OBJECT DETECTION AND CLASSIFICATION FOR AUTONOMOUS ROBOTS

GENG5511/5512 Research Paper

Written by Alexander Arnold
Master of Professional Engineering (Software), UWA
Student Number: 21304455

Supervised by Professor Thomas Bräunl
School of Electrical, Electronic and Computer Engineering, UWA

Word count: 6122

Declaration of Authorship

I hereby declare that the thesis proposal submitted below is my own work.

All direct or indirect sources are acknowledged as references.

Alexander Rhodes Arnold

ARArnold

22/10/2018 . . . .

Contents

Abstract

With increasing global interest in semi- and fully-autonomous vehicle systems, the University of Western Australia has several research groups continuing to advance the capabilities of autonomous driving vehicles. This research comes with significant cost and risk, so a team was formed to research the effectiveness of embedded robotic vehicles (using the Raspberry Pi-based Eyebot platform) and determine the viability of a multi-robot autonomous driving system on the low-power, low cost hardware.

The navigation system of an autonomous robot requires the synthesis of data from a variety of sensors. The camera systems provide a significant amount of information, including the classification and location of objects, though this usually requires running expensive and complex algorithms. Therefore, the purpose of this thesis is to research, design and implement an object detection and classification algorithm that achieves high speed and accuracy while running on the computationally-limited Eyebot platform.

After researching, testing and comparing several object detection algorithms, it was determined that a mobile-specific neural network implementation would give the best combination of performance and efficiency.

This neural network was trained on a relatively small dataset, with impressive accuracy for its speed when compared with other researched algorithms. The solution provides real-time detection of hazards such as other vehicles, as well as the ability to recognise different traffic signs, with speeds approaching real-time detection (approximately 4.5 frames per second).

1. Introduction

Autonomous vehicles are vehicles that operate with higher levels of automation, allowing them to function without human intervention (ADVI 2018). While still a relatively new technology, they are becoming more prominent in numerous industries, such as mining (Rio Tinto 2017) and transportation (nuTonomy 2017). By using numerous sensors, such as cameras, laser scanners, and proximity sensors (Waymo 2017), the car is able to observe its environment and make decisions regarding navigation and collision avoidance. These sensors are often expensive, with LiDAR scanners for Google's autonomous car costing around $70,000 US (ArsTechnica 2017)

Mainstream media coverage of major breakthroughs in autonomous vehicle technology has propelled the topic into the forefront of social consciousness. However, autonomous vehicles (especially self-driving cars) have been met with hot debate around key topics such as safety, ethics, cost, and legal responsibility. These debates have been exacerbated after several fatal car accidents involving self-driving cars, such as a driver who was killed after his Tesla collided with a truck (Tesla 2016) and a pedestrian who was killed when crossing the road with her bike by a self-driving Uber car (National Transport Safety Board 2018).

These incidences demonstrate the complexity of both the problem and solutions, and the significant risks involved in developing autonomous vehicles. These risks can be mitigated, if not removed, by creating small-scale variants of these vehicles and testing them in various environments, both physical and simulated.

1.1 Small-Scale Implementations

With the high risk involved in exploring and creating autonomous vehicle solutions, many car companies, industry bodies and universities turn to scaled-down implementations to develop their concepts. Audi is one of many vehicle manufacturers that has allied with local universities, forming a competition between eight German universities where participants create an autonomous driving system (Audi Autonomous Driving Cup 2018b). Teams receive two scaled-down Audi Q1 vehicles fitted with five ultrasonic distance sensors, two cameras (front and rear), accelerometer, wheel speed sensors, and a LiDAR sensor (Audi Autonomous Driving Cup 2018c). The vehicles also come equipped with desktop computer-level hardware, similar in performance to the hardware available on the real Audi Q1 (Audi

Autonomous Driving Cup 2018a). The vehicles come with an API used for accessing this hardware and are tasked with building a control system that allows the car to autonomously navigating a pre-made environment complete with road signs, intersections, parking areas, and other cars (AADC 2018 Rulebook 2018).

While these model cars use cheaper and far less complex hardware than the vehicles they are modelled on, they still use expensive components such as LiDAR and desktop graphics cards (Audi Autonomous Driving Cup 2018a, Audi Autonomous Driving Cup 2018c). To complete a thesis with such hardware would be prohibitive, due to the limited number of vehicles afforded with the given budget. However, the Robotics and Automation Lab at the University of Western Australia has a large number of EyeBots, custom-made robots based around the cheap and readily-available Raspberry Pi 3 platform (Robotics UWA 2008). These robots have three infrared-based proximity sensors, a front-facing camera, and a differential drive wheel configuration, keeping the cost down while still providing similar functionality to the Audi robots. The robots have their own high-level API for reading sensor data, driving the robot, and basic image processing. Additionally, a cross-platform simulator (called EyeSim) is also available that can be used to virtually test the robots (Robotics UWA 2017). Therefore, any successful autonomous control systems developed on the EyeBots should be able to run on much cheaper hardware than what is required for the real-life Audi Q1, potentially paving the way for low-cost consumer autonomous vehicles in the future.

1.2 Autonomous Vehicle Control Systems

The control system for an autonomous vehicle is complex, consisting of many subsystems. They can be categorised into three types: perception, decision, and manipulation (Behere and Törngren 2016). Manipulation consists of the components involved in the motion of the vehicle such as steering and braking and is used by the decision-making system to operate the vehicle. In most cases, the manipulation systems are already implemented in vehicles that have manual controls, such as cars and boats.

Decision-making systems consist of the components that are concerned with vehicle behaviour given its external environment (Behere and Törngren 2016). This includes components such as path planning, localisation, collision avoidance, and object recognition (Herbert 1988). Localisation and object recognition are key to determining what possible

routes are available when path planning and are also heavily reliant on the perception capabilities of the vehicle.
Perception systems consist of both the components that are concerned with the gathering of environmental data and the components used to interpret the data at a higher level (Behere and Törngren 2016). This includes components such as proximity sensing, LiDAR imaging systems, object tracking, image processing, object detection, and object classification (Agarwal et al. 2017). While the first two perception components listed simply gather data from the environment, the latter components extract and interpret information from raw sensor data (in this case, a camera). The information from these components is used to create a "world model" (Behere and Törngren 2016), which is vital to the functionality, accuracy and performance of the decision-making components.

1.3 Project Scope

As this project is large and complex, it will be undertaken by a team of four Masters students: myself, Nicholas Burleigh, Bin Cui, and Yang Zhang. Together, our goal is to create an autonomous robot system for use on the EyeBots, capable of navigating environments and overcoming obstacles similar to what is present in the Audi Autonomous Cup competition. To do this, we will break the work down into its components – Yang will create a lane- and intersection-detection subsystem to assist with navigation, Bin will create a localisation subsystem, Nicholas will work on path planning and navigation, and I will create a subsystem to track, detect and classify objects with the camera for use with navigation and path planning.

My intended contribution is a high-level C++ API for object detection, tracking and classification designed for use with the EyeBots and the Robios 7 API. This API will be designed with efficiency in mind (due to the limited compute power of the EyeBots) and will be used by other team members for path planning, navigation, and decision-making tasks within the autonomous control system.

2. Literature Review

In this section, the existing literature is reviewed to identify various object detection and classification methods. Firstly, the more common and well understood algorithms are explored, followed by a review of the emerging algorithms explored in more recent papers.

2.1 Existing Solutions

The project requires the detection and classification of the objects around the EyeBots, using the available sensors. Unlike the single forward-facing infrared sensor, the camera gives a significant amount of information about the objects in front of the vehicle. Image processing and computer vision techniques can therefore be used to extract information about the objects in the camera's view.

