noting that the USB 3.0 maximum output is 5V and up to 2.1A for charging-specific devices). This need for a separated power supply is also a design consideration required when using the Kinect for robotic purposes.

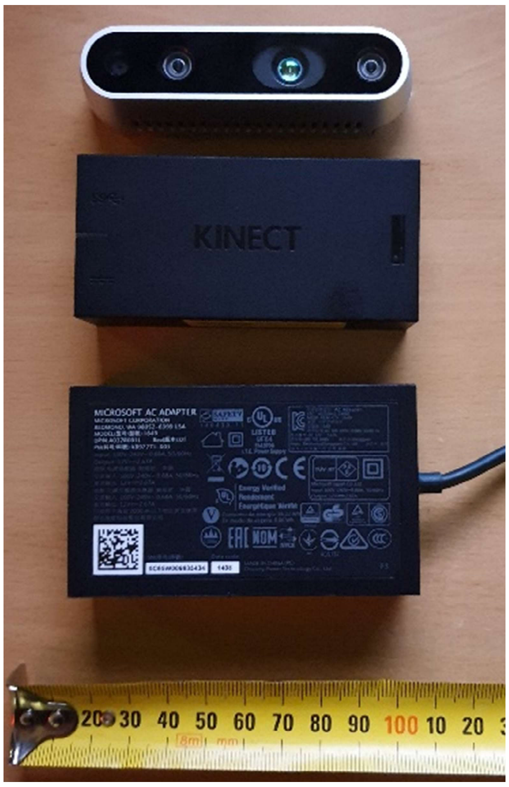
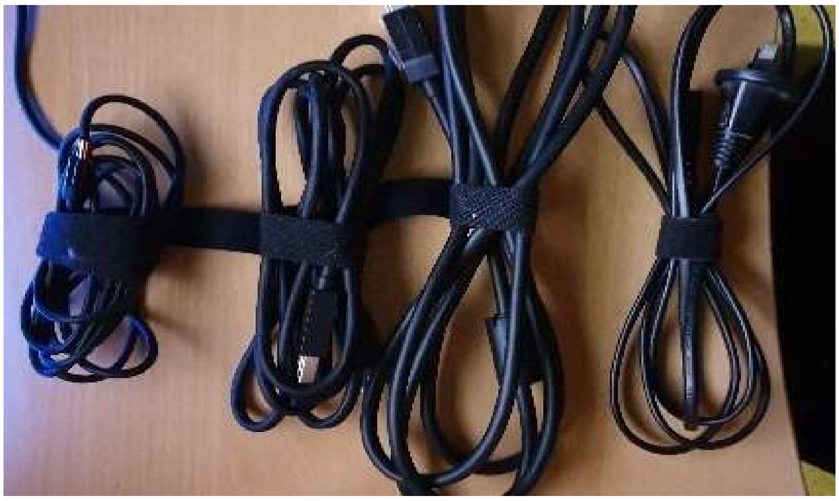


Figure 19 (left) The Kinect power transformer (bottom) and adapter (centre) are both larger than the D435 camera which requires no additional external units.

Figure 20 (below) The Kinect for windows package has 4 cables, with a combined length of 8.8m. Fitting this much wiring into a robotic solution might be a problem where space is a consideration.



3.2.4.2 *D435*

The Intel RealSense camera benefitted largely from its high resolution for both the colour and depth sensors, and the frames captured were clear and detailed. Its API and code support from Intel are of high quality and are easily accessed, and there is a convenient SDK for Windows, Linux, and Mac operating systems, with precompiled tool and code bases to work with.

As with most modern cameras, the D435 has a full HD colour camera capable of capturing video at 1920 x 1080 pixels with a frame rate of 30 FPS. Its field of view is slightly smaller than the Xbox Kinect at 70 degrees horizontally by 42 degrees on the vertical plane. However, the depth sensor has a resolution of 1280 x 720 pixels and a field of view of 89 degrees horizontally and 57 degrees vertically. The depth sensor also has a good field of range, with a minimum range of 0.3m and maximum range of 6.7m.

The lack of support for the past versions of the RealSense could result in problems with extended long-term usage of the device as it is likely that the robotic unit will have an extended lifetime, which may leave the camera device unsupported in the case of potential hardware or software updates. However, for its performance, it is clearly the best device currently available. There was also a noticeable issue this camera experienced when another IR source was used in the vicinity, such as the Xbox One Kinect camera. When the IR light profile of the room changed, the camera struggled to compensate, often leaving the display flashing with measured distances and the brightness of the IR imaging fluctuating rapidly. This could be a serious issue if multiple devices are to be used in a single environment and may require further research.

3.2.4.3 *Decision*

The Kinect and D435 both performed very well in a range of areas but the D435 has a higher resolution and framerate on the depth sensor, which is beneficial for applications in which the device is moving, or the environment is changing. The depth sensor from the Kinect had a much better representation of the environment, with high definition for objects at a range. However, the data was noisy in general and suffered from poor representation of flat surfaces at high angles.

As well as measured device performance, the smaller physical size of the D435 and the ability to power via USB, rather than requiring a 12-volt adapter, make it a better choice for mobile robotics. As such the D435 was used as the device of choice in this project.

3.3 Image Type

During the initial scope and data collection, the type of data being collected by the cameras was considered. Due to the addition and requirement for depth sensors on the camera, it was tempting to use the depth stream to add extra data to the object detection system. However, after initial testing it was decided that the depth stream should be used in a separated navigation and object detection module which acts independently from the object detection and classification system. This data can be later integrated into a central control system which manages robotic behaviour and navigation, however the increased computational processing load of this type of algorithm on an individual robotic unit should be avoided – based on the result of using the processors for object detection alone.

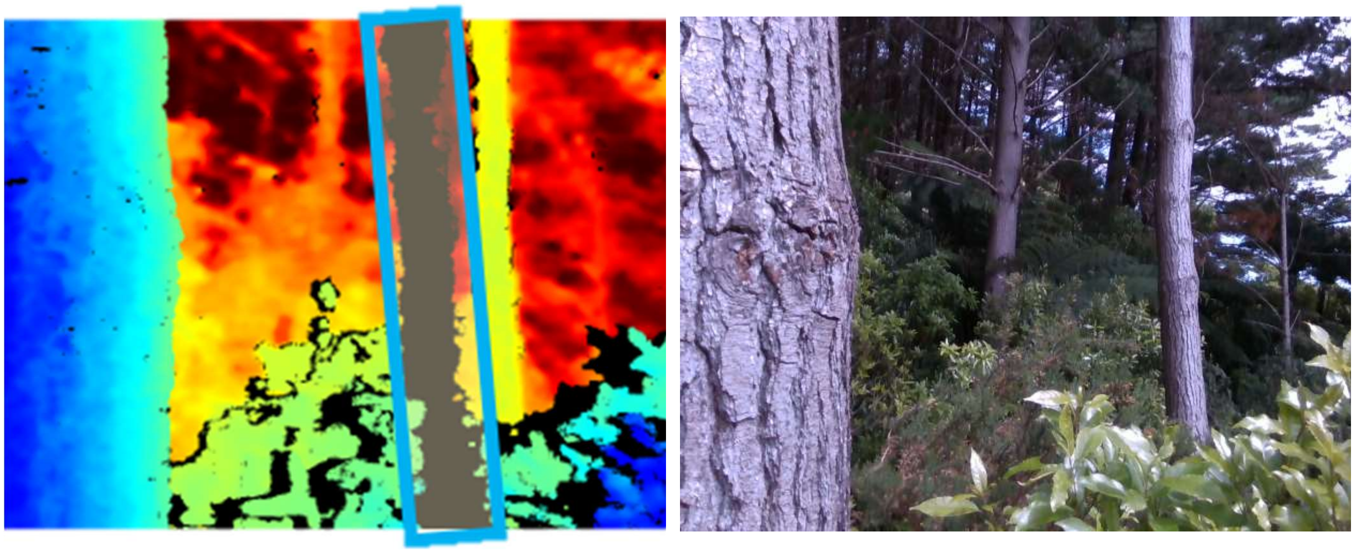
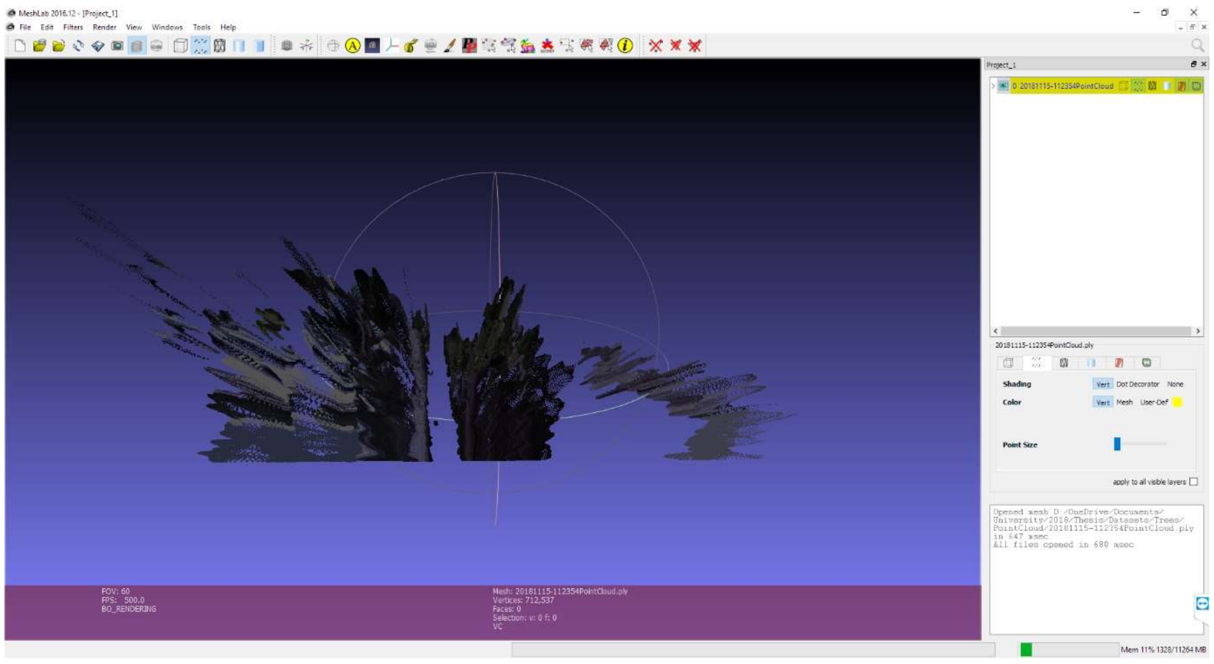


Figure 21 Foreground noise (highlighted) in depth heat-maps is difficult to identify and adds confusing information

This decision was made after examining the data collected from the depth sensor in a forestry environment in both false-colour depth mapping and as a single point of origin point cloud. As shown in Figure 21, the data collected from the depth sensor of the cameras is easily disrupted, incomplete, or simply confusing. This data may be trainable on a neural network. However, it also means adding extra variables which may be confusing features, obfuscating obvious truths, which should be identifiable (for example the trees below in Figure 22 are very hard to find, even for a human).


MeshLab 2016.12 - [Project_1]
File Edit Filters Render View Windows Tools Help
Project_1
0 20181115-112354PointCloud
20181115-112354PointCloud.ply
Shading Vert Dot Decorator None
Color Vert Mesh User-Def
Point Size
apply to all visible layers

```
FOV: 60
FPS:   500.0
BO_RENDERING
```

```
Mesh: 20181115-112354PointCloud.ply
Vertices: 712,537
Faces: 0
Selection: v: 0 f: 0
VC
```

```
Opened mesh D:/OneDrive/Documents/
University/2018/Thesis/Datasets/Trees/
PointCloud/20181115-112354PointCloud.ply
in 647 msec
All files opened in 680 msec
```

Mem 11% 1328/11264 MB

Figure 22 Single point of view point cloud of the same scene – finding the trees in this image is difficult even for a human operator

However, for navigation and object avoidance, depth sensors and point maps are powerful tools, able to create representations of objects and landmarks in 3D space. SLAM technology uses depth sensors in a range of methods to accurately map and navigate environments [24, 54]. Using a combination of optical depth sensors (such as those included on the Kinect and D435) and laser range finders for obstacle avoidance is significantly faster than object recognition. This suggests that separating these is the best method currently available for real-world application.

