University College Dublin

BE Electronic Engineering Project Final Report

Cyclist and Pedestrian Detection Based on Computer Vision and Artificial Intelligence Algorithms

Author
Shane Mullins

Supervisors
Prof. Anding Zhu & Dr. Muhammad Usman

April 24, 2021

Abstract

Artificial intelligence (AI) has progressed tremendously over the past two decades. Increased computational capabilities and the advancement of efficient and elaborate algorithms has brought about a resurgence in popularity in related fields. From recommender systems to anomaly detection, AI plays a role in many widespread technological tools used throughout the world today. One such field, boasting huge current and future potential, is computer vision and, more specifically, object recognition. Developing machines that can visually perceive and understand their environments has progressed beyond science-fiction and currently sees use in systems such as autonomous vehicles and facial-recognition devices. The ultimate goal of this project is to manipulate state-of-the-art object recognition technology to engender a positive impact with regards to the safety of vulnerable road users (VRUs).

Contents

1 Introduction

1.1 Problem Statement

It is important to be clear on the exact problem at hand, the societal demand for a solution and the specifics of the proposed methodology to achieve such a solution. This clarity helps to avoid ambiguity in progression and paves a straightforward path to achieve the goal. Given the breadth of the topic of object recognition and the extensive literature available, this idea is of particular importance. First, a concise problem should be defined. A discussion and exploration of the nature of the problem should then follow, leading to an appropriate possible solution. Through investigation and experience, this solution is realised within the constraints of feasibility and available resources. The following paragraphs use this general structure in the formulation of a pressing, real-world problem and a suitable solution.

"Driver failure to observe was the most cited driver action preceding a cyclist injury in a collision with a car or a goods vehicle" as per cyclist injury data from 2006-2018 in Ireland (RSA, 2018). According to the Road Safety Authority (RSA), VRU fatalities represented over one third of all fatalities for the year of 2019 (RSA, 2020). However, due to recent improvements in vehicle safety equipment and road-safety regulations, overall road accidents are decreasing rapidly as seen in Figure 1.

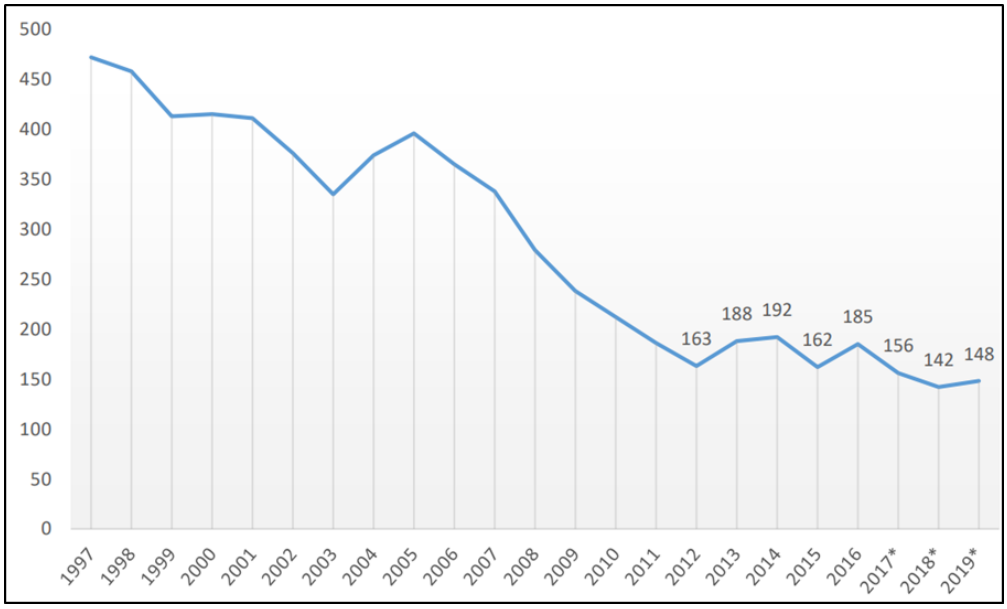


Figure 1: Road Fatalities, 1997-2019

This is true even in spite of an overall increase in cyclists and drivers on the road as per the Central Statistics Office (CSO) reports (CSO, 2018). It is surprising that despite the decrease in overall road accidents, **accidents relating to pedestrians and cyclists continue to rise** as per 2020 collision data (RSA, 2021). Figure 2 reflects this by showing the proportion of accidents per road user type. Despite the existence of far more vehicles on Irish roads than pedestrians and cyclists, a disproportionately large amount of deaths occur

within the category of the latter two. Two important conclusions can be drawn from this. Technological advancements and tailored, preventative policies can have a profound effect on the achievement of safety within a particular area. The improvement of Advanced Driver Assistance Systems (ADAS) have a real impact on reducing road accidents and saving lives (Geronimo, Lopez, Sappa, & Graf, 2009). It is a source of comfort that there exists such a strong relationship between implemented technology and road safety. On the other hand, it is evident that the safety of VRUs in particular is being somewhat neglected.