Common computer vision techniques employed to extract objects from an image use a classifier trained on features extracted from the image. Traditional solutions include using a support vector machine (SVM) trained on HoG (Histogram of Orientated Gradients) features extracted from training images (Dalal and Triggs 2005). This an optimised algorithm with good accuracy and performance, however it is a binary object detector, requiring a new detector for each new object (Dalal and Triggs 2005). For many classes, such as the case with an autonomous vehicle object detection system, the large number of SVMs would outweigh any performance gains. However, newer solutions based on convolutional neural networks have been developed in the last decade which can detect multiple different classes in a single network.

Convolutional neural networks (CNNs) are a type of deep, artificial neural network that are capable of processing image data. The network uses its training to learn what filters to apply to the image, as opposed to relying on a pre-specified algorithm. However, they cannot be used to classify multiple objects within an image, since they cannot distinguish between separate regions of the image. Region-proposal CNNs (R-CNNs) were designed to solve this issue by incorporating a region proposal algorithm based on selective search (Girshick et al. 2014). This method proposes almost two thousand regions (Girshick et al. 2014), which gives the algorithm a high accuracy but causes the runtime of the algorithm (at up to a minute per image) to be far too slow for real-time object detection (Girshick 2015).

Several improvements have been made to R-CNNs in the years following Girshick et al.'s 2014 paper. The first was the development of the Fast R-CNN, which addressed the runtime performance and training time of the original implementation. This was achieved by calculating the convolutional feature map before processing the proposed regions to improve detection speed (Girshick 2015), and also by incorporating bounding box regression into the network training to reduce total training time. This resulted in a performance increase of over 150 times compared to the original R-CNN implementation (Girshick 2015). The next major development came in the form of Faster R-CNN, which replaces the selective search algorithm used in previous implementations with a small convolution network called the Region Proposal Network (Ren et al. 2017). This resulted in an average runtime of just 0.2 seconds for the deepest tested network with an improved accuracy over Fast R-CNN (Ren et al. 2017).

With the limited hardware considerations of the Raspberry Pi-based EyeBots, and the high-performance hardware used to test these R-CNNs (Girshick 2015, Ren et al. 2017), it is important to consider alternative algorithms that sacrifice some accuracy for extra performance. One such algorithm is the Single-Shot MultiBox Detector (SSD), a feed-forward network convolutional neural network which generates scores for the presence of objects at each of a default set of bounding boxes (Liu et al. 2016). This eliminates bounding box proposals and subsequent resampling, resulting in an approximate seven times increase in performance compared to Faster R-CNN with little change in accuracy (Liu et al. 2016).

2.2 Emerging Solutions

The next major breakthrough in convolutional neural network research came with the release of "MobileNets" – efficient networks designed by researchers from Google for mobile and embedded vision applications (Howard et al. 2017). They are based on factorising standard, high-dimension convolutions into lower-dimension depthwise and pointwise convolutions. These lower-dimension convolutions are far more efficient, giving a reduction in computation of approximately eight to nine times when compared to regular convolution methods, and with only a small reduction in accuracy (Howard et al. 2017, figure 1). When compared to other networks, MobileNet was able to match alternatives such as GoogleNet, Squeezenet and VGG16 in accuracy with significant improvements in speed (Howard et al. 2017).

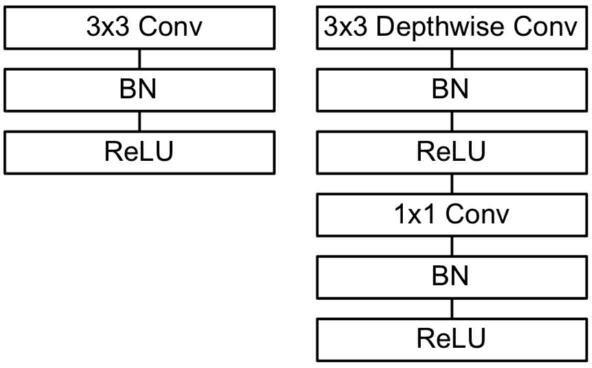
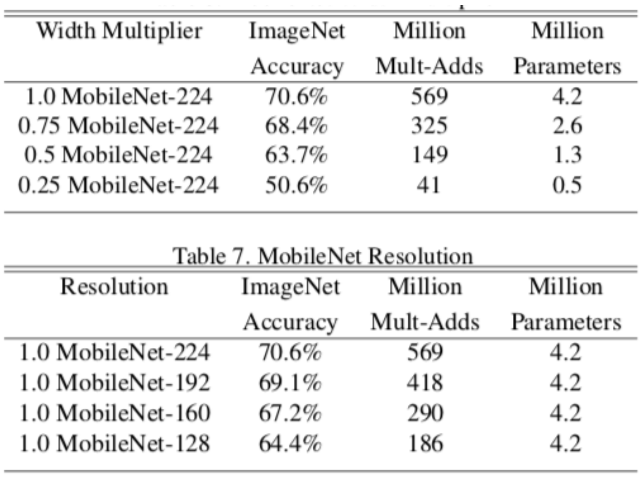


Figure 1: Comparison of standard convolution layer with the proposed MobileNet layer (Howard et al. 2017)

MobileNets were designed for easy modification with two hyperparameters: the width multiplier and the resolution multiplier. The width multiplier (α) is designed to uniformly thin out the network at each layer and ranges from 0 to 1, though it is typically set to a value of 1, 0.75, 0.5 or 0.25 (Howard et al. 2017). This multiplier effectively reduces the number of computations as well as the number of parameters by roughly α^2. The resolution multiplier (ρ) is used to reduce the dimensionality of the layers in the network (Howard et al. 2017). It also ranges from 0 to 1 but is implicitly set so the input resolution of the network is 224, 192, 160, or 128. Reducing the resolution multiplier also has the effect of reducing the number of computations by ρ^2. A comparison of the effects of these two hyperparameters on accuracy can be seen in Figure 2 below:



Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

Table 7. MobileNet Resolution

Resolution	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
1.0 MobileNet-192	69.1%	418	4.2
1.0 MobileNet-160	67.2%	290	4.2
1.0 MobileNet-128	64.4%	186	4.2

Figure 2: MobileNet accuracy compared to width and resolution (Howard et al. 2017)

This architecture was later improved upon by the researchers at Google to produce MobileNet v2, which improved performance by 30-40% while also increasing accuracy (Sandler et al. 2018, Sandler & Howard 2018>. This is done by changing the convolutional architecture from two to three layers – input data is fed into an expansion layer, which outputs high-dimensional features to a lightweight depthwise convolution layer, before being projected back down to a lower dimension with a linear convolution layer. This architecture preserves more information when compared to MobileNet v1's non-linear convolutions (Sandler et al. 2018), resulting in the increased accuracy and speed.
In implementing the MobileNet v2 architecture, the paper's authors demonstrated how it could be paired with the Single Shot Detector (SSD) algorithm, using MobileNet v2 as the basis for the neural network component. This allows the combined MobileNet-SSD architecture to detect objects within an image instead of just classifying images, retaining the speed and accuracy characteristics of both MobileNet and SSD (Sandler et al. 2018). Furthermore, the paper takes the improvements to convolutions in MobileNet v1 (replacing regular convolutions with depthwise and pointwise convolutions) to the SSD architecture, producing a much less computationally expensive variant dubbed SSDLite. This network reduces the number of parameters by a factor of seven, and reduces the computational cost by over three times (Sandler et al. 2018), making it one of the fastest object detectors for the given accuracy.

3. Design

In this section, the design constraints are identified and justified, and a methodology is presented to evaluate the effectiveness of a solution. The design process is then detailed, explaining the decisions behind architecture selection and the final system design.