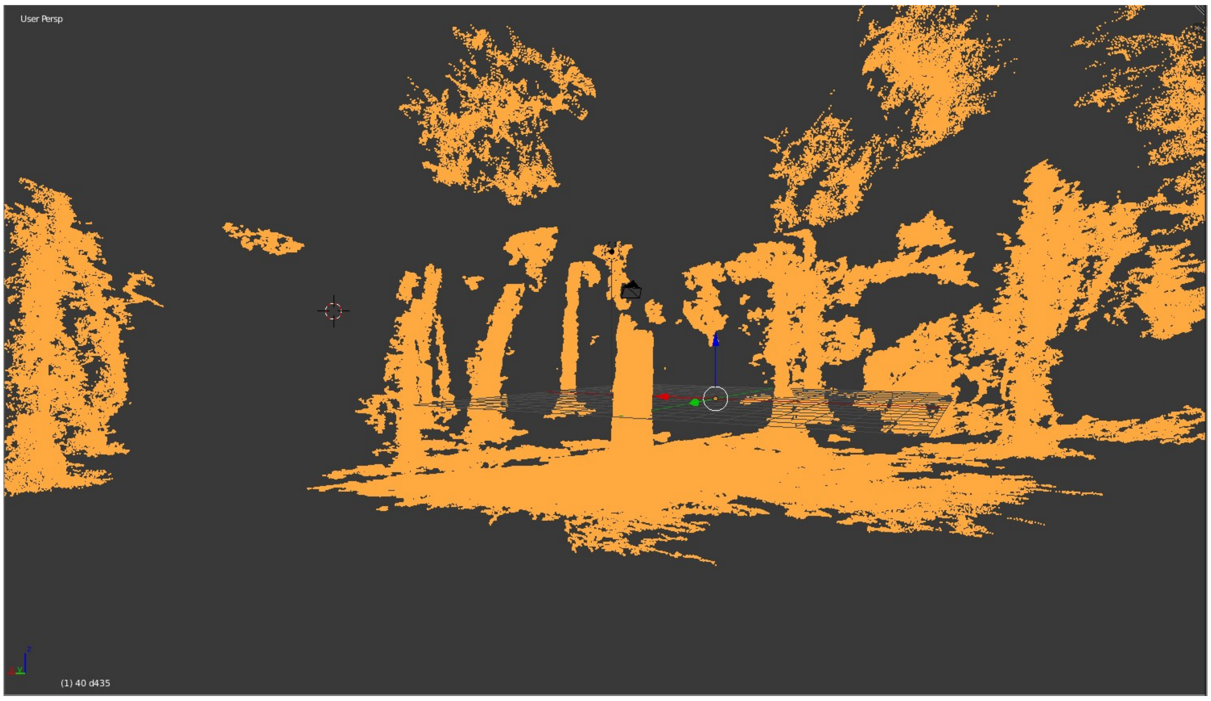


Figure 23 Point Cloud of trees from created from 40 images 360 degrees around a central tree provides a clear map of the area (point cloud provided by Dr Richard Green from the University of Canterbury [55])

3.4 Machine Learning Tests

Before effectively using a machine learning model in this project, it is important to assess the strengths of the model and thus create a basis for expectations of the results. This is important due to the black-box nature of the connections and values passed between layers in deep neural networks. This lack of visibility into the internal processes, as well as the some-what serendipitous nature of configuring and training machine learning algorithms, makes this step essential for evaluating the quality of the results collected in the final tests. Without creating this basis for testing, any results collected in the final test would have no direct comparison, and thus any results collected, whether good or poor, could be entirely based on a poor configuration or dataset feature.

While there are many effective networks available, the ability for these to accurately identify objects on a limited hardware device was something that required further examination; as such the networks being tested are split into the faster models and the masking models. The former of these two meaning models designed for speed of detection over accuracy, including models such as the SSD (Single Shot Detectors) [56] and YOLO (You Only Look Once) models. The models to be used in the category are the MobileNet SSD and InceptionV2; both models are reported to have a framerate of over 10 FPS but are lacking the complex architecture of the Faster R-CNN models which theoretically increases the precision of the models. The DarkNet (also known as YOLO) model was also considered for testing in this category of network; however, it is not compiled to be trained with the standard methods utilised by TensorFlow and as such combining it with another model such as the Mask R-CNN requires much transcoding. This could require a shift from the CNN architecture that the other models are based on, making any comparison of the results created less reliable as multiple non-relevant variables might have been changed – such as the evaluation methodologies and scripts.

The second category of models, includes models of the R-CNN family (R-CNN, Faster R-CNN, and Mask R-CNN). Of these models, only the Mask R-CNN and the Faster R-CNN were tested. This is because masking feature of the Mask R-CNN was most likely to create the results desired, but this would be compared with the results from the very similar Faster R-CNN to show the difference the newer ROI pooling and the Masking layers had on the overall performance of the model[7]. These architectures were tested using the ResNet 50 and ResNet 101 backbones, primarily as these are the most common backbones for these architectures, and are the backbones utilized when architectures were presented [46, 8].

The models were tested on two datasets, these are the COCO dataset and a Fruit Dataset collected for this project. The fruit dataset is an image recognition dataset of a little over 200 records for each fruit (Kiwi, Orange, Apple, Pear, and Lemons). This is a very small dataset, and the images are captured on white backgrounds, so it is sub-optimal for object detection scenarios, but it shows the effectiveness of the models when operating with very little training data available. The COCO dataset is collected in assorted live environments and will therefore better represent the real world, as well as this, the COCO dataset is well established with over 200,000 labelled images over 80 categories.

The fruit database was chosen for testing due to the availability of the subjects (fruit), as well as their ease in positioning. This allowed for testing of the system on the previously identified camera, in a controlled environment. Fruit was also convenient for simulating the densely cluttered forest

[7] See Section 2.5.4 for more information about the changes added in the Mask R-CNN

environment, as they are often available in large numbers with little variance between the individual objects – much like the homogenous pine trees in New Zealand forestry plots.

The networks were tested using a batch script which loaded each model in turn and used to identify the current frame from the camera. This represents a test of the models' capabilities in a live situation, taking into account any transformations to the data from the format collected from the camera. This also means the FPS accounts for time taken and any decrease in performance due to memory usage by reading the camera stream.

Each scene was tested (consecutively) 30 times on each network, and the recorded FPS is the average (mean) of the times taken to collect each image, with the system reading system time before the image is captured, and subtracting that from the system time read when the classification learner provided a result (not when the result was recorded or displayed), to record the time taken to complete a single frame. The inverse of this being the number of frames capable of being captured in a single second; the FPS.

For tests on the fruit database, an instance of each fruit was tested with a slight overlap with another instance of every fruit in the database. This methodology allows for equivalent testing of every class, while also showing any abnormalities in the model's ability to separate certain classes. This means that each class of fruit was included in nine scenes – five at the front, and five at the back; with one scene having two instances of that fruit (hence total 9). This results in each fruit being included in a total 270 detections, from which the average accuracy and precision are calculated for the final results.

Each scene was tested 30 times to create a balanced result by reducing the influence a misclassification due to unexpected change in the testing environment (flashing lights, objects moving, etc.). These retests were not used for statistical sample analysis as this is not a standard practice in the field. The measure of accuracy, the mAP, is an average of an average, and as such further statistical analysis is unrequired.

The precision and accuracy of the bounding boxes on these images is measured with an IoU of 0.5. This number is what is used by TensorFlow when evaluating models in the training stage, and as such allows for better comparison of the two metrics. But also, it provides is a good compensation for the potential inaccuracy of the ground truth bounding boxes the models are tested against. This inaccuracy comes because the objects must be labelled by a human prior to testing; which, in comparison to the pixel-wise bounding boxes used in the training data, has a much larger margin for error.

Collection of tree data was also required during the project; however, this was managed by an external organisation in the industry as a part of the SFTI project. Due to the time required to collect and label this data for usage, testing with it was to be held off for the testing of the final proposed solution (see Section 4.1).

3.4.1 Training the models

All of the models were trained on a high-performance GPU enabled desktop computer using TensorFlow on python3. The TensorFlow libraries by Google are one of the most common machine learning tools with support and pretrained models for most popular object detection models such as MobileNet versions one and two, Inception version two (InceptionV2) with SSD and Faster feature extraction, and the Resnet 50 and 101 for both Faster and Mask.

TensorFlow also benefits from CUDA supported training, which significantly decreased the training time of every model – initial tests on the NUC suggested that training an effective model could take days rather than the hours it took on the desktop with the 1080Ti. This however, required configuring a training environment in Windows 10 (as the drivers for the desktop's 'gamers edition' GPU on Linux did not include fan controllers, which was deemed essential for the hardware's operational safety). Once these models were trained, they could be loaded into any environment, however performance would depend on the hardware available in the loaded environment. I.e. a network trained on a CUDA GPU will train faster than one trained on a CPU, but will perform equivalently (accuracy, precision, and speed) to a model trained with CPU only when tested on the same hardware.

The basic models were retrained on their respective datasets using transfer learning based from a COCO trained model. Transfer learning is a major feature of convolutional neural networks and allowed training of a functional model in a day on the hardware available, unlike the "one to two days on a single 8-GPU machine." taken to train the COCO dataset from nothing on the Mask R-CNN [8].

The two main features being tested were mAP (mean average precision), referred to as accuracy, which is based on the correctness of the predictions, and the order of their certainty. And the speed at which predictions are processed – or more accurately the number of frames which are calculated in a second – thus including image collection and transformation time from the camera's format.

For preliminary testing, the models were trained on a fruit dataset, where the objects being identified are on white backgrounds, originally designed for usage with feature extraction and object classification training. This should have made training faster overall, with higher precision in clean environments. However, this type of data can cause issues when training for real world applications, with the machine learning system not having been effectively trained to distinguish background and foreground images as shown in preliminary tests shown in Figure 24.

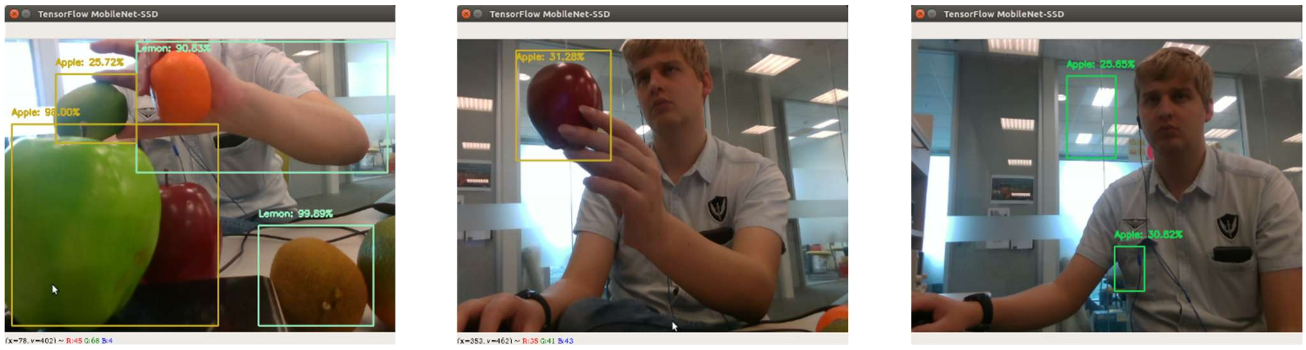


Figure 24 Initial tests at identifying fruit in a live environment on the MobileNet SSD trained to 10,000 iterations. The left image fails to separate the two apples clustered in the bottom right corner, misclassifies the kiwi as a lemon, and both misclassifies (also as a lemon) and fails to separate the orange from my arm. The central image shows the network correctly identifying the apple; however, the confidence is low at only 31.28% - for standard testing this is limited to 50%. The third image shows the network identifying items where none are present, again the confidence of these is low, this tendency, especially on the MobileNet SSD lead to the aforementioned increase in minimum reporting confidence.

The models will also be trained on the COCO dataset, which unlike the fruit database, is a complex dataset with thousands of images including backgrounds and overlapping objects. This increased complexity is expected to increase the training time, while decreasing training prediction accuracy, due to the increased complexity of the testing data. It is expected however, that this should provide a much higher level of accuracy when tested in a real world situation, as the training data includes real world environments which will better prepare the model for operating in such situations.

Using fruit for detection was considered an effective surrogate for trees for two main reasons. First was the availability for fruit in databases, as well as the fruit database, apples and oranges are present in the COCO datasets, they are also present in ImageNet with the addition of Lemons. This makes training and testing significantly easier as custom data collection is unrequired. The second benefit is the ease and availability of physical subjects for testing, unlike trees or other large subjects, fruit are small and readily available allowing for live image testing with multiple instances in a single frame, as well as being portable and easily configured to showcase multiple scenarios and test cases.

3.4.1.1 *Fruit Dataset*

To save on the amount of time spent training the network, several preliminary tests were run to evaluate the accuracy of the classifiers after a number or iterations. These tests were run using the MobileNet SSD as it was both fastest to train and run for evaluation. These test took place at 5, 10, 25, 50, and 100 thousand steps or iterations of training and simply involved holding the fruit in front of the camera and evaluating the ability of the camera to identify the fruit as objects, the level of misclassification of background objects, and the correctness (accuracy and precision) of the classifications.

The preliminary tests showed that models were able to separate objects from the background when trained to 50,000 steps and the minimum confidence filter was set to a prediction confidence of 0.5. While training the models ,100,000 steps may have produced more accurate results this test showed that difference was minimal while the training time was doubled from 5 hours 20 minutes to 11 hours, which was considered an unnecessary investment of time resources.

50,000 steps is a quarter of the TensorFlow default, however, due to the transfer learning from the COCO database trained models they were based off (which includes two of the fruit database's classes), the models already had valid weights (starting points for the training process) which only needed to be honed to the smaller and more specialised fruit dataset. As shown in the graphs in Figure 25, training slowed after 25,000 steps, with mAP not showing any significant (> 5%) increase until 40,000 steps. After which point overfitting occurred, to be slowly diminished due to the TensorFlow training engine; on the MobileNet this was around the 40,000th step and the InceptionV2 at 50,000.