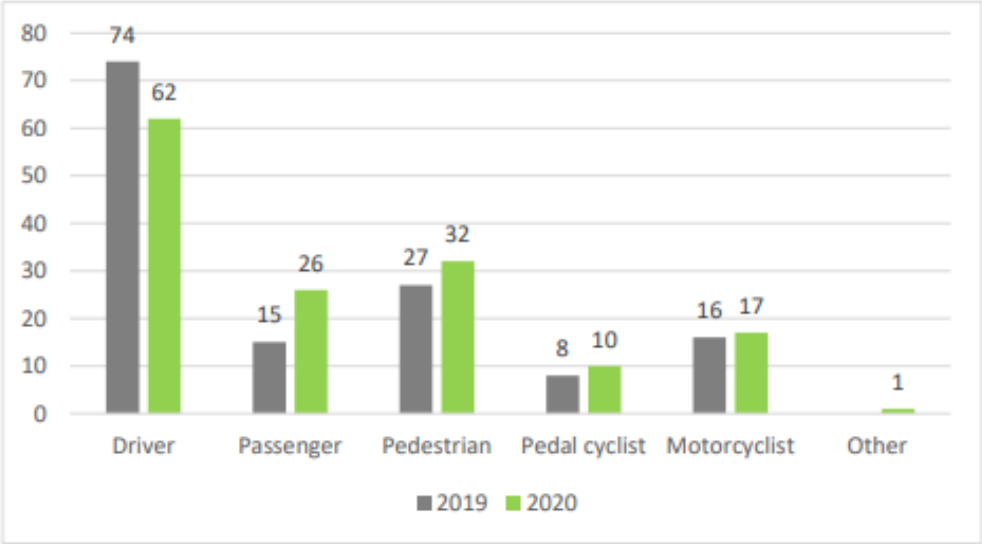


Figure 2: Fatalities per Road User Type

The problem of road safety is mirrored all across Europe with over 1.23 million road injuries occurring in the year 2018 in Europe (Eurostat, 2018). In over two out of five cyclist injuries in collisions with cars and goods vehicles, it was reported that the vehicle driver failed to observe prior to the collision (RSA, 2018). This statement, along with the opening line of the introduction, helps to pinpoint the cause of road-related fatalities and injuries: **inadequate observation.** It is clear that drivers need assistance when it comes to the safety of VRUs around them. In 2016, over half of all cyclist collisions occurred at a junction where there typically exists various blind spots (RSA, 2018). It is also interesting to note that the majority of pedestrian deaths occur on roads with a speed limit of 50km/h or less (RSA, 2020). The nature of such incidents, often occurring at lower speeds and at junctions, **lends itself to the use of supplementary technology.** Given that less than 8% of cyclist injuries occurred without the involvement of another vehicle or person (RSA, 2018), suggests that the responsibility lies in the hands of vehicle users. With this in mind, **the goal of this project revolves around the construction of a compact camera module that uses real-time images to detect nearby VRUs in a moving vehicle.** It is hoped that this information can then be conveyed, in a useable format, to a system (person, autonomous car...) that can perform preventative measures. If done successfully, the aforementioned shortcomings of human observation will be overcome with the forecasted **result of fewer VRU fatalities and injuries.** It is important to realise that the aim is

not to create a state-of-the-art object detection device that can compete with existing commercial products. Due to the nature of the project and the resources/expertise available to large multinational companies, this would not be feasible. Instead, the hope is to create a device with useful and versatile functionality that tests the limits of object detection on computationally-inexpensive systems and has the potential to be built upon for future real-life application. The originality of the project stems from working within the constraints of small, inexpensive hardware and from the unique combination of the resulting device's functionality. The following sections deal with three pressing questions. What pedestrian/cyclist detection algorithm should be implemented on the chosen hardware? What hardware should make up the compact module to produce real-time images of the road? And how will the software and hardware be integrated?

2 Literature Review

There exists extensive literature on object recognition and, as such, only noteworthy papers relevant to this particular project are discussed. This review discusses many popular object recognition algorithms, leading to the final choice of Single-Shot Detector (SSD) which formed the basis of the project work over the past months. It is important to note that many variations of each algorithm exist, often with overlap between them. Figure 3 reflects popularity and importance (particularly in recent years) of the field of object detection amongst the academic community (Zou, Shi, Guo, & Ye, 2019).