3.1 Design Constraints

The design is primarily constrained by the available compute power. While the Raspberry Pi 3 platform offers significant performance for the price, it falls short of modern mobile phone processors and far behind regular PC capabilities. Additionally, there is little room in the budget to expand the Raspberry Pi's compute capabilities, either through the use of more powerful alternative compute boards to the Raspberry Pi or through external hardware modules such as the Intel Movidius Neural Compute Stick (Intel Developer Zone 2018). Therefore, the solution must be efficiency-focused, to ensure that the EyeBots can detect objects in real time.

Accuracy must also be a major factor in the design – in particular, making sure that the number of false negative detections is kept to a minimum. This is because an incorrectly classed object is generally (but not always) more useful than a missing one – a car being classified as a truck is generally safer than a car that is not detected, as this still gives the navigation system the ability to avoid a collision. A high accuracy will ensure that the object detection subsystem can be relied upon by the navigation subsystem when other sensors and systems have limited world information.

Regarding interactions with other subsystems, the solution is required to be programmed in C++ to work with the existing C/C++ RoBIOS API for the EyeBots (Bräunl, Keat and Pham 2018). Furthermore, due to the limited free space on the EyeBot SD cards, the size of most modern computer vision libraries, and the reliance of other subsystems on OpenCV, the solution must also use OpenCV for any computer vision and neural network capabilities. The solution must also be modular, so that it may be worked on by future students and easily integrated into the other subsystems.

3.2 Design Evaluation

To ensure a certain level of accuracy, this paper will use a "confusion matrix" to compare predicted classifications to the true classifications, a technique commonly used when comparing learning classifiers such as neural networks (Cireşan et al. 2012, Karpathy et al. 2014). This has the benefit of not only allowing the accuracy to be calculated, but also the sensitivity and specificity of the solution. The sensitivity of the solution is its ability to avoid overlooking positive results, whereas specificity is the ability of the solution to classify true negatives correctly. Both statistical measures are useful when measuring the capabilities of an object detector for autonomous driving purposes – low sensitivity values may prevent the object detection system from identifying a pedestrian or vehicle in time to avoid an accident. Therefore, the solution must have a significantly high sensitivity to ensure a level of practical reliability. Additionally, the solution must have a significant specificity so that it does not misclassify objects from the background. This is especially true in the case of detecting signs, as detecting a non-existent traffic sign may have undesirable effects when processed by the navigation system (such as driving at very high speed if it incorrectly detects a speed sign).

In addition, the solution should be fast enough to be performed in real time. While the definition of “real time” differs considerably between proposed architectures (Ren et al. 2017, Redmon et al. 2018), the relatively slow movement of the EyeBot robots and their limited hardware capabilities allows the inference time to be higher. This paper proposes that the target inference time of the object detection subsystem should target 200 milliseconds, or 5 frames per second (fps), similar to that of the Faster R-CNN paper (Ren et al. 2017).

3.3 Design Process

The MobileNet-SSD architecture was chosen for further investigation based on their balance of accuracy, latency, and network size on embedded hardware, as determined by the literature review. While its original implementation uses Google’s TensorFlow library (Sandler et al. 2018), the version of OpenCV available on the EyeBots at the time (version 3.3.0) did not have native support for TensorFlow models. Due to previous experience in the deep learning framework Caffe (Caffe 2018) and the existence of a port of MobileNet-SSD in this framework (GitHub 2018a), initial testing of the MobileNet architecture was performed with Caffe instead. Detection using the available pretrained models demonstrated a high level of accuracy, however the inference time was considerably higher than specified in the original paper (Howard et al. 2017), and there was little documentation on retraining the network on other datasets. While other solutions were being explored, a simple test dataset were prepared using the EyeSim simulator – 100 images on three different objects were gathered, including a soccer ball, a soft drink can, and an EyeBot. Bounding boxes with ground truth categories were defined in XML for each image using the open-source tool *labelImg* (GitHub 2018c).

Later updates to the OpenCV library allowed TensorFlow models to be loaded directly within the framework (starting with version 3.4.1), and so a test network was created using TensorFlow based on MobileNet-SSD v1 to assess the viability of porting networks between the two frameworks. The test network was created by retraining an existing MobileNet-SSD network using the EyeSim test dataset in TensorFlow until the losses reached a value of 1.0 (or until the losses reached a consistent minimum value). After several hours of training on the multi-GPU lab machine, the network was able to detect objects within the EyeSim simulator with a very high degree of accuracy with an average inference time/latency of 30 milliseconds – over 33 frames per second – on the same machine. While the excellent performance results were discounted due to the superior hardware compared to the EyeBots, the accuracy results were significant, and demonstrated that the network could be optimised to reduce accuracy in favour of speed with minimal effect. The results are shown in Figure 3 below:

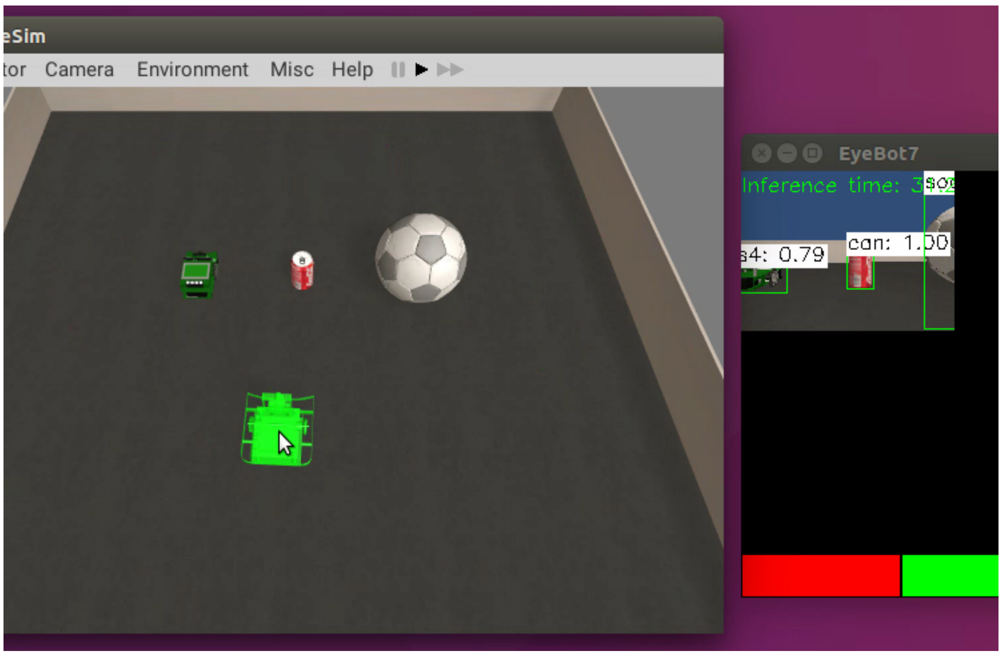


Figure 3: Demonstration of the accuracy of MobileNet-SSD in EyeSim

With the recent addition of MobileNet-SSDLite in TensorFlow (Sandler et al. 2018) and the high accuracy of the SSD-based MobileNet, there was the capacity to lose some accuracy and gain significant speed by using the new SSDLite architecture. A second network was trained using the same test EyeSim dataset on a MobileNet-SSDLite network previously trained on the COCO dataset (GitHub 2018b), resulting in a decrease in the inference time down to just 26 milliseconds (an increase to 38 frames per second). However, the overall accuracy of the network diminished significantly, with bounding boxes often not centred on objects or grouping multiple objects into a single box. For this reason, further experimentation focused on the MobileNet-SSD architecture.