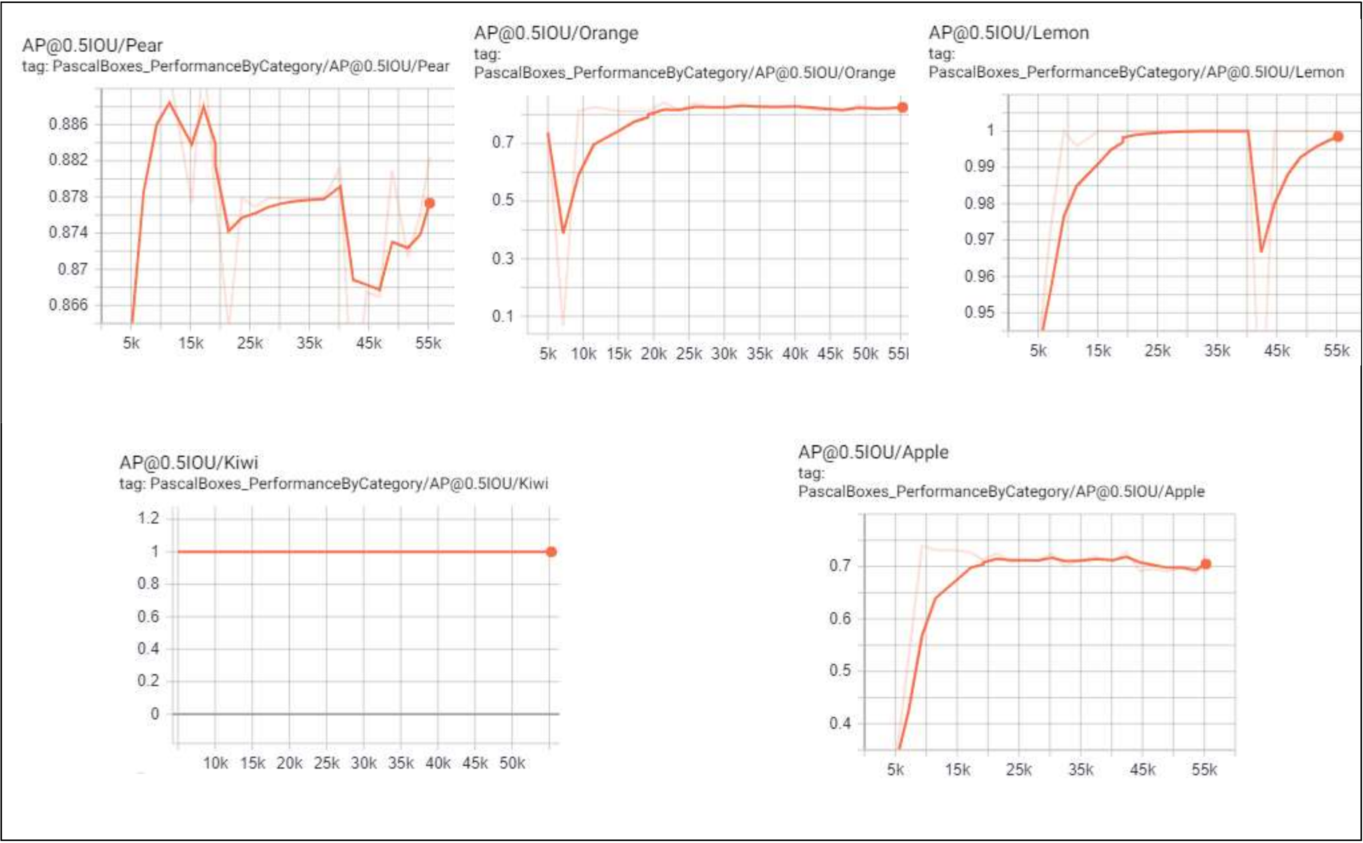


Figure 25 Average precision by category on MobileNet SSD trained on the fruit dataset. These graphs are outputs from the TensorBoard monitoring application, the scales and numbers used are therefore preconfigured which causes the inconsistency between graphs.

Table 4 Fruit Database Performance as measured by the TensorFlow evaluation process

Model	*mAP@0.5IoU*	*Trained to*
MobileNet SSD	0.8812	55,280 Steps[8]
Inception V2	0.8808	50,000 Steps
Faster R-CNN ResNet 50	0.9491	50,000 Steps
Faster R-CNN ResNet 101	0.8486	50,000 Steps
Mask RCNN ResNet 50	0.8932	50,000 Steps
Mask RCNN ResNet 101	0.8905	50,000 Steps

[8] Training for the MobileNet exceeded the standard 50,000, due to a configuration error and the haphazard nature of time-based saving. This is because the TensorFlow model export process runs in a separate time-based thread from the training process.

3.4.1.2 *COCO Dataset*

The models were pretrained by google (TensorFlow) on the COCO dataset. Therefore it was originally considered that further training on the dataset was unnecessary and would have resulted in overfitting. As such, the models were run through 5000 steps of training to create valid TensorFlow evaluation records. However, the original published models were used for the classification tests.

The evaluated mAP of the models trained on this dataset, as measured by the TensorFlow training algorithm is significantly lower than the same tests on the fruit dataset due to the complexity of the COCO dataset. However, theoretically this model complexity should enable the model to perform better in complex environments.

Table 5 COCO Database Performance as measured by the TensorFlow evaluation process – models trained by TensorFlow as a part of their 'Model Zoo'

Model	*mAP@0.5IoU*	*Trained to*
MobileNet SSD	0.4586	6.356M Steps
Inception V2	0.4794	5.014M Steps
Faster R-CNN ResNet 50	0.4867	9.528M Steps
Faster R-CNN ResNet 101	0.4674	9.188M Steps
Mask R-CNN ResNet 50	0.4028	10.369 Steps
Mask R-CNN ResNet 101	0.4012	6.897M Steps

3.4.2 “Faster Models”

The MobileNet SSD model presented by [11] in 2017 was designed, as the name suggests, for running on low performance smart phone (or “mobile”) devices. This model is suited for this series of tests as high accuracy predictions on low performance devices is the end goal of the thesis. MobileNet was recognized by a Google study [13] as one of the best balances between accuracy and efficiency and as such was a likely basis for the faster model which this project was hoping to develop. Its comparison, the Inception V2, is an advance on the Inception V1 otherwise known as the GoogLeNet; a model is known for its high precision while still maintaining a high accuracy [57].

Table 6 Average Precision of the tested fast models, with IoU@0.5. At this level of IoU the models trained on fruit are too inaccurate to detect any objects. The COCO trained models are more accurate and 100% of their predictions are correct.

	COCO			Fruit						Recall		FPS	
	Mean	Apple	Orange	Mean	Apple	Orange	Pear	Kiwi	Lemon	COCO	Fruit	COCO	Fruit
MobileNet SSD	1.00	1.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	12.15	9.01
InceptionV2	1.00	1.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	7.42	2.96

Both of the models tested in this section performed very poorly on the Fruit dataset, resulting in a complete failure to detect objects with an IoU of 0.5. For example, in Figure 26 the left image of has large bounding boxes which do not closely bound the objects being classified. However, the central positioning of the object within the bounds and the consistent margin of the boxes suggest that the model is accurately detecting (and classifying) the objects, just with a poor IoU.

Table 7 Precision of the tested fast models, with IoU@0.3 on the fruit dataset.

	Fruit						Recall	FPS
	Mean	Apple	Orange	Pear	Kiwi	Lemon	COCO	Fruit
MobileNet SSD	0.21	0.31	0.60	0.07	0.00	0.12	0.26	9.90
InceptionV2	0.37	0.90	0.79	0.43	0.63	0.32	0.41	3.12

To compensate for this lack of positive results, the minimum IoU requirement for tests on the COCO dataset was decreased to only 0.3 overlap between the prediction and the ground truth. This largely improved the results from the InceptionV2, which was able to clearly identify objects within the scene. However, it demonstrates that these simple models are unsuited for the intended purpose of robotic navigation, and any live navigation-based process, as they are unable to accurately bound objects such that an accurate measurement of the space required to navigate around them can be calculated. This is because the object in question only fills 33% of the predicted space.

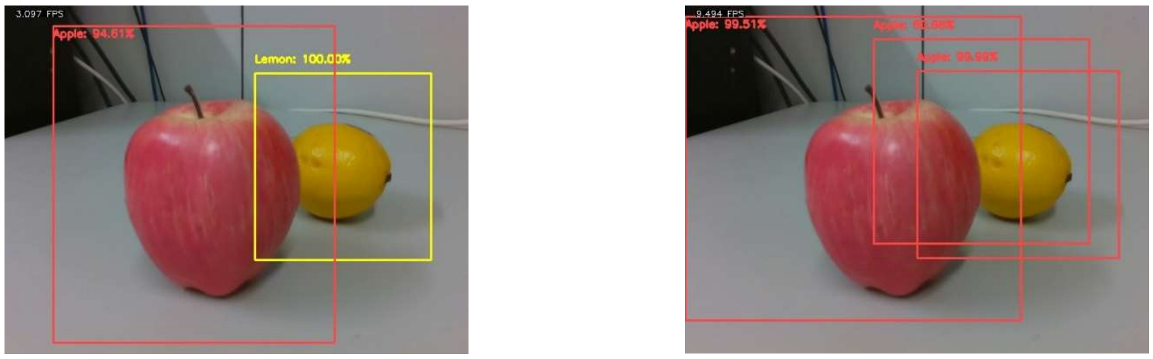


Figure 26 InceptionV2 (left) and MobileNet SSD (right) trained on the fruit dataset identifying an apple overlapping a lemon. The Inception network has correct identifications for both objects, with large ill-fitting bounding boxes, whereas the MobileNet has three incorrect bounding boxes with the apple identified with 99.51% confidence (label top left) being the only correct classification, however the bounding box is too large and unfocused to reasonably consider it an accurate detection within the apple's ground truth.

When trained on the COCO dataset, both models were able to accurately detect and classify both the orange and apple classes with ease, making no errors. They were also recorded to be significantly faster when detecting and classifying objects with a 34% increase on the MobileNet and an impressive 150% increase in speed on the InceptionV2. The reason for this is possibly due to an optimisation used by TensorFlow in the training of these models, but no particular information confirming this assumption is available. Furthermore, to get such a large performance increase on the InceptionV2 seems incredible and unlikely due to a simple optimisation.

3.4.2.1 *Discussion*

Both of these models showed high accuracy in testing, able to clearly identify, bound, and classify objects in the frames captured. However, with the Fruit trained database, the bounds were significantly looser and as such the results all fell outside of the IoU, resulting in the failure of the models. That the COCO trained models did not suffer from this issue, suggests that either a more comprehensive dataset is required for the accurate training of these models, or that more training steps were required to obtain the optimal results.

The processing speed of both of these models was very high with them both scoring above the minimum requirement of 5 frames per second. However, the MobileNet was 64% faster in the COCO test and 2.04 times faster when trained on the Fruit dataset. As such the decision was made to continue testing with this model as the simple model component of the final solution presented in Section 0.

3.4.3 R-CNN Models

While the InceptionV2 and MobileNet models in the previous section clearly showed how effective simpler models are in situations where the objects are clearly presented, they were unable to clearly identify objects in a densely populated area. This is theoretically where the slower and more complex Faster and Mask R-CNN fit in. With significantly more layers (50 or 101 on the ResNets in comparison to the 16 on the MobileNet) and more advanced feature extractors (ROI pooling) these models should be better at extracting objects and subsequently identifying them.

To test this, a benchmark was set with the Faster R-CNN which was trained in the same fashion as the MobileNet and InceptionV2. This benchmark shows the capabilities of the R-CNN technology with the given situations and sets expectations for the Mask R-CNN. The Mask R-CNN tested afterwards is expected to better bound the objects in frame, due to the mask segmentation layers, as well as the ROI pooling.

The ResNet 50 and ResNet 101 were chosen for testing the abilities of the Faster R-CNN without the additional technologies from the Mask R-CNN, as they are the most common models used with this architecture; being those demonstrated in the Faster R-CNN paper. They also are the two backbone models used with the Mask R-CNN, which means that comparisons can be made based on only the additions of the masking layers and changes to the ROI approaches.

The results from this test were interesting with the ResNet 50 outperforming the more complex ResNet 101 model. Furthermore, both of these models had a higher error level on the COCO database than the simpler models tested in section. It was also immediately apparent that the speed of these networks was a serious issue, with the faster of the two networks (the ResNet 50) taking more than two seconds to process a single frame.