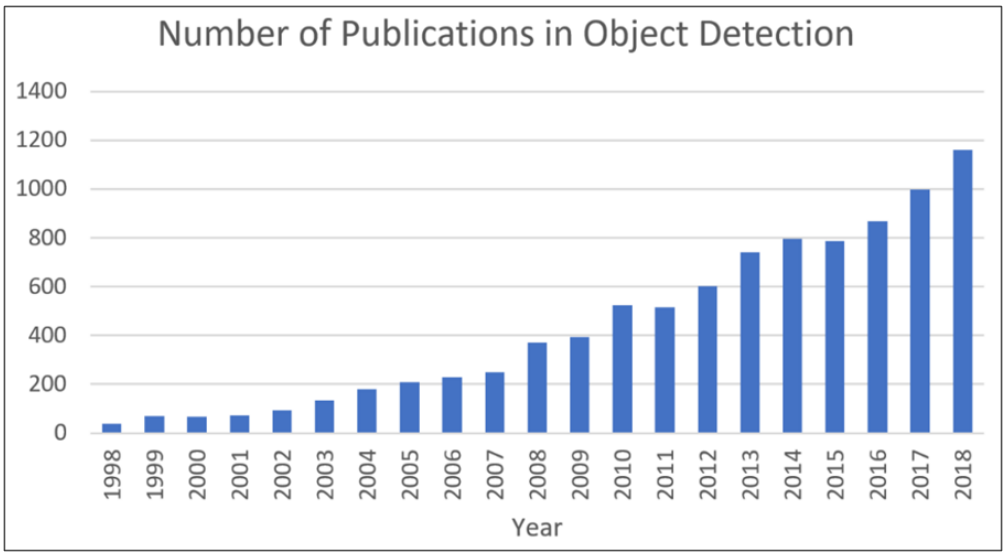


Figure 3: Object Detection Publications, 1998-2018

The algorithms discussed below provide foundational frameworks to be modified and suited for a particular purpose. In that sense, an in-depth analysis of each algorithm is deemed extraneous to the task at hand. Rather, a structured overview of the state-of-the-art and key time-lined events are presented. With this, comes a broad understanding of the capabilities and limitations of object recognition, on top of which the specific procedure of this project can be built.

Speed and accuracy are the two pillars of object recognition algorithms. In 2016, a Microsoft research team (He, Zhang, Ren, & Sun, 2016) implemented the fastest and most efficient object recognition algorithm at the time, achieving large relative improvements ($10\% - 30\%$) over its predecessors in various image classification competitions. The team found that the number of stacked neural network layers was of crucial importance to performance. The 152-layer residual net discussed in this paper was the deepest network ever presented on the ImageNet database. Degradation problems associated with such networks were addressed through deep residual learning frameworks. One year later (Krizhevsky, Sutskever, & Hinton, 2017) further improved upon the current state-of-the-art set by the team at Microsoft. The pervading idea was that the existence of extremely large, labelled datasets and powerful computation capabilities were fundamental to the success of deep neural networks. To put things into perspective, the largest existing image database is ImageNet containing approximately 15 million labelled high-resolution images in over 22,000 categories. By introducing prior knowledge to the model to compensate for the lack of available image data and using an extremely efficient Graphics Processing Unit (GPU), (Krizhevsky et al., 2017) were able to achieve an error rate of 15.3% compared to 26.2% achieved by the second-best entry in the ILSVRC-2012 competition. These two papers are considered to be amongst the most significant breakthroughs in object detection in recent years.

To gain an appreciation and deeper understanding of object recognition, it is important to review the most influential algorithms to date. Object recognition originally took the form of image matching. Its origins can be traced back to the work of (Moravec, 1981) and (Harris, Stephens, et al., 1988) on corner detectors. It was not until 1997 that (Schmid & Mohr, 1997) showed that image matching could be extended to general object recognition problems in which a feature was matched against a large database of images. A more robust solution to object recognition was subsequently proposed by (Lowe, 1999) and further built upon in 2004 by the same author (Lowe, 2004). This method dealt with the problem of scale variance and rotation, providing accurate image matching over "a large range of affine distortion, change in 3D viewpoint, addition of noise, and change in illumination". The algorithm was known as Scale-Invariant Feature Transform (SIFT). In the ground-breaking paper by (Dalal & Triggs, 2005), a new approach to human detection was introduced: Histogram of Oriented Gradients (HOG). The method involved dividing an image into cells and counting the occurrences of gradient orientation in portions of that image. The majority of visual recognition tasks during the following decade were built on the foundations of SIFT and HOG. Convolutional Neural Networks (CNNs) gained popularity after the publication of Ross Girshick's paper on Regions with CNN features (R-CNN) (Girshick, Donahue, Darrell, & Malik, 2014). The introduction of a region proposal module before the feature extractor was responsible for the success of this algorithm. A selective search method is employed to propose bounding boxes around potential objects within an image before the neural network attempts to extract and classify specific features. The simplicity of this three-module approach sup-

plemented its popularity, however, its success was tapered by the time taken to perform feature extraction on each proposed region. The issue of speed was addressed in later publications by the same author with Fast R-CNN (Girshick, 2015) and Faster R-CNN (Ren, He, Girshick, & Sun, 2016). Fast R-CNN trains the deep network 9 times faster than R-CNN with a test time 213 times faster. Faster R-CNN is capable of detecting objects with a frame rate of 5-17 frames per second, perfectly sufficient for practical object detection systems.

The final algorithms touched on in this review are the SSD (Liu et al., 2016) and You Only Look Once (YOLO) (Redmon, Divvala, Girshick, & Farhadi, 2016). Both of these methods are simple relative to previously discussed methods and boast significant speed improvements. For this reason, they are more suited to devices with lower computational resources such as embedded systems. At the slight detriment of accuracy, YOLO treats object recognition as a regression problem. The result being extremely fast detection capabilities. Framing the problem in this manner lends itself to further end-to-end optimisation. Images can be processed at 45 frames per second using the original YOLO model. Various adaptations have been created (Redmon & Farhadi, 2017) and (Redmon & Farhadi, 2018) to address accuracy issues and further improve detection speed. SSD is based on a feed-forward CNN rather than backpropagation methods common to neural network usage. Possible boxes within each image are efficiently represented using "multi-scale convolutional bounding box outputs attached to multiple feature maps at the top of the network". This one-stage detector arrangement provides fast and reliable results. **Low computational cost, speed and the potential for optimisation suggest that the SSD algorithm shows great promise in the field of pedestrian and cyclist detection.** The primary focus of this project is to manipulate the SSD algorithm for use with computationally-inexpensive hardware and to evaluate its performance. Due to large variability, occlusion, orientation, body pose etc. this remains a challenging task (Li et al., 2016). Human detection algorithms have yet to reach the accuracy levels of the biological detection system and, as such, much work is still needed in the area (Ahmed et al., 2019). Despite this, it is important to remember that much progress has been made in improving object detection algorithms throughout the years. This can be seen in Figure 4 by the improvement in the mean Average Precision (mAP) of popular algorithms on VOC07, VOC12 and MS-COCO image datasets (Zou et al., 2019).