The high accuracy of the MobileNet-SSD network within EyeSim was contributed in part to the relatively low number of classes – 3 compared to the 91 classes used in the COCO dataset that the MobileNet-SSD network was originally trained on (Lin et al. 2018), which yielded an accuracy of around 73% (Howard et al. 2017). Considering that a single object detection network would require knowledge of many different classes (vehicles, traffic signs, pedestrians, common obstacles, etc.) the accuracy seen in the EyeSim-based network would decrease significantly. This led to the concept of creating several networks designed for each category of classifier to retain this accuracy, while potentially allowing the networks to be optimised and reduced to improve inference times. This has the added benefit of allowing certain networks to be disabled by other components of the autonomous driving subsystem when they are not needed. For example, when the vehicle is waiting for pedestrians to cross a road, the navigation system can disable the traffic sign and vehicle detection networks until the system determines there are no more pedestrians, improving overall system performance and resource sharing. Additionally, this allows the different analytical properties (specificity, sensitivity and accuracy) to be compared and evaluated differently depending on the categories of objects that are being detected. For example, the behaviour of the navigation system will be highly dependent on the type of sign, such as a stop sign compared to a give way sign, and therefore need a high degree of sensitivity and accuracy to prevent the sign being misclassified as a different type of sign (or as the background). Vehicles are less susceptible to this problem, as the behaviour from the point of view of the navigation system will not likely change much between vehicle classes, so a lower level of accuracy is still acceptable.

Given the significant training time and vast number of images required to train a neural network from scratch, the networks would be retrained from the existing COCO-trained models available in TensorFlow (GitHub 2018b). Due to the limited real-world scenarios that would be reproducible on the track in the lab, there would be a very high chance that a network trained from scratch would exhibit overfitting to the presented scenarios. Additionally, there are several students working on postgraduate research projects that require use of the lab machine for training various learning algorithms, which means extended use of the lab machine for training is not possible. Extending a pretrained model has the advantage of reusing the models' learned features for detecting general objects, drastically reducing both the training time and the training dataset size, as the network must only learn the specifics of the new objects presented to it.

For the final object detection subsystem design, two networks were thought to give enough variety to test the navigation subsystem effectively without introducing too many variables and would demonstrate the key advantages of the proposed multi-network design. One network was designed to detect two very similar types of EyeBot (the original green Soccerbot, and an updated blue Soccerbot), while the other was designed to detect 6 different traffic signs (stop sign, give way sign, pedestrian crossing sign, 30 kilometre per hour zone sign, end of 30 kilometre per hour zone sign, and parking sign).
To create the real training dataset, each object was photographed using several EyeBots (to ensure the network did not overtrain to specific camera irregularities) from multiple angles and distances. This would ensure that the EyeBots could detect the objects with a high level of accuracy even from the other side of the track. Lighting, the relative position of the object in the frame, and the scale of the objects in the images were all varied to ensure that the network would learn the objects and not parameters about the environment or the camera.
Bounding boxes were manually recorded for each of the 476 images used to train the EyeBot network and the 378 images used to train the traffic sign network. In addition, a validation dataset was created with 100 images that would be used to test the performance of the networks in different situations. 50 of those images were similar to images used to train the network, while the other 50 differed from the training dataset (with partial occlusion, multiple similar nearby objects, etc.) to ensure that the network remained general enough to handle unfamiliar situations.

To extract the maximum performance from the networks while retaining acceptable accuracy levels, different values for the MobileNet width hyperparameter (α) were used on each of the two networks (1.0, 0.75, 0.5 and 0.25). This results in a total of eight MobileNet-SSD networks. The resolution parameter was not used to reduce network size for two reasons. Firstly, separate reductions in width and resolution result in an equal decrease in accuracy, however the reduced-width network has far fewer computations (and therefore faster inference time) as well as almost a third of the parameters of the reduced-resolution network (resulting in a smaller memory footprint). Secondly, the width hyperparameter is easier to tune in TensorFlow whereas changing the resolution hyperparameter requires readjustment of the SSD component of the network, making it less likely to negatively impact overall accuracy.

4. Results

Each of the eight networks were run on the validation dataset with a confidence threshold of 0.7 and non-maxima suppression of 0.2. The high confidence threshold ensures that the object detection algorithm does not output unlikely detections (incorrect detections could have major consequences when the navigation system uses the data for path planning) and the low non-maxima suppression value allows the network to detect objects separated by relatively small distances. A copy of the processed images complete with labelled bounding boxes is saved for processing later, and the inference times for each image in the validation set is also recorded. The following section details how that data was processed to extract relevant statistical parameters for each network, and presents the findings on network accuracy, sensitivity, specificity and latency.

4.1 Confusion Matrices

As mentioned in section 3.2, several statistical parameters can be determined by creating a confusion matrix from the test results. By analysing the results of the object detection networks on each individual image in the validation dataset, a confusion matrix can be constructed, as shown in Table 1 below. Note the bold font for the true positive predictions:

Table 1: Confusion matrix for the EyeBot network, with a width hyperparameter of 1.0

EyeBot-1.0		Actual Class		
		Green EyeBot	Blue EyeBot	None
Predicted Class	Green EyeBot	**29**	3	2
	Blue EyeBot	1	**15**	3
	None	11	4	**51**

A full set of confusion matrices for all eight networks can be found in Appendix A.
From here, a table of confusion can be created for each class, reducing the data down to a simple binary table. Table2 shows a table of confusion for the Green EyeBot class from the data in Table 1:

Table 2: Table of confusion for the "Green EyeBot" class in the EyeBot network, with a width hyperparameter of 1.0

EyeBot-1.0	Green EyeBot	Non-Green EyeBot
Green EyeBot	29 True Positives	5 False Positives
Non-Green EyeBot	12 False Negatives	73 True Negatives

This table of confusion can be used to calculate the sensitivity, specificity and accuracy of the network on these parameters, using the following formulas from Fawcett (2006):

$$Sensitivity = \frac{TP}{P} = \frac{TP}{TP + FN}$$

$$Specificity = \frac{TN}{N} = \frac{TN}{TN + FP}$$

$$Accuracy = \frac{TP + TF}{P + N} = \frac{TP + TF}{TP + TN + FP + FN}$$

From these formulas, the sensitivity for the Green EyeBot class in the table above is calculated to be 0.7073, the specificity is 0.9359, and the accuracy is 0.8571.
These calculations have been performed for all classes in each of the eight networks, with the results attached in Appendix B.

4.2 Accuracy

Averaging the accuracy of all categories in the networks allows the overall network accuracy to be calculated. The results are shown below in Figure 4, showing the effects of the MobileNet width parameter on network accuracy for both networks:

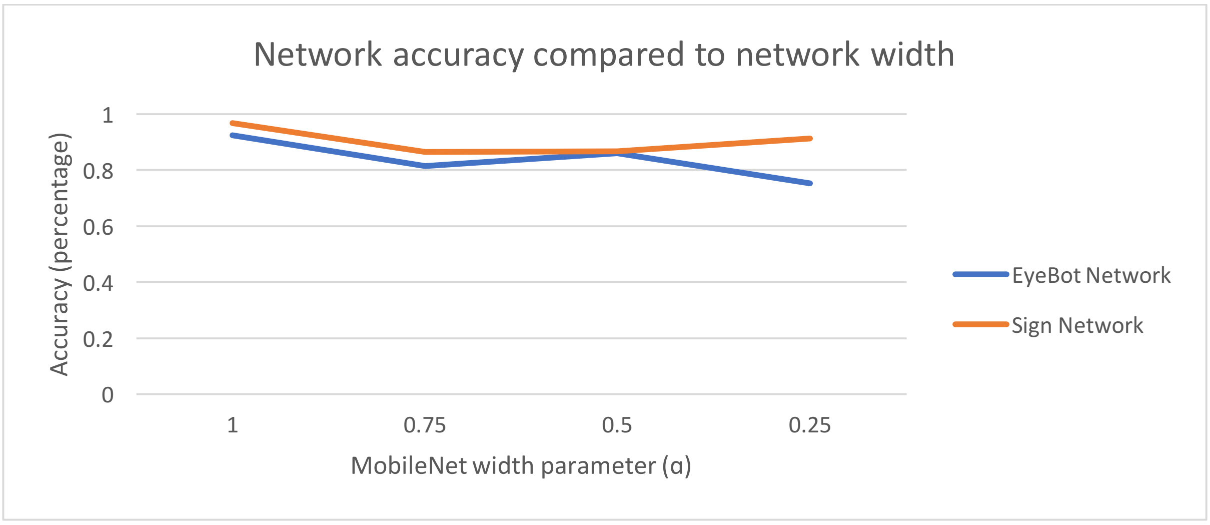


Figure 4: Graph of network accuracy compared to network width.