Table 8 Results from the Faster R-CNN models compared with the simple models. The top section has the performances with an IoU threshold of 0.5 where the bottom has the performances at an IoU of 0.3

	COCO			Fruit						Recall		FPS	
	Mean	Apple	Orange	Mean	Apple	Orange	Pear	Kiwi	Lemon	COCO	Fruit	COCO	Fruit
ResNet 50	0.90	0.85	0.91	0.38	0.65	0.73	0.49	0.00	0.38	1.00	0.44	0.45	0.48
ResNet 101	0.76	0.99	1.00	0.44	0.85	0.00	0.86	0.39	0.34	0.76	0.50	0.35	0.39
MobileNet SSD	1.00	1.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	12.15	9.01
InceptionV2	1.00	1.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	7.42	2.96
MobileNet SSD				0.21	0.31	0.60	0.07	0.00	0.12		0.26		9.90
InceptionV2				0.37	0.90	0.79	0.43	0.63	0.32		0.41		3.12

These results show that the simpler MobileNet SSD and InceptionV2 models have a significantly higher FPS than the complex FasterR-CNN models, with the fastest of these, the ResNet50 having a framerate of only 0.17% of the slowest simple model; the InceptionV2 (as tested on the fruit dataset). However, comparing the accuracy of these models does not create a clear image of the model's features. In tests on the COCO dataset the ResNet101 has the worst performance, with a low mAP and being the only model not to get a perfect score for recall, while the MobileNet and Inception both have perfect mAP. On fruit database however the ResNet 101 is the best performing model, while the simpler models are unable to identify any objects at an IoU threshold of 0.5, and even with the decreased threshold, still have a lower mAP.

This difference in performance was due to the over prediction of objects in the frame, unlike the MobileNet and InceptionV2 which attempted to only identify the key objects in the frame, the ResNets attempted to identify complex objects and partial objects within the frames. This, for example, includes labelling the table as a "dining table", for which an exception was needed in the testing criteria, as while the object was unlabelled and unexpected from classification, the images were taken on a table, making the classification largely correct. Unfortunately, this increased detection of less obvious objects results in higher mistakes – for example the scissors, which were commonly identified over the apples in frame (as shown in below Figure 27)

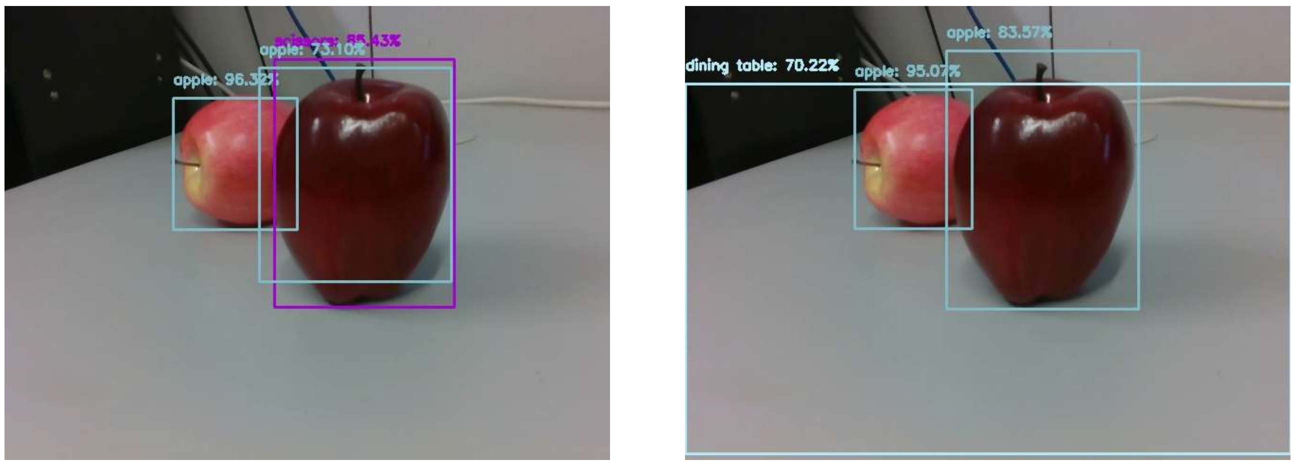


Figure 27 the ResNet models identified objects in the frame which were less obvious, such as the table (right). This however led to increased misclassifications such as the commonly mis-identified scissors (left).

3.4.4 Masked Models

The Mask R-CNN uses several methods that should create an increase in the accuracy (IoU) of the bounding boxes, and separation of classes. To demonstrate this, the tests with the Mask R-CNN models were run in the same conditions as those for the corresponding ResNet models. This includes using the same testing tool, which means the application did not include the ability to read and display the mask data, as it was designed for the evaluation of bounding box systems.

While a solution for displaying the results with the mask overlaid is common for these models, adding an unknown processing step creates space for additional unknown variables with the time taken for processing or model performance. This is particularly important as the method used for loading the mask model is initialised differently from that used by the non-mask models.

The results from the masked model tests show a decreased mean accuracy for the COCO tests, with the precision for both the apple and orange classes being lower than the Faster R-CNN versions on the same backbones. Like the Faster R-CNN models tested in the previous section, the more complex models had a higher precision for each individual fruit – meaning that the fruit was more likely to be correct if identified. Unfortunately, this was also linked to a decrease in recall and thus mean Average Precision, which considers recall.

These tests also display a decrease in frame rate by an average 61% with the mask models, decreasing the model speed to just under one frame every ten seconds on the mask R-CNN with the ResNet 101.

Table 9 Comparing the results of the Faster R-CNN models and the Masked models.

	COCO			Fruit						Recall		FPS	
	Mean	Apple	Orange	Mean	Apple	Orange	Pear	Kiwi	Lemon	COCO	Fruit	COCO	Fruit
ResNet 50	0.90	0.85	0.91	0.38	0.65	0.73	0.49	0.00	0.38	1.00	0.44	0.45	0.48
ResNet 101	0.76	0.99	1.00	0.44	0.85	0.00	0.86	0.39	0.34	0.76	0.50	0.35	0.39
Mask R-CNN (ResNet 50)	0.78	0.71	0.81	0.54	0.98	0.67	0.75	0.00	0.46	0.99	0.56	0.21	0.37
Mask R-CNN (ResNet 101)	0.62	0.75	0.84	0.12	0.14	0.55	0.06	0.00	0.00	0.75	0.13	0.11	0.24

The mask R-CNNs, however, do benefit from more accurate bounding boxes, managing to increase the IoU of object detection bounding boxes across all objects. The amount by which they did this however was very small, usually resulting in a decrease of a few pixels, which is not equivalent to the decrease in both recall and processing speed.

The difference in performance between the faster R-CNN and Mask R-CNN models was varied, with the mAP and class average precision being, especially for the fruit dataset, inconsistent between the models. While the Mask R-CNN with a ResNet 101 backbone had a very low average accuracy for identifying pears, the Faster R-CNN version was the most accurate. Inversely the Mask R-CNN ResNet 101 was able to identify oranges, whereas the Faster R-CNN was not.

In the COCO tests as well as the TensorBoard evaluation process for the COCO datasets, the masked models were shown to have a lower mAP than the non-masked alternatives. However, in the tests, the bounding boxes provided by the masking models were tighter to the objects in frame, as shown in Figure 28.

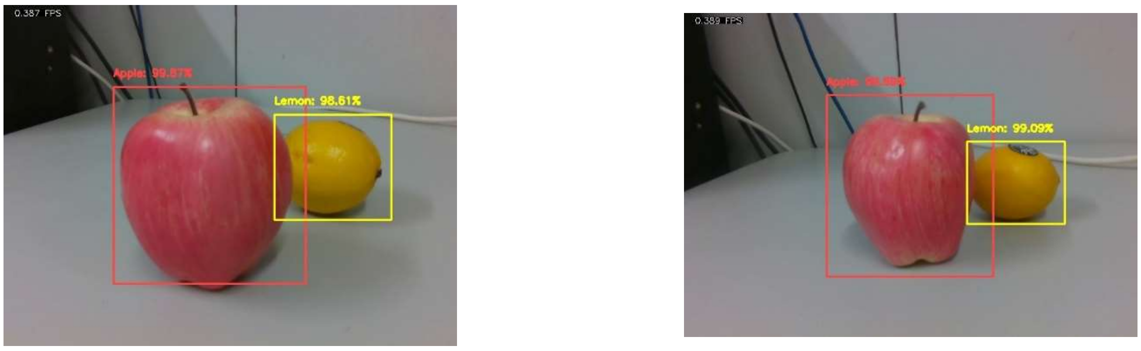


Figure 28 Mask R-CNN with ResNet 50 (left) and the Faster R-CNN with ResNet 50 (right). The bounding box for the apple is close to the ground truth on the Mask R-CNN than the Faster R-CNN with an IoU of 0.91 against the 0.83.

3.4.5 Summary of Results

The results from these tests were very interesting, with the MobileNet SSD and Inceptionv2 both performing well above the expectations; with perfect scores for recall and precision on the clear desk tests and the COCO dataset. However, as expected, the models were unable to accurately separate objects in a clustered environment. The MobileNet also demonstrated poor performance on the transfer learning from the COCO to the Fruit Dataset being altogether incapable of accurately identifying and localizing objects.

This suggests that the inception V2 is the best simple model, in situations where rapid detection is required. And its high accuracy on the COCO, and even fruit database (once the IoU threshold was lowered) suggest that it could be a very valid option for tree recognition. However, it would require significantly more training steps and data to effectively function. The MobileNet, however, was unable to accurately identify objects within the frame, despite the higher score in the evaluation data. This suggests a misalignment between the training and evaluation data, and the final tests; that the MobileNet was poorly suited to compensating for.

Despite the low accuracy, the MobileNet's high framerate made it the only option for continued testing as the Inception was unable to meet the minimum requirement of 5 frames per second in both tests. It was noted that while testing was continued, either more training, or a specialised dataset is required for the creation of a functional solution.

However, the ResNet based Faster R–CNN models were less effective in the simpler clear desk tests. This was due to their attempts to identify objects that were not clearly positioned in the centre of the frame, such as the desk. This resulted in an increased error level as the model predicted scissors within the reflection and lighting on apples. In general, the 101-layer version of the ResNet was less effective than the 50-layer version, with a lower recorded mean average precision for the COCO, and evaluation tests for both Faster and Mask R-CNN architectures.

The masked version of the Faster R-CNN models had, on average, a lower precision than the original versions of the networks, especially the ResNet 101 variations. The ResNet 50 trained on the fruit dataset being the only version to have higher mAP than its corresponding Fast R-CNN model.

Despite this however, the bounding boxes that were correctly positioned were more accurate with a lower IoU value. This low precision on the ResNet 101 based model is a potential issue for the thesis of this project; that a simple model can be augmented with the masking layers of the Mask R-CNN to increase the accuracy and precision of its predictions, as it was premised on the Mask R-CNN having very little to no effect on the precision of the backbone model; as is demonstrated in its accompanying paper [8].

3.4.5.1 *Discussion*

The evaluation tests for the models, when trained on the fruit dataset, would suggest that the identification of kiwi fruit and lemons are the most accurate at the set training level, while apple detection is the worst. However, during live testing, the kiwi fruit performed very poorly, not getting correctly identified even once by all but two of the models, most commonly these were misclassified as lemons. The oranges also suffered from misclassification, again, commonly being identified as lemons. This discrepancy between the evaluation and test results may be indicative of an imbalance in the number of records in the dataset; upon review, the lemon and kiwi have the lowest representation in the testing data at only 15 and 17%, apple has the highest at 30%. Furthermore only 18% of the total lemon and kiwi population are in the testing data, with apple being the second highest at 21% - pear was the first at 22%.

4 Final Model Test

4.1 Data Collection

Before the final tests could be concluded, working data had to be collected and annotated. This data was originally supposed to be provided by an external source as a part of the SFTI, and as such a low importance consideration requirement until this point. However, when received, the data was in an unexpected format (point cloud) with the only images being taken from a wide-angle panoramic camera. This resulted in a distorted image, that when compared to the data that would be collected on the previously tested, best suited, camera; the RealSense D435, would not be suitable for training a working model, as show below in Figure 29

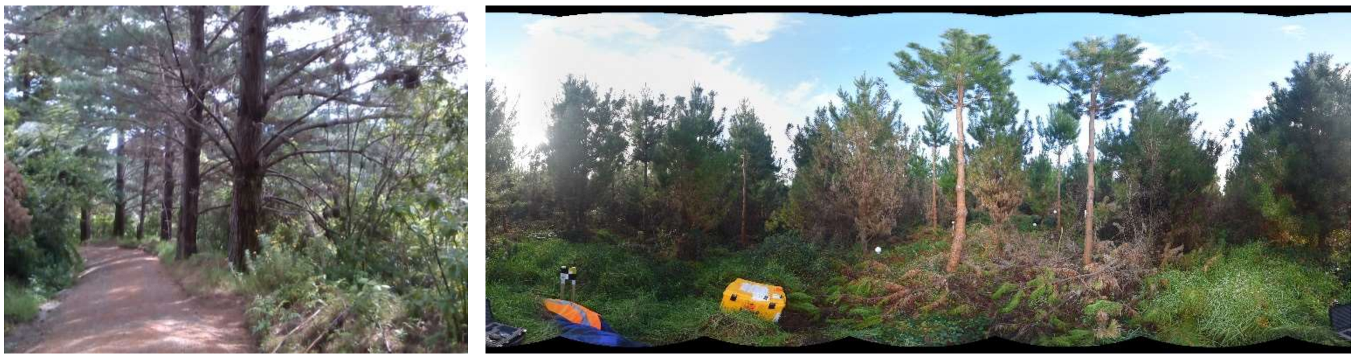


Figure 29 RealSense Image of forestry (left) and panoramic image of different forestry scene [55] (right) note warping of central tree; most noticeable in the bent trunk and "squashed" look of the upper branches

To train the model, an estimated minimum of 500 identified trees in functional images were required, this number is based on the failure of the fruit database (with 200 images of each object removed from any contextual background) with its low accuracy, tempered by the time constraints of collecting and annotating such data. To collect this data, a hand-held RealSense connected to a Surface Pro 3 tablet computer was used with a bespoke script set to capture images every 5 seconds. This set-up was walked through a local forestry block resulting in 125 usable images of (poorly maintained) forestry, with each image containing at least 5 trees.