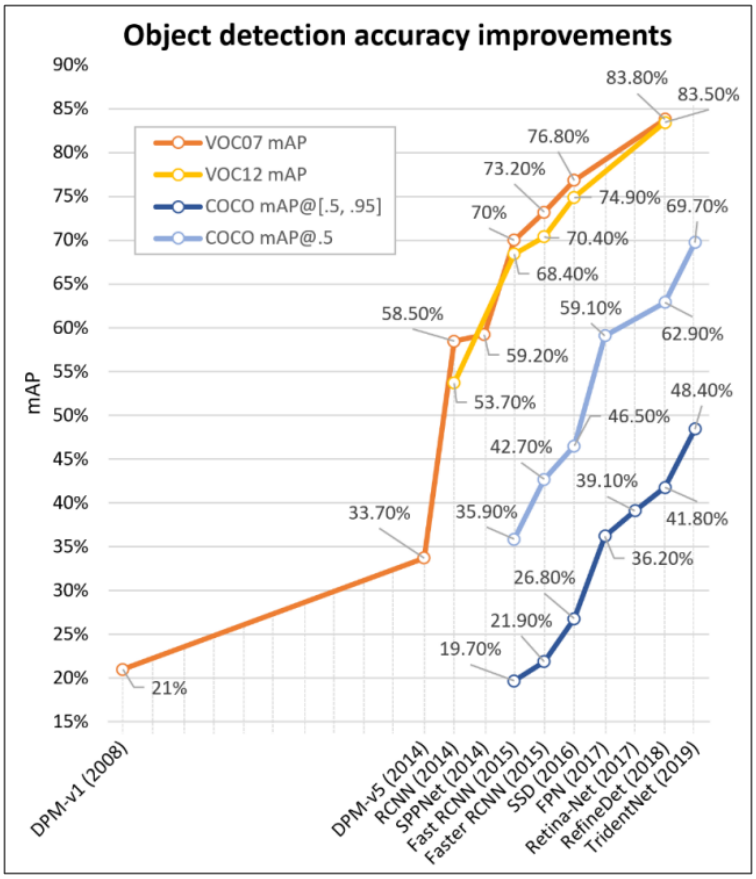


Figure 4: mAP Improvement, 2008-2019

3 Challenges

Creating a functioning human detection device, using computationally-limited resources, that can robustly deal with the variability of non-ideal environments is a significant challenge. The complexity and scope of the field of computer vision provides further obstacles to achieving the project goals. Despite refining the research area to object detection rather than computer vision as a whole, navigating the extensive, complex literature proved to be one of the biggest challenges. Furthermore, building an object detection device requires a deep understanding of esoteric concepts such as convolutional neural networks. A strong foundational background in mathematics and engineering, acquired over four years of undergraduate study, proved invaluable for this work.

As mentioned in the introduction, one of the goals of this project is to push the boundaries of the computational power of a smaller system. Memory and processing power limitations will significantly affect the accuracy and speed of the object recognition algorithm. Dealing with this will be one of the biggest challenges. Real-time webcam detection will prove particularly taxing. As such, there exists a constant trade-off between precision and speed.

Challenges more specific to image analysis also exist. Below are factors that can negatively impact the productivity of the detection algorithm and, ultimately, the success of the project.

- **Multiple Objects:** Dealing with multiple objects within a single frame is a primary challenge when attempting to successfully detect cyclists and pedestrians. Which object should be detected first? How will the detections be displayed simultaneously? Specific accommodations in the code may need to be made to allow for multiple iterations or some form of discrimination between single object and multiple object frames. Considerations about a possible increase in required computational power for an increased number of objects needs to also be taken into account.

- **Differing Object Classes:** Highly related to the previous challenge, the question of how to deal with objects that belong to different classes needs also to be addressed. Should different objects take precedence over others? How does one deal with objects outside of the pedestrian and cyclist classes? This is an important question. The primary goal is to detect only pedestrians and cyclists, however, detecting other objects such as trees and road signs may also prove useful. One also needs to determine if it is it more difficult or computationally-expensive to detect many different objects?

- **Occlusion:** Another obvious challenge is that of occlusion. If an object is not fully located within the image or is partially blocked, this can often lead to incorrect detection. The program may need some way of distinguishing between occluded and non-occluded objects. Perhaps certain adjustments need to be made for the former. One needs to weigh up whether falsely detecting an occluded object is less damaging than not detecting it at all.

- **Scale Variance:** Scale variance has long frustrated the minds of researchers in the area. The algorithm will be created in an ideal environment and evaluation and testing will be subsequently performed to test the robustness of the program. The real world environment is somewhat unpredictable and full of variability. Incoming images vary in size as do features and instances matched to the original image. Additionally, the physical size of objects within an image has great bearing on the outcome of object localisation. Small objects provide less information and require more precise classification techniques to be detected correctly. To function beyond the simulated environment, alterations need to be put in place to account for this.