Note that the accuracy remains relatively high despite significant decreases in the width of the network.

4.3 Sensitivity

Averaging the sensitivity of all categories in the networks allows the overall network sensitivity to be calculated. The results are shown below in Figure 5, showing the effects of the MobileNet width parameter on network sensitivity for both networks:

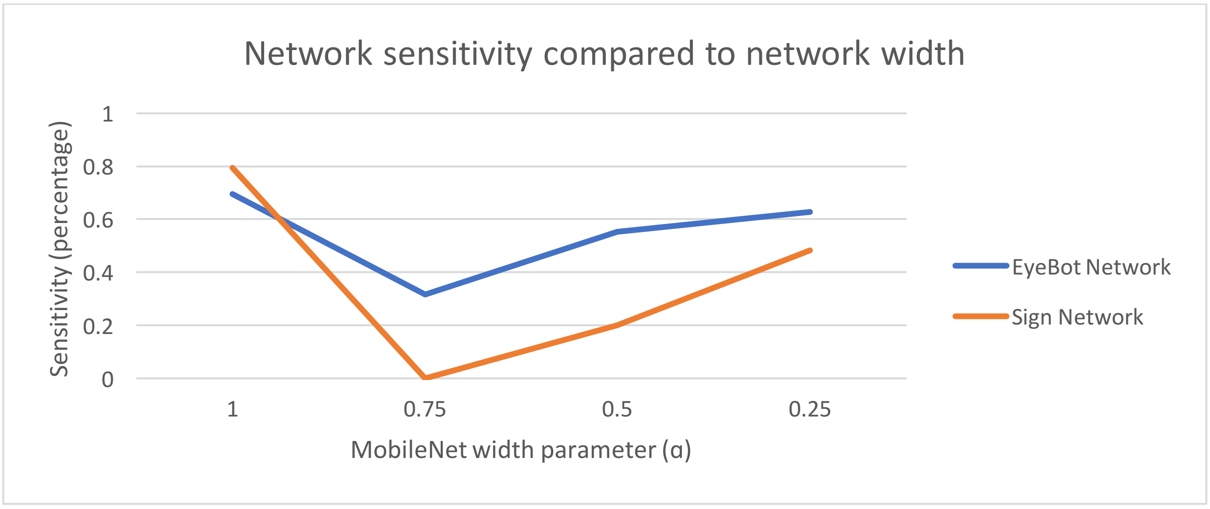


Figure 5: Graph of network sensitivity compared to network width

Note the significant lack of sensitivity of both the EyeBot and Sign networks when the width parameter is 0.75 – this will be discussed further in Section 5. Both networks achieve their highest sensitivity at the full network width, as expected from the review of the literature.

4.4 Specificity

Averaging the specificity of all categories in the networks allows the overall network specificity to be calculated. The results are shown below in Figure 6, showing the effects of the MobileNet width parameter on network specificity for both networks:

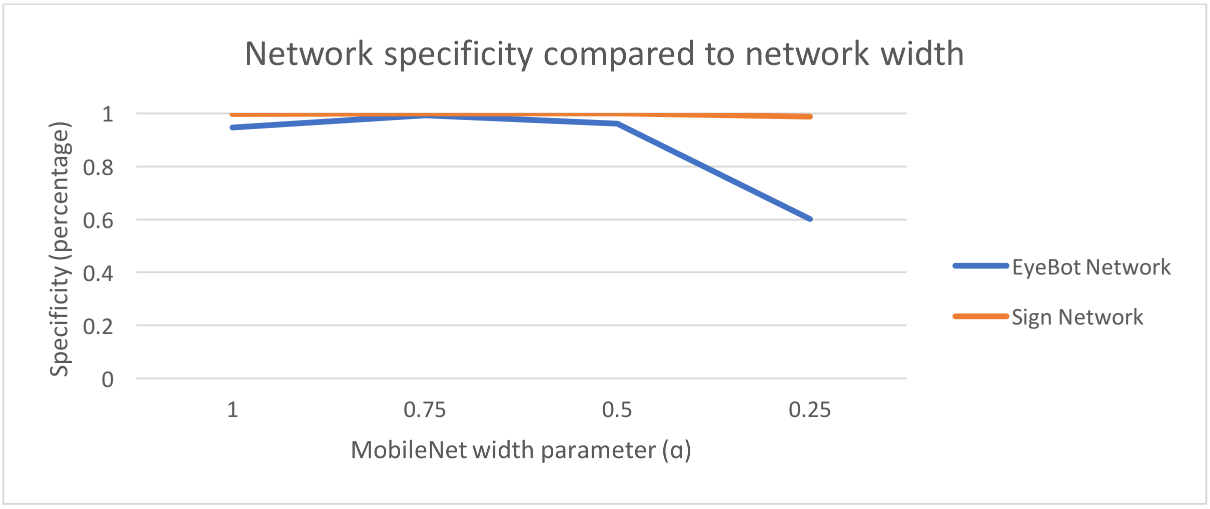


Figure 6: Network specificity compared to network width

Note that the specificity stays near a value of 1 for all width values, except for the EyeBot network at a width of 0.25. Further discussion on how these parameters relate back to the sensitivity values will be explored further in Section 5.

4.5 Latency

Network latency times were recorded for each image in the validation dataset using the EyeBot hardware, and the average, minimum and maximum times were calculated for each of the eight networks. The results were run five times to ensure that the results were consistent, and the latency results are displayed below in Figure 7:

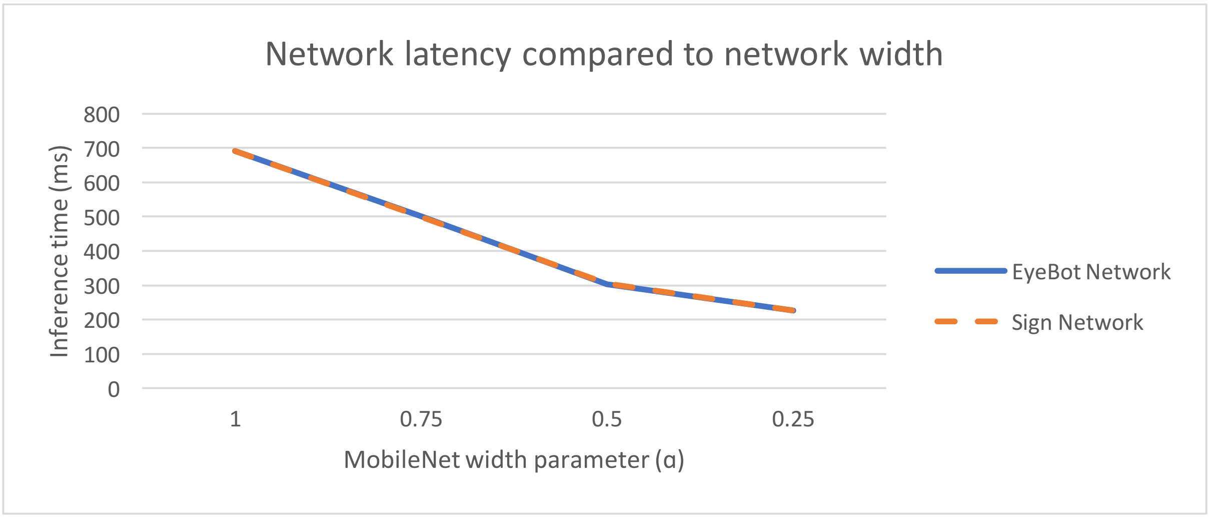


Figure 7: Network latency compared to network width

Note the near linear relationship between network latency and the MobileNet width hyperparameter, only approaching an asymptote at the smallest width value. Interestingly, both networks achieved almost identical inference times with the same width hyperparameter, which will be discussed further in the next section.