There are two levels to the data annotation for this, the first is the bounding box which is used for the MobileNet and ResNet backbones instance recognition layers of the Mask R-CNN and Mobile Mask R-CNN, and the second level is an image mask outlining the trees, this is for the Masking layer of the models. To create the bounding boxes, an annotation software created by Oliver Batchelor was used [58], this software uses a machine learning algorithm to learn annotation as they are “submitted”, this streamlines the process by recommending annotations on future images which can be accepted or modified. As more images are annotated and processed, the model gets more accurate, requiring less time on the behalf of the user per image. This also served to show potential issues with the data, before training on the model of choice, the primary of these being the orientation of the bounding boxes. In this system, as well as most annotation and bounding box based sysetms, the bounding boxes are fixed in alignment with the top left and bottom right corners recorded only, creating the box from these points. This causes issues however, when the subjects in question are long, thin, and at a diagonal – in which case much of the bounded area is not actually the subject. This resulted in two common errors as shown in figure 8 below, first the objects in general do not fit within the constraints of their boxes, and secondly some trees have multiple boxes, attempting to cover the large diagonal area of the tree with the slim boxes expected from the average (vertically straight) tree.

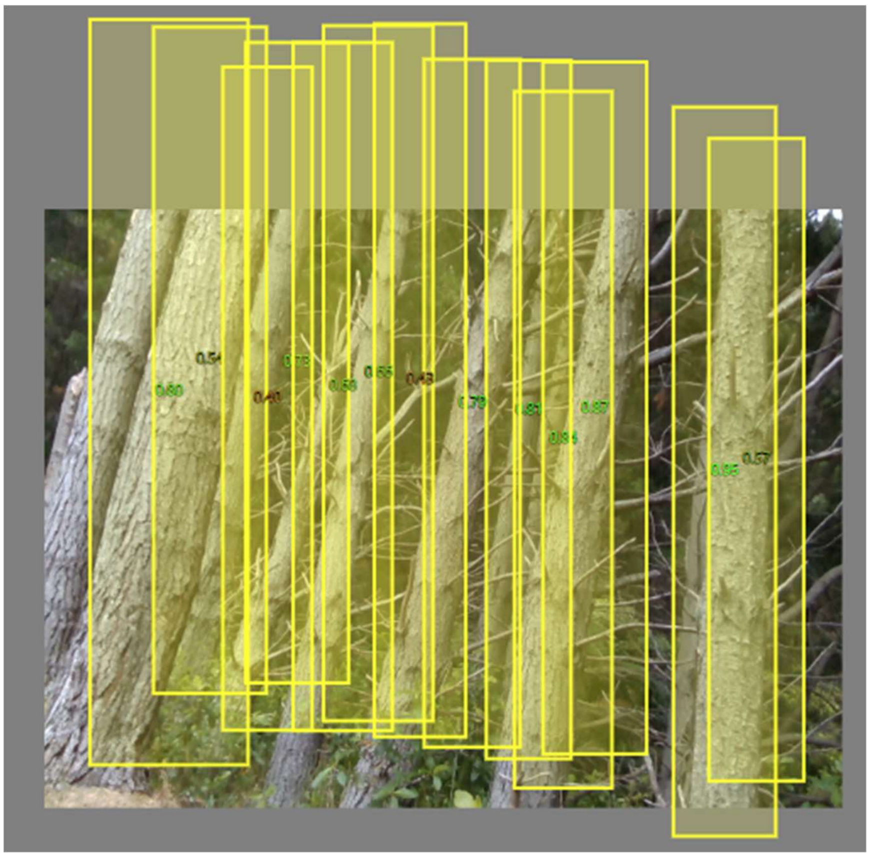


Figure 30 Trees on a diagonal cause the non-rotated bounding boxes to overlap and not fit correctly – image from Oliver Batchelor’s annotation tool [58], configured to show a minimum predicted certainty of 0.5

Creating Masks for this data however was an involved process and eventually took more time than was available to the project. To accurately mask a single tree, the trunk, branches, and any clearly identifiable foliage must be selected and recoloured to a single unique colour. This recolouration must be pixel perfect without any mixing which is a common feature in all compressed image formats such as jpg. Once every tree is recoloured in this method with a unique colour, the remaining pixels must be set to a solid, pure, black (value 0). And the resulting image saved as a PNG or uncompressed JPEG.

Once all of the image format mask files are created, a script is run over the folder, identifying each unique colour in the file and recording the maximum and minimum x and y values; these values become the bounding boxes for classification. For each individual colour in the file, a binary mask is compiled. This mask has equal dimensions to the original image, with areas where the pixel value is the same as the unique colour stored as 1 and all others being 0. These are stored as an array as individual objects for classification alongside the corresponding bounding boxes. These lists and files are eventually written into a .TFRECORD file using the TensorFlow library tools.

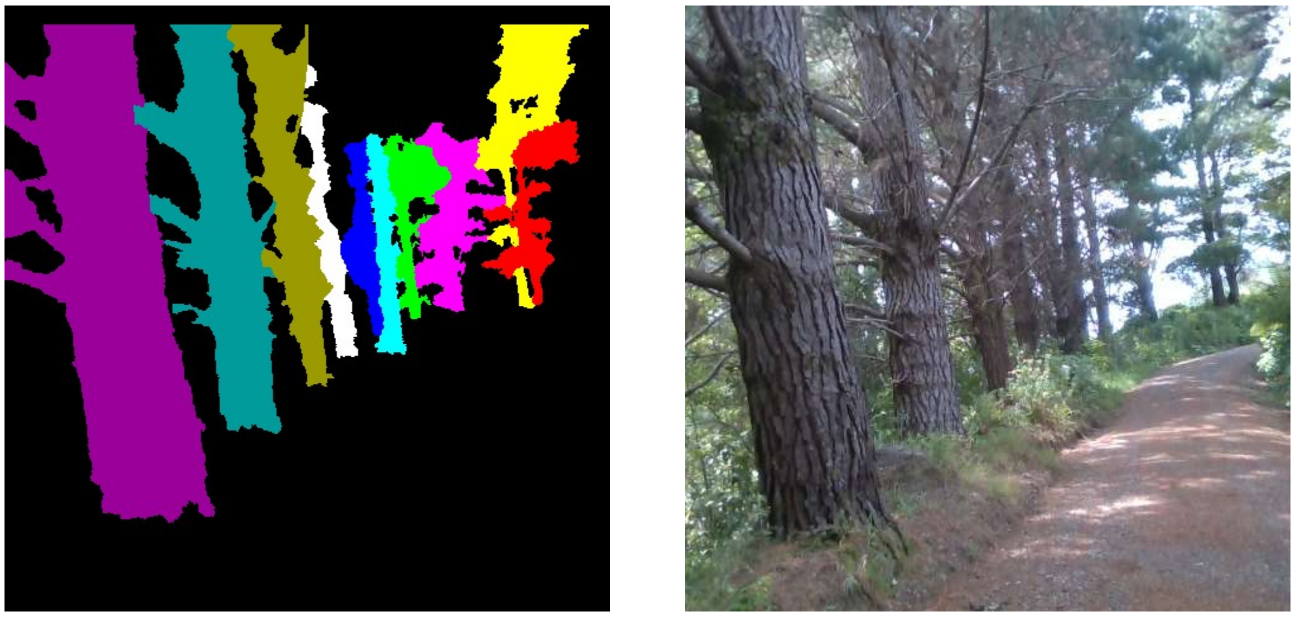


Figure 31 PNG mask image of a forestry scene requires each tree to be identified individually and recoloured on a black background.

4.2 Model Design

The final developed model is a Faster R-CNN based network, using the MobileNet as the backbone and feature extractor. This combination is expected to benefit from the significantly (1800%[9]) faster processing of the MobileNet, while the ROI pooling and masking layers from the Mask R-CNN architecture provide a better base for the extraction of meaningful object segmentation in cluttered environments.

In tests described section 3.4, the MobileNet SSD was found to be the fastest of the tested models on both datasets, achieving a maximum framerate of 9.90 FPS on the fruit database, and 12.15 FPS on the COCO dataset. On the COCO dataset this was 4.73 FPS faster than the next best model (the InceptionV2), and on the fruit dataset 6.05 FPS faster than that same model. Although it was fast, the MobileNet was inaccurate achieving a mAP of only 0.21 and an equally low recall of 0.26, and only when tested with a very low IoU threshold of 0.3.

In comparison, the Mask R-CNN was demonstrated to have a much higher mAP of 0.54 and a recall of 0.99 when tested with an IoU threshold of 0.5. But this massive increase in model accuracy results in a significant decrease in model speed, with the Mask R-CNNs only managing 0.37 frames per second on the fruit database.

Tests comparing the Fast and Mask R-CNNs show that the masking version of the networks has a lower mAP and recall than the original models on which they are based, but also improves the accuracy of the model's output; with very little decrease in the recall (0.01), and an increase in the IoU values of the bounding boxes created.

This model has a much lower memory usage, with only 16 layers in the MobileNet in comparison to the 50 in the lowest ResNet, and the decrease in both layers and memory usage, should enable the model to run significantly faster.

While the MobileNet is being used to increase the speed of the Mask R-CNN, the Mask R-CNN's ROI align, and masking layers, will help the model to better localise the detected regions from the MobileNet, and thus should be able to increase both the recall and accuracy of the model as a whole.

[9]Based on results from training on the fruit database, when compared to the fastest Faster R-CNN model, the ResNet50

4.3 Model Training

Training the Mobile Mask R-CNN was more difficult than the other models as there was no pre-trained dataset to base the training on. This meant that weights either had to be relearnt from zero, an unrecommended solution, or the weights from a similar model would be used, with the remaining weights trained from tabula rasa . This was the solution used with the original MobileNet being used for weights while the masking layer was forced to learn from the ground up.

The training for the Mask R-CNN was very irregular; taking a quarter of the time as even the MobileNet SSD, while still achieving expected results when evaluated. The reason for this is unclear as the training process is contained across many scripts built into TensorFlow, and without intimate knowledge of the training scripts, direct calls and subroutines as programmed by Google testing and reporting on this irregularity is limited. A potential cause for this might be in the way TensorFlow handles the R-CNN's threads, potentially allowing the network to better utilize the GPU threads. Furthermore, the smaller size of the MobileNet feature extraction might be better fitted for the GPU processing blocks, which could again significantly increase training times.

Table 10 COCO training results of the models

Model	*mAP@0.5IoU*	*Trained to*
MobileNet SSD	0.4586	6.356M Steps
Mask RCNN ResNet 101	0.3575	9.188 M Steps
Mobile Mask R-CNN	0.0966	500,000 Steps

Table 11 Fruit training results of the models

Model	*mAP@0.5IoU*	*Trained to*
MobileNet SSD	0.8812	55,280 Steps
Mask RCNN ResNet 101	0.8905	50,000 Steps
Mobile Mask R-CNN	0.1729	62,780 Steps

4.4 Test Results

The previous test demonstrated that the MobileNet performed as expected; providing accurate results in the COCO tests in non-cluttered environments, and the Mask R-CNN was able to accurately identify separate objects in a frame (albeit slowly). This simplified extraction and identification method within the CNN architecture is hoped to decrease the amount of time required to identify objects, while the masking layer and improved ROI proposals provides the contextual object segregation which should aid in cluttered environments.

Table 12 Performance of the various Mobile Mask R-CNN versions. The Final 500,000 + 50,000 model is the final test version, utilizing the 500,000 steps of training on the COCO dataset with an additional 50,000 steps on the fruit dataset

	COCO			Fruit						Recall		FPS	
	Mean	Apple	Orange	Mean	Apple	Orange	Pear	Kiwi	Lemon	COCO	Fruit	COCO	Fruit
Mobile Mask R-CNN 150,000 Steps	0.50	0.00	1.00							0.50		3.87	
Mobile Mask R-CNN 350,000 Steps	0.61	1.00	0.98							0.55		3.86	
Mobile Mask R-CNN 500,000 Steps	0.46	0.00	0.99							0.47		3.76	
Mobile Mask R-CNN 50,000 Steps				0.09	0.47	0.00	0.07	0.00	0.04		0.12		4.05
Mobile Mask R-CNN 500,000 + 50,000 Steps				0.43	0.73	0.77	0.49	0.00	0.41		0.42		4.20

The results from the Mobile Mask R-CNN show that the model is a competitive model in terms of achieving a balance of speed and accuracy. While the frame rate is 1100% higher than the Mask R-CNN models using ResNet, the measured accuracy on the fruit dataset is 0.07 higher than the InceptionV2 which had a 25% lower measured average FPS. Furthermore, the accuracy of the bounding boxes this model creates is higher than the InceptionV2, and it is able to identify fruit with the original testing IoU threshold of 0.5.