- **Object Rotation:** Object rotation exacerbates this problem. Objects may seem smaller or larger at different angles. Human poses and variations in physical attributes can be tough to deal with. Contrasting this to detecting household objects or road signs one can see why human detection is

particularly difficult.

- **Poor Camera Resolution:** The camera quality of the embedded system is of poor quality. This challenge is coupled with that of poor computational power. Higher quality cameras of course exist but the project objectives need to be kept in mind. **The most accurate model possible is not desirable but rather the most accurate model relative to hardware constraints.** Poor resolution means fewer pixels comprise each image which, in turn, means the algorithm has less data to work with. There is a distinct proportional relationship between camera resolution and detection accuracy.

- **Training vs Test Data:** Differences between the dataset on which the model was trained on and data with which the model must work with can be a primary source of inaccuracy. This is common across most machine learning tasks. None of the images the device will take in will be part of the training data. This is because real-time images of the road are being dealt with. One must strike a balance between overfitting and underfitting. Researchers and experts in related fields have been making advancements in this area in order to alleviate the problems associated with training vs test data. A popular solution is cross-validation.

4 Important Concepts and Definitions

Within the area of object recognition there exists many analogous terms which often leads to their misuse.

- **Image Classification:** A machine learning model will be trained to recognise a set number of object classes. Image classification involves assigning one of these class labels to an object within an image inputted into the model.

- **Object Localization:** This refers to locating an object in a image by, for example, drawing a bounding box around it. As mentioned previously, algorithms like R-CNN use region proposals to assist this process. Single-stage detectors such as SSD do not separate the two tasks of localisation and classification. The combination of these two tasks is called object detection.

- **Object recognition:** All of the above problems are referred to as object recognition. This is analogous to processes within the brain where humans quickly retrieve and 'recognise' environmental, visual information.

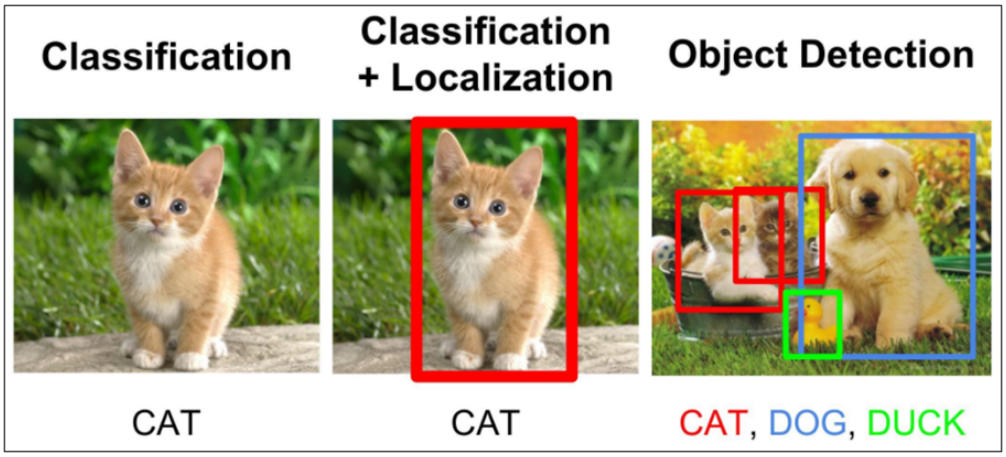


Figure 5: Classification vs Localisation vs Detection (Ouaknine, 2018)

- **Regression:** Regression is used to determine the relationship between a dependent and independent variable. It forms the basis of many important statistical concepts and finds particular use in predicting variables beyond the range of existing data. The three most common types are simple, multiple and polynomial regression. In its general form, it can be represented mathematically as:

$$Y = f(X, \beta) + e$$

  Where Y is the dependent variable, f is a function, X is the independent variable, β represents the unknown parameters and e is the error term.

- **Clustering:** Clustering models are used to divide each record of a data set into one of a small number of similar groups. They can be very useful for image analysis and pattern recognition. The three main types are partitioning, hierarchical and density-based clustering.

- **Overfitting:** This constitutes a discrepancy between a trained model's ability to perform on training data and its ability to perform on test data which it has had no exposure to. As mentioned previously, due to the fact that no test data will match the training data, this is something to be wary of when choosing the appropriate model. The higher the dimensionality of a model, the higher the risk of overfitting. Due to the smaller scale of the project, this is not as pressing of an issue.

- **Feature Map:** As filters act on each layer of the network, the appropriate neurons are activated and the output is collected in a feature map. Each feature map is assigned a particular feature such as a

straight line. Objects are recognised by the iterative process of combining features. An example of a feature map for a car is seen in Figure 6.