5. Discussion

As seen in Figure 4 above, the accuracies of the networks are impressively high. The full-sized networks (with a width hyperparameter of 1.0) achieve an accuracy of 92.5% and

96.8% on the EyeBot and traffic sign networks, respectively. Even as the worst-performing network in terms of accuracy, the EyeBot network with a width of 0.25 achieved an accuracy of 75.3%. However, this does not indicate that all the networks were successful – for example, the traffic sign network trained with a width of 0.75 had an accuracy of 86.4%, however in practice the network did not successfully identify any traffic sign. This discrepancy is due to the accuracy being calculated as the percentage of identified true results (true positive and true negative) out of all results, and why it is important to view the sensitivity and specificity values of the networks as well. The sensitivities indicate the capability of the network to detect objects when they are truly there, the accuracy indicates how often the detected objects are correctly classified, and the specificity indicates how resilient the network is against falsely detecting objects in an image. Given that half of the images in the validation dataset differ from the images in the training dataset, sensitivity values greater than 0.5 indicate that the network has been able to identify objects within the lesser-known part of the validation dataset. Therefore, it can be inferred that the network has avoided overtraining and has successfully learned to detect objects in a variety of conditions. From Figure 5 above, four networks fit that requirement – all EyeBot networks aside from the one with the width hyperparameter of 0.75, and the traffic sign network with a width hyperparameter value of 1.0. The traffic sign network with a width of 0.25 gets close with a value of 0.483.

The poor performance of the 0.75 networks is likely due to some object classification information being lost because of the smaller width value, resulting in more training steps being required when compared to the full-width network. This would be offset in smaller network architectures, as there are far fewer parameters to tune, requiring fewer total steps to converge. This would explain why the sensitivity of both networks increase at a width of 0.25 to its second highest values (the highest being when the width value is 1.0) but are at their lowest at a width value of 0.75. With enough training time this would not be an issue, however, as mentioned in Section 3.3, the limited access to the lab machines for network training along with the large number of networks to be trained resulted in most networks being stopped after approximately 10,000 steps. The networks had relatively low losses (between 1.0 and 2.0) but it is likely they were halted before they may have had a chance to relearn some of the deeper object extraction features lost in the width reduction.

Looking at the three EyeBot networks with sensitivities above 0.5, there is a clear trade-off occurring at the lowest width of 0.25 between sensitivity and specificity. As shown in Figure 4, the network can achieve a sensitivity of 0.628 at this width, close to its maximum recorded sensitivity of 0.695 on the full-sized network. This is offset by the significant decrease in the specificity from approximately 0.95 down to just 0.628 as shown in Figure 6, as well as the overall decrease in accuracy of the network to 75.3%, the lowest of all networks, as shown in Figure 4. Given that the network with a width of 0.5 only just exceeds the threshold of sensitivity (with a value of 0.553) the best overall option given the data is the full-sized network (network with a width hyperparameter of 1.0). This results in a network that can detect EyeBots with 92.46% accuracy, has a sensitivity (detection accuracy) of 0.6946, and a specificity of 0.9473. Additionally, as shown in Figure 7, this network can achieve those results with an average latency of 690.68 milliseconds, equating to 1.448 frames per second. While this frame rate falls short of the 5 frame per second target mentioned in Section 3.2, there are still optimisations that can be made to help improve speed – optimised OpenCV libraries for the Raspberry Pi were not used but may help increase performance by 30-50% (Rosebrock 2017). Additionally, further training of the network would help improve the network sensitivity at all widths, allowing the smallest and fastest networks to become viable options.

When looking at the two traffic sign networks (with width hyperparameters of 1.0 and 0.25) that have sensitivities at or near 0.5, there is a clear decrease in sensitivity at the lower width network. While the full-sized network can achieve a sensitivity of 0.793 (the highest of all networks), the next most sensitive network (with a width of 0.25) only achieves a sensitivity of 0.483. However, the specificity and accuracy of the network is retained at this low level, as shown in Figure 6 and Figure 4, respectively. Given that the smaller of the two networks can achieve around 60% of the sensitivity with only about 30% of the latency time, the higher number of frames will allow the smaller network to compensate for the lower sensitivity. While the network may miss detections more frequently on the smaller network, there is a greater chance of detecting the object overall considering there are more frames to analyse, a result of the much faster inference time. This inference time (averaging

226.725 milliseconds) equates to 4.411 frames per second, very close to the 5 frame per second target outlined in Section 3.2.

6. Conclusions

This paper demonstrates the viability of the MobileNet-SSD neural network architecture for object detection on embedded hardware. After exploring the existing literature regarding object detection algorithms, several prototype MobileNet-SSD networks were created, demonstrating the benefits of this architecture in terms of speed and accuracy. A multi-network design was proposed to ensure maximum efficiency on the hardware, and statistical analysis methods were used to measure the effectiveness of this design in practice. The analysis demonstrated the effectiveness of the proposed solution and resulted in networks that are able to detect objects at up to 4.4 frames per second while maintaining a high level of accuracy and sensitivity.

As a next step towards improving the results that were obtained here, further training of the networks with the given datasets should help alleviate some of the sensitivity issues identified in Section 5. Furthermore, an expanded training dataset including more varied lighting conditions, more cases of partial occlusion, motion blur, and general noise should increase the sensitivity of the networks, allowing smaller and faster networks to be used. Additionally, an expanded dataset such as this may allow the use of the MobileNet-SSDLite network architecture, which would further decrease detection times. Additional improvements include the use of optimised OpenCV libraries for the Raspberry Pi compute boards used by the EyeBots, and closer integrations of the API into the other autonomous driving subsystems, which should provide further speed increases.

7. References

AADC 2018 Rulebook. (2018). [ebook] Ingolstadt: Audi, pp.17-23. Available at: https://www.audi-autonomous-driving-cup.com/wp-content/uploads/2018/04/2018-04-16_en_Rulebook_2018_V1.0_EU.pdf [Accessed 21 Apr. 2018].

ADVI. (2018). *What is a driverless car? | ADVI | Australian Driverless Vehicle Initiative*. [online] Available at: http://advi.org.au/driverless-technology/ [Accessed 18 Apr. 2018].

Agarwal, A., Golash, H., Qian, Y., Zhang, M. and Zhang, Z. (2017). *Perception System for Autonomous Driving*. [online] Pittsburg, p.8. Available at: http://mrsdprojects.ri.cmu.edu/2016teama/wp-content/uploads/sites/12/2016/09/TeamA-FinalReport.pdf [Accessed 18 Apr. 2018].