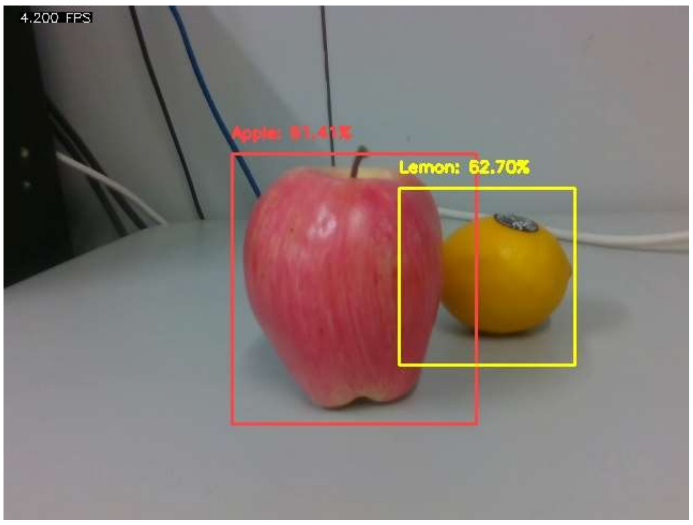


Figure 32 The Mobile Mask R-CNN shows high accuracy with its bounding boxes

4.4.1 Thesis Results Discussion

The Mobile Mask R-CNN performed with a slightly increased FPS in comparison to the Mask R-CNN and had a better mAP in cluttered environments than the MobileNet SSD which the model backbone was based off. However, it was still unable to achieve the desired FPS on the NUC, managing an average 4.2 frames per second on the best performing fruit configuration, 0.8 frames per second lower than the minimum 5 FPS set out in the robot design section (2.3.2). This suggests that a faster base model might be required to obtain the desired results, or a new technology or method altogether needs to be proposed. For future development, looking into YOLO based mask models might result in faster models. However, the incorporation of a vastly different model type into this solution might not provide the results expected.

The Mobile Mask R-CNN also did not achieve as high a mAP in busy environments as the original model, with the Resnet 50 backbone, which shows that the simpler model does have a significant effect on the accuracy of the model, even when supported by the ROI align and masking layers. This is expected, but when recommending potential solutions for research such as using a YOLO backbone, it is important to note that the model might lose a large amount of its accuracy, and thus become unviable.

While this project has explored a range modern models, hoping to create a solution for object detection in complex environments without relying on CUDA and GPU computational devices, the results have been poor when compared to the original Mask R-CNN, thus neglecting to solve the issue with satisfactory results. To properly solve this issue, it is likely that a better method of segregation and identification is required.

The experiments have indicated that identifying trees when separated from other object is simple (Figure 33, left), however when placed in a more common environment separating one tree from another becomes a difficult task (Figure 33, centre), more so when the forest is of a single tree type and cut (Figure 33, right). A solution to this issue might be the utilising of SLAM and object detection technologies, creating 3D scans of environments and matching them to 3D models like the work done by Gonzalez-Aguirre, et al. [59] in 2011.

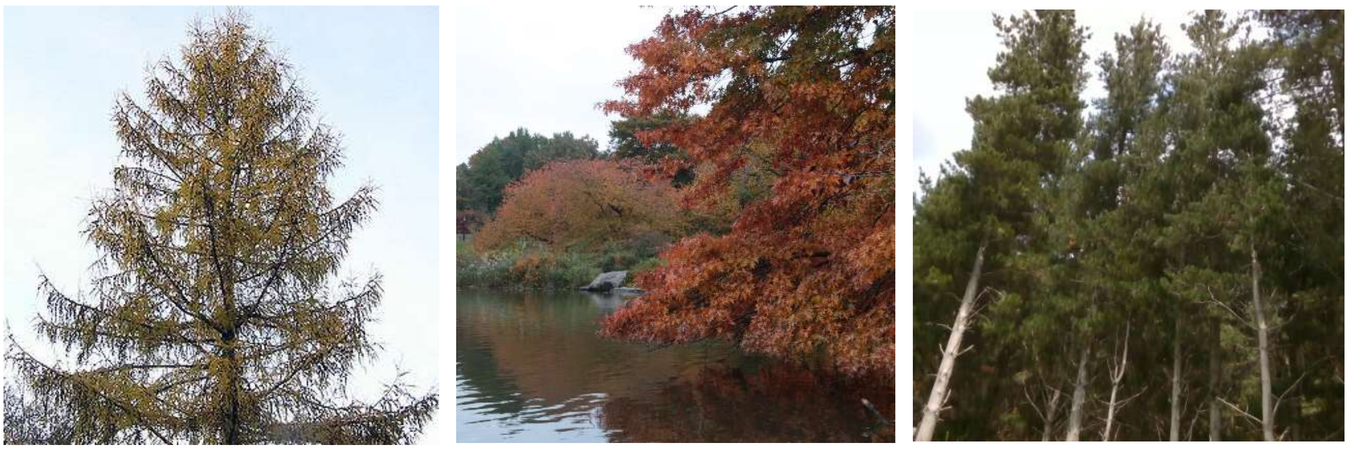


Figure 33 identifying trees is easier when the tree is alone, or at least different from its neighbour
(left) The Larch (Centre) Assorted Deciduous Trees in Central Park NY (right) Radiata Pines

5 CONCLUSION

This project has described a solution for image recognition on a physically small robotic unit, using minimal processing and simplistic models to allow this to work without the need for a GPU enabled device. This required analysing the hardware options, and object detection algorithms available and implementable on a non CUDA framework.

5.1 HARDWARE

There are several computational units available for usage in any robotic solution; ranging from laptops to single board computers. However, for the purposes of computer vision a higher performance device is normally required. For best results this needs a CUDA supported GPU however, with a simple enough model, such as the MobileNet SSD, any device with a standard multicore CPU can function.

To evaluate the capabilities of running neural networks on devices without a GPU, this project used an Intel NUC. For research purposes, this device has the benefits of using a laptop, or other personal computing device, meaning the device could be used for both the model testing and development processes, which was beneficial in that it streamlined the entire process. In testing, the NUC was unable to perform the object detection tasks at a framerate sufficient for the purposes of live object detection which was defined as a recommended 20 FPS with a minimum of 5 FPS.

There are many sensory devices available for robotic sensing and navigation, these include radars and lidar based technologies, as well as cameras and IR based depth cameras. These sensors each have advantages and disadvantages and the decision of what sensor to use depends upon many environmental variables. While radar has the ability to measure distances of objects while obfuscated by other objects, with shadowing, and lidar technology has higher accuracy in its measurements, camera and IR cameras are the most common sensors for robotics. This is in part due to their price, being significantly cheaper than both radar and lidar, but also due to the versatility of the devices.

Of the multitude of devices available, the Xbox Kinect and Intel RealSense product lines are some of the most common. The Kinect has two versions with the original being considered the first affordable option for a combined depth and colour camera device. And the RealSense is a newer development from Intel, released in 2015, to create a depth camera for IoT purposes, with a larger product range, also across three major versions; the 200, 300, and 400 series.

This project identified the RealSense D435 to be the better performing camera for the requirements of this project. However, it also identified several situations in which the Kinect V2 would be the better solution. The quality and resolution of both colour cameras was the same, with the main difference in the images taken being the field of view, with the Kinect having the higher of the two options. In comparison, the depth sensors on the devices have significant differences in the results collected, with the D435 having a higher accuracy for depth measurement at distances lower than 2m and a smoother representation of objects in general. The Kinect, however, tends to have a much higher level of noise in the data collected, and poor representation of flat objects on an angle, but a higher accuracy at ranges above 2m and significantly better results above 4m.

A deciding factor between these devices was in the physical design. The D435 is a small, compact device, well suited for mounting on a small unit. It also runs off a standard USB 3.0 to USB type-C cable, which allows for both power and data transfer. The Kinect is a significantly larger device, with several long and specialised cables and adapters required to connect the device to both the computational unit (again using the USB 3.0 for connection to the computer) and an external power unit.

5.2 Datasets

The problem of datasets became a major issue with the testing of the methods in this project. This project uses three datasets to test the models and methods critical to the evaluation of the technologies currently available. These include the COCO dataset, a fruit dataset, and a collection of tree images.

The COCO dataset is the most common evaluation tool in modern computer vision, with a large selection of images and annotated objects spanning 80 classes. The images in this dataset are collected from real-world environments with the objects being masked and bounded in natural positions. The Fruit dataset used however, is a simple dataset designed for image classification instead of detection, this means that the objects in the frame are clearly presented in the centre of the frame on a flat white background. The dataset also includes only 200 images of each of the five fruit types included in the dataset.

Tests with these datasets show that the increased complexity of the COCO dataset is absolutely required for the creation of a fully functioning computer vision solution. This is clearly shown in the tests on the MobileNet SSD which was unable to accurately separate objects from the background on the fruit dataset. The simplistic nature of the fruit dataset means that the networks trained on it are poorly optimised for identifying what are and are not objects of interest.

The tree images collected are the beginning of the tree dataset which could be used for creating a working tree identifier. However, collecting and masking quality images for this was a problematic and time-consuming task which eventually became a larger task than capabilities of the resources available of this project. To create an effective dataset, it was apparent that a large number of records, estimated at around 500 images per class, covering a range of situations was required; based on the short comings of the fruit dataset. While several records were collected it was deemed that these were insufficient for training a functional model, based on the results from the very limited (200 records per class) fruit dataset.

5.3 Neural Networks

The tests on the Neural networks had varied and conflicting results, showing that the simpler MobileNet and InceptionV2 models had a much higher accuracy than the more complex Faster R-CNNs on the COCO dataset. While the Faster and Mask R-CNNs had higher mAP on the fruit dataset.

On the COCO this was due to the simpler models' tendency to ignore obviously detectable objects, and thus probably a feature of their feature extraction methodologies. This feature would normally translate to a significantly decreased recall, as the networks will fail to identify objects that they have been trained on. In these tests however, the objects were placed centrally in a nearly empty environment and as such the recall was unaffected by this feature.

On the fruit dataset there was however, a large increase in the performance on the Faster R-CNN models, being able to identify and localise objects with a much higher IoU. These models have much more accurate predictions in comparison to the simple models, evident in the failing of the simple models to accurately localise a single object with the standard testing IoU threshold of 0.5.

The masking models tested displayed similar results to the Faster R-CNN models, with a slight increase of mAP (0.16) on the fruit database and a decrease of 0.12 in the COCO on the ResNet50 based models. They did, however, provide a slight, but clear increase in the IoU of bouding boxes, indicating the benefits of the masking and RoI pooling layers for object localisation.

The developed solution of this project, the Mobile Mask R-CNN, was based on using a Mobile Net as the backbone for the Mask R-CNN architecture. This solution suffered from the issues of both the MobileNet and the Mask R-CNN, having a framerate of only 4.2 (less than half the maximum of the MobileNet), and a mAP of 0.43 on the fruit dataset. Despite suffering from both low mAP and sub optimal framerate, the Mobile Mask R-CNN had higher performance than the InceptionV2 on the fruit database which only had a mAP of 0.37, and framerate of 3.12. Furthermore, it required a lower IoU threshold to achieve these results.

Despite its achievements in the fruit database, the Mobile Mask R-CNN had a significantly lower accuracy on the COCO dataset in comparison to all but the Mask R-CNN ResNet 101. This however might be a result of the level of training achieved on the model. The other COCO based models were all trained by Google for the TensorFlow model library, with access to large GPU server processors and long periods of time for training. This means the models were trained to an optimal level of several million steps presumably to the point at which they performed the best. In comparison the Mobile Mask R-CNN for COCO was trained on a home 'gaming' desktop and within a limited time period. These constraints mean that it was trained to a total of only 500,000 steps. Whether this lower training level influenced the results is unknown without further training, however as the model performed best at 350,000 steps rather than 500,000, it is likely that this is not the case.

5.4 Robotic Solution for Cluttered Environments

The Mobile Mask R-CNN shows the possibility for fast and accurate object detection for non-GPU enabled devices. However, the results achieved in this model are below the minimum criteria set out in the background section and as such this solution is deemed to be unsuccessful in creating a working solution. The major issue facing this technology is the model speed and achieving a sufficient framerate for 'live detection' for navigation, as the Mobile Mask R-CNN was only able to manage a testing framerate of 4.05 FPS, 0.95FPS below the minimum of 5 FPS.

For hardware, however, these tests demonstrate that there are many effective solutions for the sensing of the environment around a robotic unit. Both the D435 and KinectV2 had a high-quality colour camera, allowing for camera-based image recognition, and a depth sensor, each with their own strengths. For situations that require depth measurement over high distances (greater than four metres) the Kinect is the recommended device, with higher accuracy of measurements. The RealSense however is much more effective in shorter ranges (less than two metres) with significantly less noise and distortion on object representation and measurements.

To continue this line of research, more focus needs to be made on the capabilities and performing speeds of the more accurate models, or inversely the accuracy of faster models.

For the goals of the SFTI research project however, a different solution may be required. The solution created from this research is incapable of providing the required accuracy to operate within the intended environments, as such unless better methods are found, a new approach is required. An inquiry of research which was considered but not implemented in this particular project was to run only segregation based on the depth data and object tracking on the robotic unit, while the identified objects are sent via a wireless network protocol to a server system for identification. This could decrease the individual processing load on the small robotic units, while providing potentially more accurate situational awareness, of both the individual robot, and any other units working alongside it.