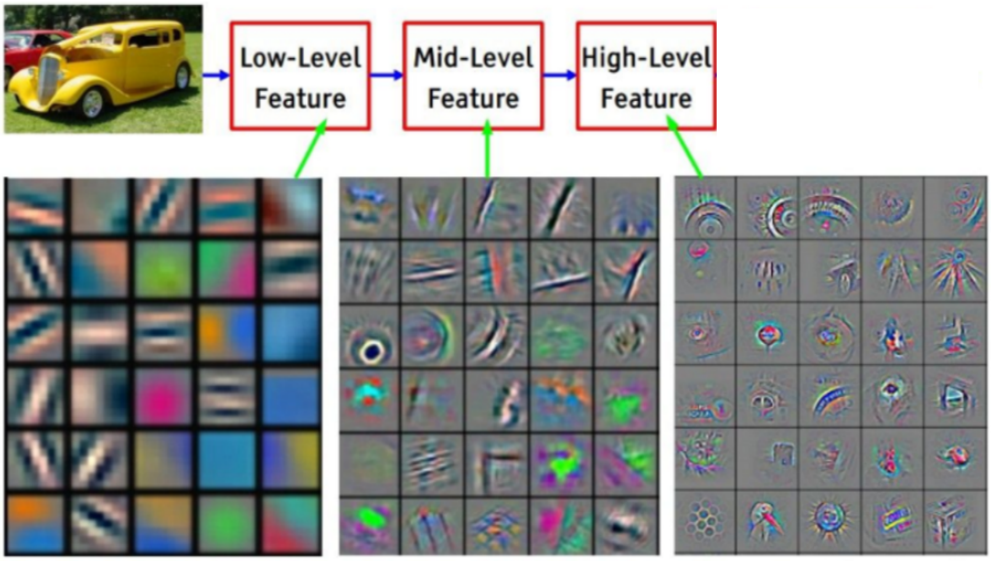


Figure 6: Typical Feature Map (Kevin, 2018)

- **Frames per Second:** Frames and images are synonymous in this context. This is a quantitative measure of the object detection speed of an algorithm. It is a measure of how many images the algorithm can process within one second. It is an important figure that can be used to compare the performance of differing algorithms.

- **Data Compression:** Images are simply represented digitally as an array of numbers called pixels. Each pixel typically holds three numbers defined as Red Blue Green (RGB) or Hue Saturation Luminance (HSL), each pertaining to a particular quality of that pixel. This is represented below in Figure 7. One single image is, therefore, often fully represented by lots of data.

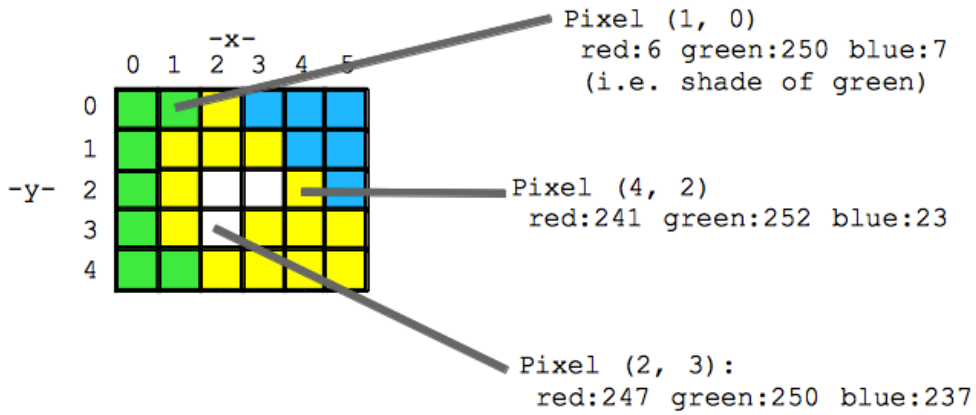


Figure 7: RGB Image Pixel Arrangement (Stanford.edu)

For increased computational efficiency, encoders typically compress the original image to be analysed.

Many lossless (JPEG) and lossy (Bitmap) compression algorithms exist. Lossless algorithms make use of statistical redundancy when reducing the overall number of bits. Lossy algorithms try to shed unnecessary or less important data. The Discrete Cosine Transform (DCT) is a very important concept for image compression. As images are simply large matrices, operations such as matrix transformations can be used with good effect. Decoders perform the reverse operation to encoders at the output of the data manipulation process.

- **Confidence Level/Classification Accuracy:** This value represents the confidence the model has that it classified the particular object correctly. It is typically given as a percentage figure and displayed alongside the object class in an outputted image. Naturally, a primary goal of object detection is to achieve a high confidence level. Much research has been done on this topic.

- **Non-Maximum Suppression:** This is a technique (parameter) that determines the strictness of possible object detection filtering. The intersection area with other potential objects and running confidence scores are typically used to refine the number of possible objects within an image. Much research has been done on optimising the process of non-maximum suppression. The popular paper (Neubeck & Van Gool, 2006) describes efficient methods of preforming this technique.

- **Mean Average Precision (mAP):** Precision is simply a measure of how many correct predictions a model made. Recall is a measure of how well positives were detected. Plotting these two against eachother and finding the area under the resulting curve yields the average precision. mAP is this figure averaged over a range of classes or iterations. It is a popular performance benchmark for state-of-the-art object detection algorithms.

5 Methodology

5.1 Software

5.1.1 AI, Machine Learning and Deep Learning

Below are some important concepts pertaining to deep learning, computer vision and object recognition. A thorough understanding of such concepts is vital to progress in this project.

It is easy to confuse the related ideas of artificial intelligence, machine learning and deep learning. Figure 8 (Williams, 2019) diagrammatically explains their differences which effectively boil down to sets and subsections.