ArsTechnica (2017). *Google's Waymo invests in LIDAR technology, cuts costs by 90 percent*. [online] Available at: https://arstechnica.com/cars/2017/01/googles-waymo-invests-in-lidar-technology-cuts-costs-by-90-percent/ [Accessed 18 Apr. 2018].

Audi Autonomous Driving Cup. (2018a). *Computers*. [online] Available at: https://www.audi-autonomous-driving-cup.com/car/computers/ [Accessed 4 Mar. 2018].

Audi Autonomous Driving Cup. (2018b). *Home*. [online] Available at: https://www.audi-autonomous-driving-cup.com/ [Accessed 4 Mar. 2018].

Audi Autonomous Driving Cup. (2018c). *Sensors*. [online] Available at: https://www.audi-autonomous-driving-cup.com/car/sensors/ [Accessed 4 Mar. 2018].

Babenko, B., Yang, M. and Belongie, S. (2009). Visual Tracking with Online Multiple Instance Learning. [online] Available at: https://ieeexplore.ieee.org/document/5206737/ [Accessed 21 Apr. 2018].

Bay, H., Ess, A., Tuytelaars, T. and Van Gool, L. (2006). Speeded Up Robust Features (SURF). *Computer Vision and Image Understanding*, 110(3), pp.346-359.

Behere, S. and Törngren, M. (2016). A functional reference architecture for autonomous driving. *Information and Software Technology*, [online] 73, pp.136-150. Available at: https://sagar.se/files/wasa2015.pdf [Accessed 18 Apr. 2018].

Bräunl, T., Keat, R. and Pham, M. (2018). RoBIOS - Mobile Robot Library. [online] RoBIOS-7 Library. Available at: http://robotics.ee.uwa.edu.au/eyebot7/Robios7.html [Accessed 21 Oct. 2018].

Caffe. (2018). Caffe | Deep Learning Framework. [online] Available at: http://caffe.berkeleyvision.org/ [Accessed 21 Oct. 2018].

Cireşan, D., Meier, U., Masci, J. and Schmidhuber, J. (2012). Multi-column deep neural network for traffic sign classification. Neural Networks, [online] 32, pp.333-338. Available at: https://arxiv.org/pdf/1202.2745.pdf [Accessed 21 Oct. 2018].

Dalal, N. and Triggs, B. (2005). Histograms of Oriented Gradients for Human Detection. [online] Available at: https://ieeexplore.ieee.org/document/1467360/ [Accessed 16 Apr. 2018].

Fawcett, T. (2006). An introduction to ROC analysis. Pattern Recognition Letters, 27(8), pp.861-874.

Friedman, J., Tibshirani, R., Hastie, T. (2000). Additive Logistic Regression: A Statistical View of Boosting (With Discussion and a Rejoinder by the Authors). Annals of Statistics - ANN STATIST. 28. 337-407. 10.1214/aos/1016120463.

Girshick, R., Donahue, J., Darrell, T. and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. [online] Available at: https://arxiv.org/pdf/1311.2524.pdf [Accessed 17 Apr. 2018].

Girshick, R. (2015). Fast R-CNN. [online] Available at: https://arxiv.org/pdf/1504.08083.pdf [Accessed 17 Apr. 2018].

GitHub. (2018a). *MobileNet-SSD*. [online] Available at: https://github.com/chuanqi305/MobileNet-SSD [Accessed 21 Oct. 2018].

GitHub. (2018b). *TensorFlow Models*. [online] Available at: https://github.com/tensorflow/models [Accessed 21 Oct. 2018].

GitHub. (2018c). *tzutalin/labelImg*. [online] Available at: https://github.com/tzutalin/labelImg [Accessed 21 Oct. 2018].

Hebert, M. (1988). *Computer Vision For Autonomous Navigation*. [ebook] Carnegie-Mellon University, pp.7-8. Available at: https://www.ri.cmu.edu/pub_files/pub3/hebert_martial_1988_3/hebert_martial_1988_3.p df [Accessed 7 Apr. 2018].

Howard, A., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M. and Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv. [online] Available at: https://arxiv.org/pdf/1704.04861.pdf [Accessed 21 Oct. 2018].

Intel Developer Zone. (2018). Intel® Movidius™ Neural Compute Stick. [online] Available at: https://software.intel.com/en-us/neural-compute-stick [Accessed 21 Oct. 2018].

Janku, P., Koplik, K., Dulik, T. and Szabo, I. (2016). Comparison of tracking algorithms implemented in OpenCV. *MATEC Web of Conferences*, [online] 76, p.04031. Available at: https://www.matec-conferences.org/articles/matecconf/pdf/2016/39/matecconf_cscc2016_04031.pdf [Accessed 7 Apr. 2018].

Kalal, Z., Mikolajczyk, K. and Matas, J. (2010). Forward-Backward Error: Automatic Detection of Tracking Failures. *2010 20th International Conference on Pattern Recognition*. [online] Available at: https://ieeexplore.ieee.org/document/5596017/ [Accessed 21 Apr. 2018].

Kalal, Z., Mikolajczyk, K. and Matas, J. (2012). Tracking-Learning-Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, [online] 34(7), pp.1409-1422. Available at: http://epubs.surrey.ac.uk/709857/1/Kalal-PAMI-2011%281%29.pdf [Accessed 21 Apr. 2018].

Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R. and Fei-Fei, L. (2014). Large-Scale Video Classification with Convolutional Neural Networks. 2014 IEEE Conference on Computer Vision and Pattern Recognition. [online] Available at: https://www.cv-foundation.org/openaccess/content_cvpr_2014/papers/Karpathy_Large-scale_Video_Classification_2014_CVPR_paper.pdf [Accessed 21 Oct. 2018].

Lin, T., Patterson, G., Ronchi, M., Cui, Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ravaman, D., Zitnick, L. and Dollar, P. (2018). COCO - Common Objects in Context. [online] Cocodataset.org. Available at: http://cocodataset.org [Accessed 21 Oct. 2018].

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. and Berg, A. (2016). SSD: Single Shot MultiBox Detector. [online] Available at: https://arxiv.org/pdf/1512.02325.pdf [Accessed 20 Apr. 2018].

Lowe, D. (2004). Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2), pp.91-110.

National Transport Safety Board (2018). *NTSB UPDATE: Uber Crash Investigation*. [online] Available at: https://www.ntsb.gov/news/press-releases/Pages/NR20180320.aspx [Accessed 18 Apr. 2018].

nuTonomy. (2017). *nuTonomy - nuTonomy and Lyft Launch Boston Self-Driving Pilot*. [online] Available at: https://www.nutonomy.com/company-news/nutonomy-and-lyft-launch-boston-self-driving-pilot/ [Accessed 18 Apr. 2018].

Redmon, J., Divvala, S., Girshick, R. and Farhadi, A. (2018). You Only Look Once: Unified, Real-Time Object Detection. [online] Available at: https://arxiv.org/pdf/1506.02640.pdf [Accessed 21 Oct. 2018].

Ren, S., He, K., Girshick, R. and Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, [online] 39(6), pp.1137-1149. Available at: https://ieeexplore.ieee.org/document/7485869/ [Accessed 18 Apr. 2018].

Rio Tinto. (2017). *Rio Tinto to expand autonomous fleet as part of $5 billion productivity drive*. [online] Available at: http://www.riotinto.com/media/media-releases-237_23802.aspx [Accessed 18 Apr. 2018].

Robotics UWA. (2008). *EyeBot - Online Documentation*. [online] Available at: http://robotics.ee.uwa.edu.au/eyebot/ [Accessed 20 Mar. 2018].

Robotics UWA. (2017). *EyeSim VR*. [online] Available at: http://robotics.ee.uwa.edu.au/eyesim/ [Accessed 13 Mar. 2018].