6 References

[1] SFTI, “National Science Challenges: SFTI,” [Online]. Available: https://www.sftichallenge.govt.nz/challenge-background.

[2] NZ Wood, “Forestry: Harvest and Manufacturing,” [Online]. Available: http://www.nzwood.co.nz/forestry-2/lc-forests-and-wood-harvest-and-manufacturing-1/. [Accessed 01 03 2019].

[3] C. Hegan, “Radiata, Prince of Pines,” *New Zealand Geographic,* no. 020, Oct 1993.

[4] M. McKinnon, *Volcanic Plateau places - Kāingaroa to Kaimanawa,* 2015.

[5] Ministry for Primary Industries, “Forestry in New Zealand,” Te Uru Rākau | Forestry New Zealand, 23 Jan 2019. [Online]. Available: https://www.mpi.govt.nz/news-and-resources/open-data-and-forecasting/forestry/. [Accessed 28 Jan 2019].

[6] CareersNZ, “CareersNZ: Forestry and Logging Worker,” 30 August 2018. [Online]. Available: https://www.careers.govt.nz/jobs-database/farming-fishing-forestry-and-mining/forestry/forestry-and-logging-worker/. [Accessed 23 February 2019].

[7] S. Edmunds, “Here's why no-one wants to plant trees for $400 a day,” *Stuff,* 21 January 2019.

[8] K. He, G. Gkioxari, P. Dollár and R. Girshick, “Mask R-CNN,” in *IEEE International Conference on Computer Vision*, Venice, Italy, 2017.

[9] Nvidia, “NVIDIA Jetson,” [Online]. Available: https://developer.nvidia.com/embedded/develop/hardware.

[10] N. Pinto, D. D. Cox and J. J. DiCarlo, “Why is Real-World Visual Object Recognition Hard?,” *PLoS Compution Biology,* vol. 4, pp. 333-341, 2008.

[11] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto and H. Adam, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” *CoRR,* vol. abs/1704.04861, 2017.

[12] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 779 - 788, 2016.

[13] 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,” Google, 2016.

[14] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Haung, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg and L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge,” *International Journal of Computer Vision (IJCV),* vol. 115, no. 3, pp. 211-252, 2015.

[15] Farm Forestry New Zealand, “NZFFA guide sheet No. 1: An Introduction to Growing Radiata Pine,” Farm Forestry New Zealand, 2007.

[16] M. Everingham, L. Van-Gool, C. K. I. Williams, J. Winn and A. Zisserman, “The Pascal Visual Object Classes (VOC) Challenge,” *International Journal of Computer Vision,* vol. 88, no. 2, pp. 303-338, 2010.

[17] J. Deng, W. Dong, R. Socher, L. -J. Li, K. Li and L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database,” 2009.

[18] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick and P. Dollár, “Microsoft COCO: Common Objects in Context,” *CoRR,* vol. abs/1405.0312, 2014.

[19] NZ Forest Owners Association Inc, “New Zealand Forest Road Engineering Manual,” NZ Forest Owners Association Inc, Wellington, 2011.

[20] M. Schimpl, C. Moore, C. Lederer, A. Neuhaus, J. Sambrook, J. Danesh and W. Ouwehand, “Association between Walking Speed and Age in Healthy, Free-Living Individuals Using Mobile Accelerometry—A Cross-Sectional Study,” *Plos One,* 10 August 2011.

[21] M. Adams, J. Mullane, E. Jose and B.-N. Vo, Robotic Navigation and Mapping with Radar, Artech House, 2012.

[22] L. Gallo and G. De Pietro, “Input Devices and Interaction Techniques for VR-Enhanced Medicine,” in *Multimedia Techniques for Device and Ambient Intelligence*, Springer US, 2009, pp. 115-134.

[23] W. Castro, J. Oblitas, M. De-la-Torre, C. Cotrina, K. Bazán and H. Avila-George, “Using machine learning techniques and different color spaces for the classification of Cape gooseberry (Physalis peruviana L.) fruits according to ripeness level,” 2019.

[24] J. Cunha, E. Pedrosa, C. Cruz, A. J. Neves and N. Lau, “Using a Depth Camera for Indoor Robot Localization and Navigation,” 2011.

[25] M. Tomono, “Building an object map for mobile robots using LRF scan matching and vision-based object recognition,” in *2004 IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04.*, New Orleans, LA, USA, 2004.

[26] Microsoft, “Windows Dev Center: Kinect for Windows,” Microsoft, [Online]. Available: https://developer.microsoft.com/en-us/windows/kinect. [Accessed 29 August 2018].

[27] G. Halmetschlager-Funek, M. Suchi, M. Kampel and M. Vincze, “An Empirical Evaluation of Ten Depth Cameras,” *IEEE Robotics and Automation Magazine,* pp. 67 - 77, March 2019.

[28] J. Tobin, L. Biewald, R. Duan, M. Andrychowicz, A. Handa, V. Kumar, B. McGrew, A. Ray, J. Schneider, P. Welinder, W. Zaremba and P. Abbeel, “Domain Randomization and

Generative Models for Robotic Grasping," in *International Conference on Intelligent Robots and Systems (IROS)*, Madrid, Spain, 2018.

[29] Intel RealSense, "LibRealSense," Github, 19 December 2018. [Online]. Available: https://github.com/IntelRealSense/librealsense. [Accessed 26 April 2018].

[30] A. Géron, Hands-On Machine Learning with Scikit-Learn and TensorFlow, O'Reilly Media, Inc., 2017.

[31] S. Pal and A. Gulli, Deep Learning with Keras : Implement various deep-learning algorithms in Keras and see how deep-learning can be used in games, Packt Publishing, 2017.

[32] T. Tuytelaars and K. Mikolajczyk, "Local Invariant Feature Detectors: A Survey," in *Foundations and Trends in Computer Graphics and Vision Vol. 3, No. 3*, 2007.

[33] C. Harris and M. Stephans, "A Combined Corner and Edge Detector," in *Alvey vision conference*, Manchester, 1988.

[34] D. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, Kerkyra, Greece, 1999.

[35] H. Bay, T. Tuytelaars and L. Van Gool, "SURF: Speeded Up Robust Features," in *ECCV 2006: Computer Vision – ECCV 2006*, 2006.

[36] J. Matas, O. Chum, M. Urban and T. Pajdla, "Robust Wide Baseline Stereo from Maximally Stable Extremal Regions," *Image and Vision Computing,* pp. 761-767, 2004.

[37] E. Rosten and T. Drummond, "Machine learning for high speed corner detection," in *Euproean Conference on Computer Vision*, 2009.

[38] F. Cabello, Y. Iano, R. Arthur, A. Dueñas, J. León and D. G. Caetano, "Automatic Detection of Utility Poles Using the Bag of Visual Words Method for Different Feature Extractors," in

Computer Analysis of Images and Patterns: 17th International Conference, Ystad, Sweden, 2017.

[39] A. Lee, “Comparing Deep Neural Networks and Traditional Vision Algorithms in Mobile Robotics,” Swarthmore College, 2016.

[40] K. Fukushima, “Neocognitron: A Self-organizing Neural Network Model,” *Biol. Cybernetic,* no. 36, pp. 193-202, 1980.

[41] D. H. Hubel and T. N. Wiesel, “Receptive fields, binocular interaction and functional architecture in the cat's visual cortex,” *Journal of Physiology,* p. 106–154.2., 1962.

[42] D. H. Hubel and T. N. Wiesel, “Receptive fields and functional architecture in two nonstriate visual area (18 and 19) of the cat,” *Journal of Neurophysilogy,* pp. 229-289, 1965.

[43] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE,* vol. 86, no. 11, pp. 2278 - 2324, 1998.

[44] A. Krizhevsky, I. Sutskever and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Network,” *Commun. ACM,* vol. 60, pp. 84-90, 2012.

[45] R. Girshick, J. Donahue, T. Darrell and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” *IEEE Conference on Computer Vision and Pattern Recognition,* pp. 580-587, 2014.

[46] R. Girshick, “Fast R-CNN,” *2015 IEEE International Conference on Computer Vision (ICCV),* pp. 1440-1448, 2015.

[47] S. Ren, K. He, R. B. Girshick and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 39, pp. 1137 - 1149, 2015.

[48] K. He, X. Zhang, S. Ren and J. Sun, “Deep Residual Learning for Image Recognition,” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 770-778, 2016.

[49] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” *CoRR,* 2015.

[50] A. Waleed, *Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow,* Github, 2017.

[51] ISO/TC 42, Photography, “ISO 12233:2017,” International Organization for Standardization (ISO), 2017.

[52] Society for Imaging Science and Technology, “Digital Capture Resolution Measurements,” [Online]. Available: https://www.imaging.org/Site/IST/Standards/Digital_Camera_Resolution_Tools.aspx?WebsiteKey=6d978a6f-475d-46cc-bcf2-7a9e3d5f8f82&hkey=f9040928-44d3-411a-9594-98ee6fd81e69. [Accessed 3 June 2018].

[53] S. H. Westin, “ISO 12233 Test Chart,” 16 April 2018. [Online]. Available: https://stephen-westin.com/misc/res-chart.html. [Accessed 30 May 2018].

[54] V. Sequeira, J. G. Gonçalves and I. M. Ribeiro, “3D Environment Modelling using Laser Range Sensing,” *Robotics and Autonomous Systems,* no. 16, pp. 81 - 91, 1995.

[55] R. Green, *Tree Data,* Christchurch: University of Canterbury, NZ, 2018.

[56] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu and A. C. Berg, “SSD: Single Shot MultiBox Detector,” *European Conference on Computer Vision (ECCV),* pp. 21-37, 2016.

[57] C. Szegedy, W. Lui, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, “Going Deeper with Convolutions,” *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 1-9, 2015.

[58] O. Batchelor, *Annotation Software,* University of Canterbury, NZ, 2018.

[59] D. Gonzalez-Aguirre, J. Hoch, S. Röhl, T. Asfour, E. Bayro-Corrochano and R. Dillmann, “Towards shape-based visual object categorization for humanoid robots,” in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, 2011.

[60] D. Cireşan, U. Meier and J. Schmidhuber, “Multi-column Deep Neural Networks for Image Classification,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Manno-Lugano, Switzerland, 2012.

[61] R. Horaud, M. Hansard, G. Evangelidis and C. Ménier, “An Overview of Depth Cameras,” *Machine Vision and Applications Journal,* p. 1005, 2016.

[62] D. Tran, “Building a Real-Time Object Recognition App with Tensorflow and OpenCV,” *Towards Data Science,* 22 Jun 2017.

[63] T. Bell, B. Li and S. Zhang, “Structured Light Techniques and Applications,” *Wiley Encyclopedia of Electrical and Electronics Engineering,* 2016.

[64] International Organization for Standardization, “All about ISO,” [Online]. Available: https://www.iso.org/about-us.html. [Accessed 28 05 2018].

[65] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, R. Jozefowicz, Y. Jia, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, M. Schuster, R. Monga, S. Moore, D. Murray, C. Olah, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu and X. Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems,* 2015.

[66
] F. Chollet and And Others, *Keras,* https://keras.io, 2015.

[67
] A. J. Davison, I. D. Reid, N. D. Molton and O. Stasse, “MonoSLAM: Real-Time Single Camera SLAM,” *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 29, no. 6, pp. 1052-1067, 2007.

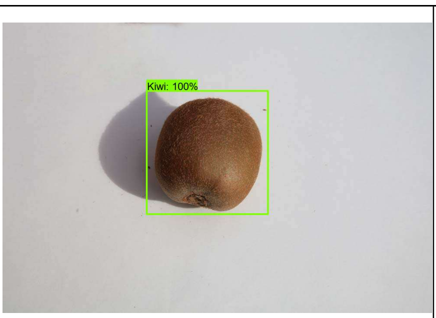
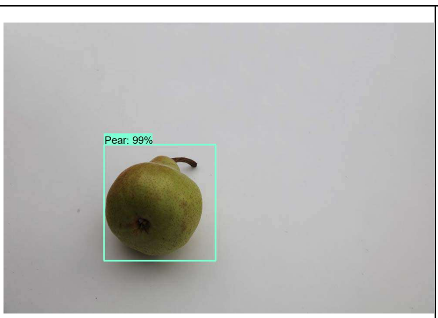
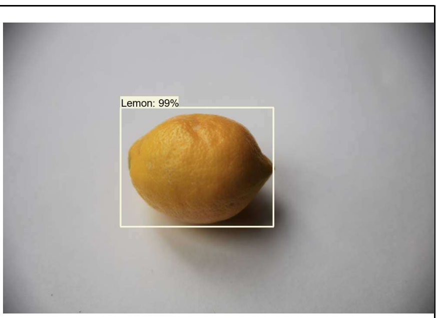
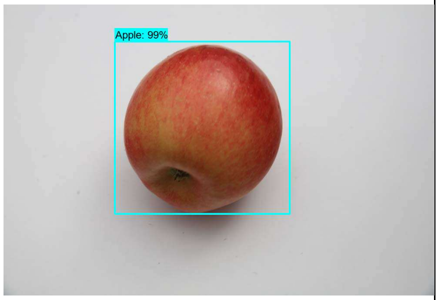
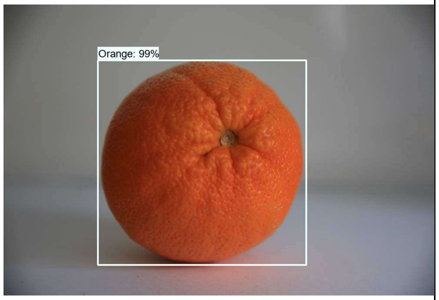
APPENDIX