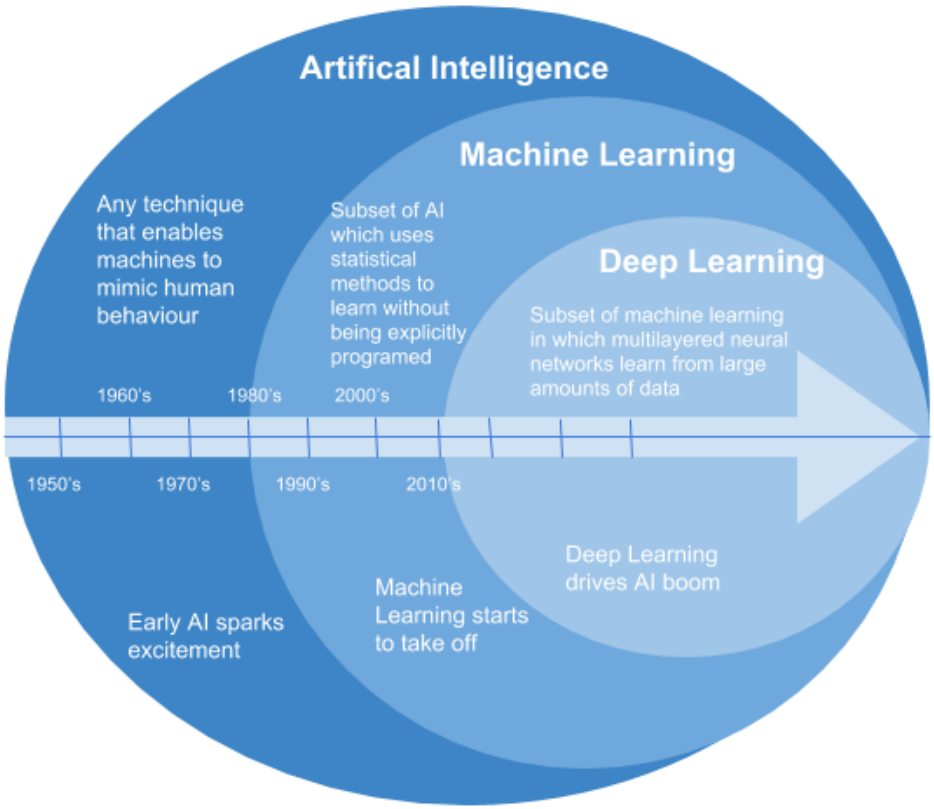


Figure 8: AI, Machine learning and Deep Learning Distinctions (Williams, 2019)

Artificial intelligence includes everything that allows computers to learn how to solve problems and make intelligent decisions. This is the broadest field and finds application from vehicle automation to Apple's Siri. Machine learning is what enables machines to solve problems on their own and make accurate predictions using the provided data. Machine learning uses algorithms known as models to identify patterns in the data. The process by which the model learns these patterns from data is called model training. Predicting outcomes based on these processes is called inference. Recommender systems for the likes of Netflix and

Youtube are examples of machine learning algorithms.

Many levels of processing exist in the brain. With regards to vision, each level tries to understand features at varying levels of abstraction (Murphy, 2012). Deep learning attempts to replicate this structure. A neural network is a collection of small computing units called neurons that take incoming data and learn to make decisions over time. Deep learning is a specialized subset of machine learning that uses layered neural networks to simulate human decision-making. A deep neural network (DNN) is a collection of neural networks with multiple layers between the input and output layers. Considered as a revolutionary thinker in the field of deep learning, Yann LeCun suggests that, "Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction" (LeCun, Bengio, & Hinton, 2015). It is prophesised in this paper that unsupervised learning (where input and output data is not provided by a human) will play a significant role in deep learning applications in the future. Deep learning has supplemented significant advancements in image and speech processing over the last decade.

CNNs have proven to be particularly efficient for image processing. A basic CNN representation can be seen in Figure 9 (Phung, Rhee, et al., 2019). Five layers make up the body of the network: the input, the convolution layer, the pooling layer, the fully-connected layer and the output. The convolution layer convolves the input image with weighted kernels. The sigmoid function is typically used to compress the weighted values between 0 and 1. The pooling layer reduces the dimensionality of the image whilst retaining its original information. The resulting output takes the form of a feature map. Extracted features are then combined in the fully-connected layers and classified as a single neuron in the output layer. Each output neuron signifies an object class for example 'car' or 'pedestrian' and often contains information on the confidence that the image is within this particular class. This is an oversimplified representation. In reality, there can exist many variations and copies of each layer type.

5.1.2 MobileNetV3 + SSDLite Model

There exists many efficient object recognition models designed specifically for mobile and embedded systems. The advancements discussed in the literature review do not necessarily improve the efficiency of object recognition with respect to size. **Often recognition tasks for real world applications are limited both by computational availability and time.** There has been a significant focus lately on building small, low latency models to account for these limitations in an attempt to create practical, rather that purely theoretical, detection systems. One particularly successful attempt has been the combination of two mobile-friendly models: MobileNetV3 (A. Howard et al., 2019) and SSDLite (Sandler, Howard, Zhu, Zhmoginov, & Chen, 2018). The aim of these, and similar, models is often to drastically increase detection speed at the