Rosebrock, A. (2017). Optimizing OpenCV on the Raspberry Pi. [online] PyImageSearch. Available at: https://www.pyimagesearch.com/2017/10/09/optimizing-opencv-on-the-raspberry-pi/ [Accessed 21 Oct. 2018].

Rublee, E., Rabaud, V., Konolige, K. and Bradski, G. (2011). ORB: An efficient alternative to SIFT or SURF. [online] Available at: https://ieeexplore.ieee.org/abstract/document/6126544/ [Accessed 17 Apr. 2018].

Sandler, M. & Howard, A. (2018). MobileNetV2: The Next Generation of On-Device Computer Vision Networks. [Blog] Google AI Blog. Available at: https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html [Accessed 21 Oct. 2018].

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A. and Chen, L. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. arXiv. [online] Available at: https://arxiv.org/pdf/1801.04381.pdf [Accessed 19 Oct. 2018].

Tesla (2018). *A Tragic Loss*. [online] Available at: https://www.tesla.com/en_AU/blog/tragic-loss [Accessed 18 Apr. 2018].

Waymo. (2017). *Technology – Waymo*. [online] Available at: https://waymo.com/tech/ [Accessed 18 Apr. 2018].

8. Appendices

Appendix A: Confusion Matrices

Table 3: Confusion matrix for the EyeBot network, with a width hyperparameter of 1.0

EyeBot-1.0		Actual Class		
		Green EyeBot	Blue EyeBot	None
Predicted Class	Green EyeBot	**29**	3	2
	Blue EyeBot	1	**15**	3
	None	11	4	**51**

Table 2: Confusion matrix for the EyeBot network, with a width hyperparameter of 0.75

EyeBot-0.75		Actual Class		
		Green EyeBot	Blue EyeBot	None
Predicted Class	Green EyeBot	**10**	0	1
	Blue EyeBot	0	**8**	0
	None	30	13	**53**

Table 3: Confusion matrix for the EyeBot network, with a width hyperparameter of 0.5

EyeBot-0.50		Actual Class		
		Green EyeBot	Blue EyeBot	None
Predicted Class	Green EyeBot	**17**	1	3
	Blue EyeBot	0	**13**	2
	None	16	8	**49**

Table 4: Confusion matrix for the EyeBot network, with a width hyperparameter of 0.25

EyeBot-0.25		Actual Class		
		Green EyeBot	Blue EyeBot	None
Predicted Class	Green EyeBot	**17**	0	37
	Blue EyeBot	9	**17**	42
	None	12	4	**1**

Table 5: Confusion matrix for the traffic signs network, with a width hyperparameter of 1.0

Signs-1.0		Actual Class						
		Pedestrian Sign	Give Way Sign	Parking Sign	30 Zone Sign	30 End Sign	Stop Sign	None
Predicted Class	Pedestrian Sign	**12**	0	0	0	0	0	0
	Give Way Sign	0	**20**	0	0	0	0	0
	Parking Sign	0	0	**10**	0	0	0	0
	30 Zone Sign	0	0	0	**12**	0	0	1
	30 End Sign	0	0	0	0	**14**	0	1
	Stop Sign	0	0	0	0	0	**12**	0
	None	1	3	8	4	5	1	**22**

Table 6: Confusion matrix for the traffic signs network, with a width hyperparameter of 0.75

Signs-0.75		Actual Class						
		Pedestrian Sign	Give Way Sign	Parking Sign	30 Zone Sign	30 End Sign	Stop Sign	None
Predicted Class	Pedestrian Sign	**0**	0	0	0	0	0	0
	Give Way Sign	0	**0**	0	0	0	0	0
	Parking Sign	0	0	**0**	0	0	0	0
	30 Zone Sign	0	0	0	**0**	0	0	0
	30 End Sign	0	0	0	0	**0**	0	0
	Stop Sign	0	0	0	0	0	**0**	0
	None	13	22	20	16	19	13	**24**

Table 7: Confusion matrix for the traffic signs network, with a width hyperparameter of 0.5

Signs-0.50		Actual Class						
		Pedestrian Sign	Give Way Sign	Parking Sign	30 Zone Sign	30 End Sign	Stop Sign	None
Predicted Class	Pedestrian Sign	**2**	0	0	0	0	0	0
	Give Way Sign	0	**3**	0	0	0	0	0
	Parking Sign	0	0	**3**	0	0	0	0
	30 Zone Sign	0	0	0	**3**	0	0	0
	30 End Sign	0	0	0	0	**2**	0	0
	Stop Sign	0	0	0	0	0	**6**	0
	None	10	19	17	13	17	7	**2**

Table 8: Confusion matrix for the traffic signs network, with a width hyperparameter of 0.25

Signs-0.25		Actual Class						
		Pedestrian Sign	Give Way Sign	Parking Sign	30 Zone Sign	30 End Sign	Stop Sign	None
Predicted Class	Pedestrian Sign	**3**	0	0	0	1	0	1
	Give Way Sign	0	**7**	0	0	0	0	0
	Parking Sign	0	0	**7**	0	0	0	0
	30 Zone Sign	0	0	0	**3**	0	1	0
	30 End Sign	0	0	0	0	**11**	0	2
	Stop Sign	0	0	0	0	0	**4**	0
	None	4	10	7	8	2	3	**4**

Appendix B: Model Parameters for each Class

Table 1: Model parameters for the "Blue EyeBot" class

Width Parameter	Sensitivity	Specificity	Accuracy
1	0.68181818	0.95876289	0.90756303
0.75	0.38095238	1	0.88695652
0.5	0.59090909	0.97701149	0.89908257
0.25	0.80952381	0.56779661	0.60431655

Table 2: Model parameters for the "Green EyeBot" class

Width Parameter	Sensitivity	Specificity	Accuracy
1	0.70731707	0.93589744	0.85714286
0.75	0.25	0.98666667	0.73043478
0.5	0.51515152	0.94736842	0.81651376
0.25	0.44736842	0.63366337	0.58273381

Table 3: Model parameters for the "Pedestrian Sign" class

Width Parameter	Sensitivity	Specificity	Accuracy
1	0.92307692	1	0.99206349
0.75	0	1	0.8976378
0.5	0.16666667	1	0.90384615
0.25	0.42857143	0.97183099	0.92307692

Table 4: Model parameters for the "Give Way Sign" class

Width Parameter	Sensitivity	Specificity	Accuracy
1	0.86956522	1	0.97619048
0.75	0	1	0.82677165
0.5	0.13636364	1	0.81730769
0.25	0.41176471	1	0.87179487

Table 5: Model parameters for the "Parking Sign" class

Width Parameter	Sensitivity	Specificity	Accuracy
1	0.55555556	1	0.93650794
0.75	0	1	0.84251969
0.5	0.15	1	0.83653846
0.25	0.5	1	0.91025641

Table 6: Model parameters for the "30 Zone Sign" class

Width Parameter	Sensitivity	Specificity	Accuracy
1	0.75	0.99065421	0.96031746
0.75	0	1	0.87401575
0.5	0.1875	1	0.875
0.25	0.27272727	0.98507463	0.88461538

Table 7: Model parameters for the "End 30 Zone Sign" class

Width Parameter	Sensitivity	Specificity	Accuracy
1	0.73684211	0.99065421	0.95238095
0.75	0	1	0.8503937
0.5	0.10526316	1	0.83653846
0.25	0.78571429	0.96875	0.93589744

Table 84: Model parameters for the "Stop Sign" class

Width Parameter	Sensitivity	Specificity	Accuracy
1	0.92307692	1	0.99206349
0.75	0	1	0.8976378
0.5	0.46153846	1	0.93269231
0.25	0.5	1	0.94871795