A.1 TESTING DEVICE SPECIFICATIONS

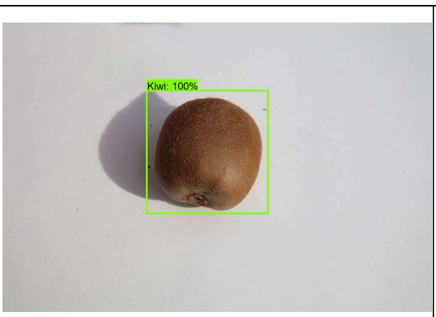
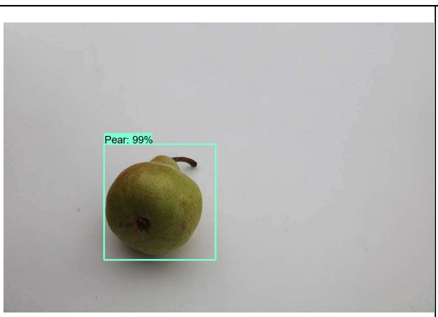
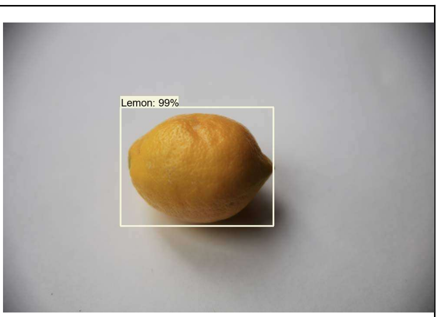
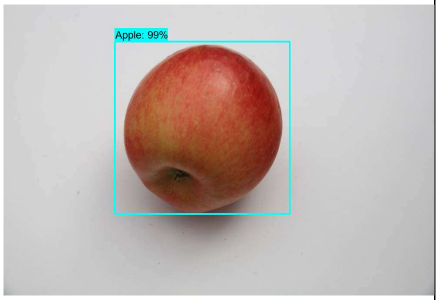
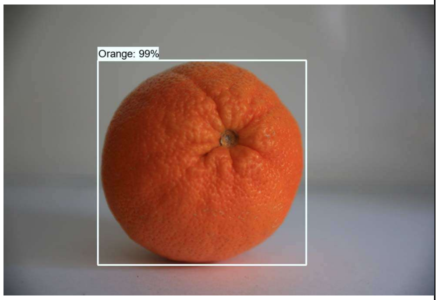
	Intel NUC (BOXNUC7i7)	Windows training desktop
CPU	Intel® Core™ i7-7567U CPU @ 3.50GHz, 4 Cores	Intel® Core i7-8700K @ 3.70GHz, 8 Cores, Overclocked at 4.70GHz
HDD	256GB HDD	4TB HDD
GPU	Intel HD emulated graphics (No dedicated GPU)	EVGA Nvidia GeForce GTX 1080Ti SC2, 11GB, CUDA 9.2
RAM	8GB	32GB DDR4, 2933MHz
OS	Linux Ubuntu 16.04 “Xenial Xerus” x64	Windows 10 Pro N x64

A.2 Training Images

6.1.1.1 *Fruit*

6.1.1.2 *COCO*

A.3 Test Results

Average Precision IoU@0.5

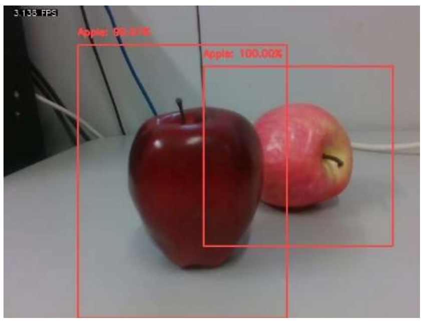
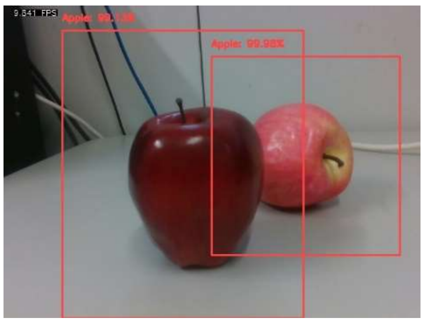
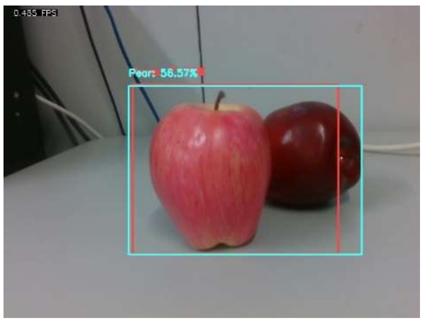
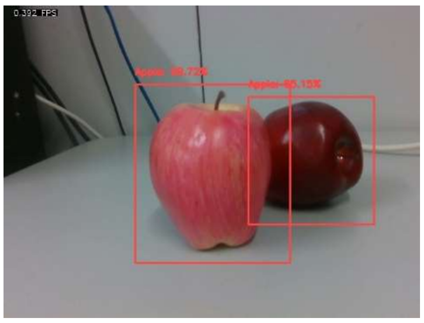
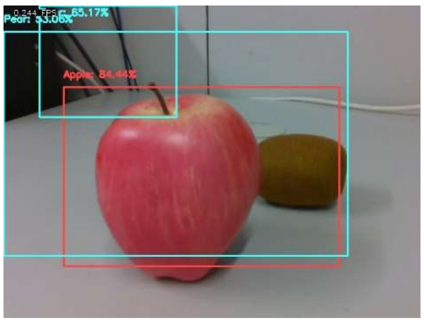
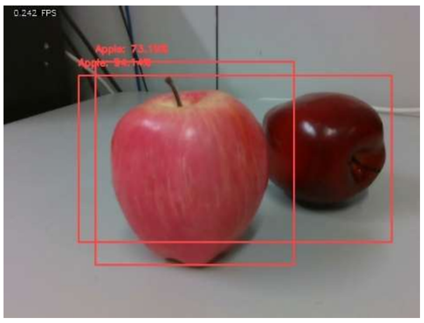
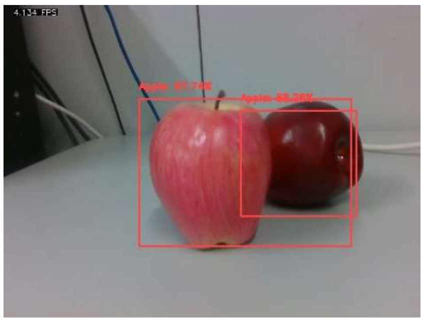
	COCO			Fruit						Recall		FPS	
	Mean	Apple	Orange	Mean	Apple	Orange	Pear	Kiwi	Lemon	COCO	Fruit	COCO	Fruit
MobileNet SSD	1.00	1.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	12.15	9.01
InceptionV2	1.00	1.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	7.42	2.96
ResNet 50	0.90	0.85	0.91	0.38	0.65	0.73	0.49	0.00	0.38	1.00	0.44	0.45	0.48
ResNet 101	0.76	0.99	1.00	0.44	0.85	0.00	0.86	0.39	0.34	0.76	0.50	0.35	0.39
Mask R-CNN (ResNet50)	0.78	0.71	0.81	0.54	0.98	0.67	0.75	0.00	0.46	0.99	0.56	0.21	0.37
Mask R-CNN (ResNet101)	0.62	0.75	0.84	0.12	0.14	0.55	0.06	0.00	0.00	0.75	0.13	0.11	0.24
Mobile Mask R-CNN 150,000 Steps	0.50	0.00	1.00							0.50		3.87	
Mobile Mask R-CNN 350,000 Steps	0.61	1.00	0.98							0.55		3.86	
Mobile Mask R-CNN 500,000 Steps	0.46	0.00	0.99							0.47		3.76	
Mobile Mask R-CNN 50,000 Steps				0.09	0.47	0.00	0.07	0.00	0.04		0.12		4.05
Mobile Mask R-CNN 500,000 + 50,000 Steps				0.43	0.73	0.77	0.49	0.00	0.41		0.42		4.20

Average Precision IoU@0.3

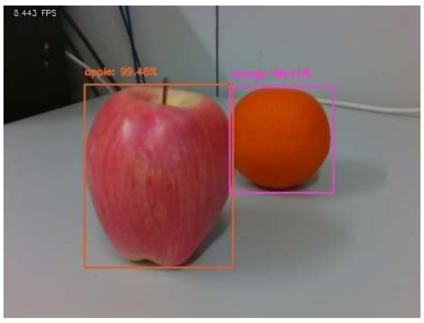
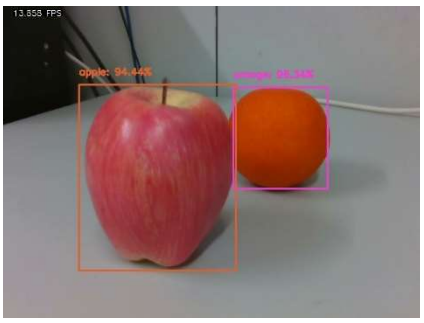
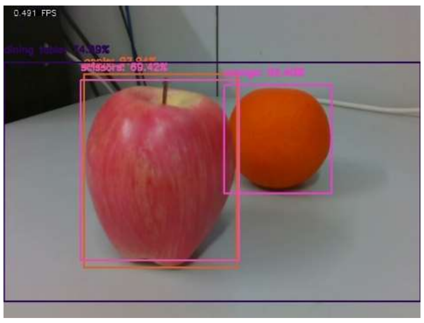
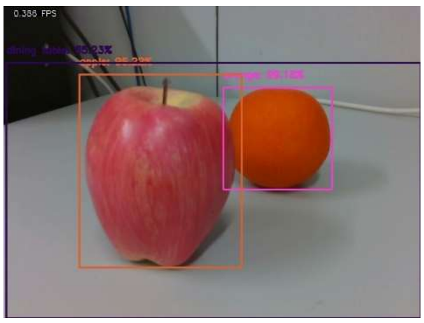
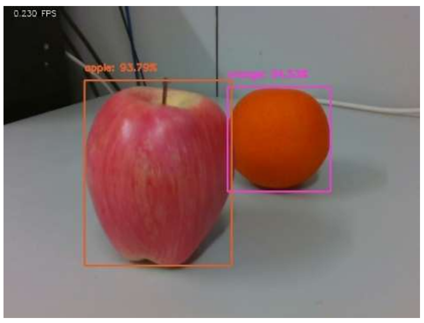
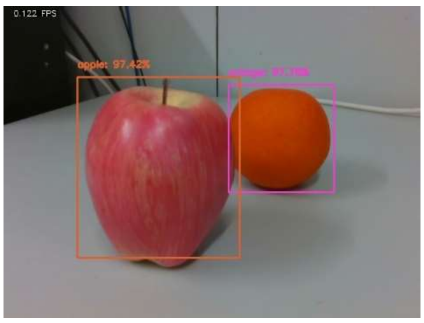
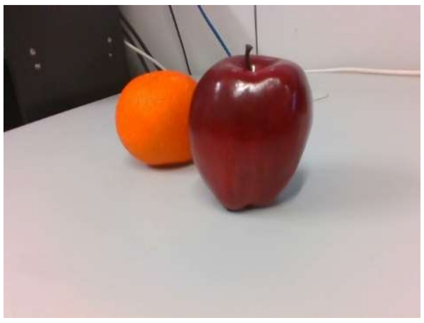
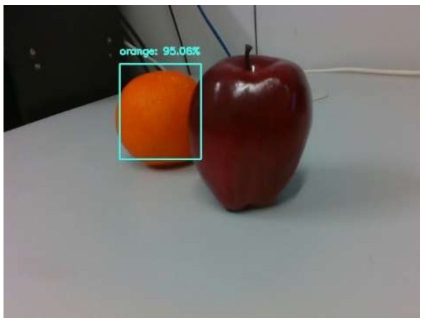
	COCO			Fruit						Recall		FPS	
	Mean	Apple	Orange	Mean	Apple	Orange	Pear	Kiwi	Lemon	COCO	Fruit	COCO	Fruit
MobileNet SSD				0.21	0.31	0.60	0.07	0.00	0.12		0.26		9.90
InceptionV2				0.37	0.90	0.79	0.43	0.63	0.32		0.41		3.12

A.3.1 Example Test Images

A.3.1.1 Fruit

 InceptionV2	 MobileNet SSD	 Faster R-CNN: ResNet 50
 Faster R-CNN: ResNet 101	 Mask R-CNN: ResNet 50	 Mask R-CNN: ResNet 101
	 Mobile Mask R-CNN (500 + 50 Steps)	

A.3.1.1 COCO

 InceptionV2	 MobileNet SSD	 Faster R-CNN: ResNet 50
 Faster R-CNN: ResNet 101	 Mask R-CNN: ResNet 50	 Mask R-CNN: ResNet 101
 Mobile Mask R-CNN (150 Steps)	 Mobile Mask R-CNN (350 Steps)	Mobile Mask R-CNN (500 Steps)