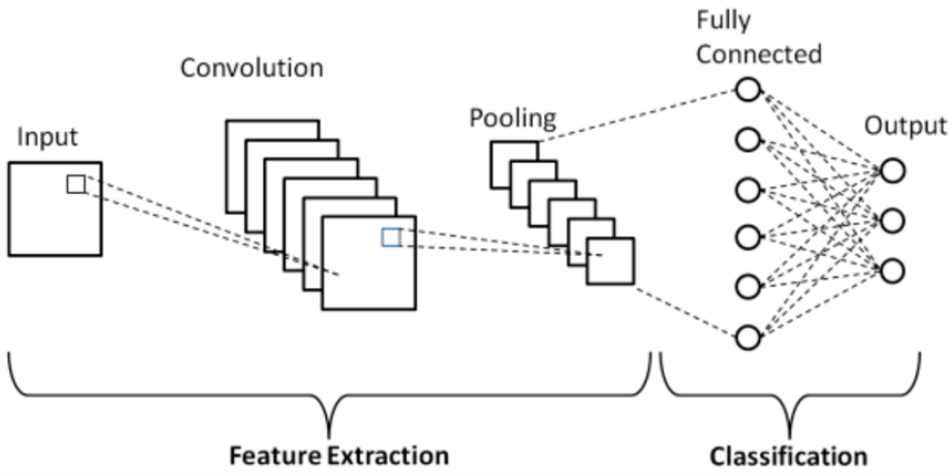


Figure 9: Basic CNN Architecture (Phung, Rhee, et al., 2019)

detriment of a disproportionately smaller reduction in detection accuracy. The aforementioned combination of models was chosen for this project due to their proven efficiency and versatility (Sandler et al., 2018). A large and small version of the model exist, each referring to the overall number of parameters and operations performed by the model in detection tasks. Each suit different applications. The large version is used for this project. Another promising aspect of the MobileNet model is the consistent release of improved versions demonstrating an interest amongst industry experts and an overall promising amount of potential for future advancements. Described below is some background into the SSDLite model and the third and latest version (as of writing) of MobileNet. The MobileNet model is used as a backbone for the larger SSDLite model.

The efficiency of MobileNet's architecture (A. G. Howard et al., 2017) ultimately lies in the reduction of network parameters. Models with fewer parameters incur a lower level of computational cost. This is reflected in Figure 10 where previous versions of the MobileNet model uses fewer parameters and operations (MAdd) when compared to similar computationally efficient algorithms. Achieved performance is also comparable as seen by the mAP figures.

Network	mAP	Params	MAdd	CPU
SSD300[34]	23.2	36.1M	35.2B	-
SSD512[34]	26.8	36.1M	99.5B	-
YOLOv2[35]	21.6	50.7M	17.5B	-
MNet V1 + SSDLite	22.2	5.1M	1.3B	270ms
MNet V2 + SSDLite	22.1	**4.3M**	**0.8B**	200ms

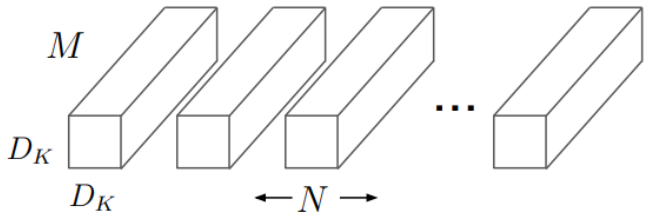
Figure 10: Performance vs Parameters State-of-the-Art Models (A. G. Howard et al., 2017)

As the name suggests, convolution plays a pivotal role in the structure of CNNs. An alternative convolution structure is designed for MobileNet which dramatically reduces computational complexity whilst retaining comparative levels of accuracy to the current state-of-the-art algorithms. Standard convolution for typical

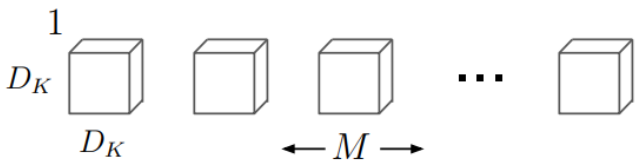
neural networks is computationally very expensive. In this context, the computational cost of standard convolution can be mathematically formulated as:

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F \tag{1}$$

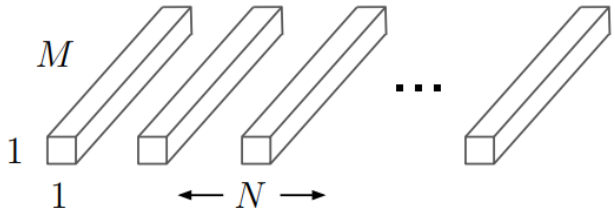
As illustrated by Figure 5(a), D_k is the spatial width of the convolution kernel, D_F is the spatial width of the feature map derived from the input, M is the number of input channels and N is the number of output channels. All kernels and feature maps are assumed to be square. Cost is multiplicatively dependent on each of these terms which leaves much room for potential improved efficiency.



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 11

The MobileNet architecture addresses this by factorising the standard convolution into two separate convolutions in a process called depthwise separable convolution. The two layers are known as depthwise convolutions and pointwise convolutions and can be seen in Figure 11. As is analogous to standard convolution, the depthwise convolution layer Figure 5(b) applies a single lightweight filter to each input channel

without combining them to create features. The computational cost can be seen as:

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F \tag{2}$$

An additional simple 1x1 convolution layer Figure 5(c) is then applied (pointwise convolution layer) to compute a linear combination of the previous layer's output. The cost of this layer is:

$$M \cdot N \cdot D_F \cdot D_F \tag{3}$$

Computing the reduction in computation using (1), (2) and (3):

$$\frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F} = \frac{1}{N} + \frac{1}{D_K^2}$$

MobileNetV3, built upon previous versions (Sandler et al., 2018), uses three depthwise separable convolutions resulting in a computational cost of 8 to 9 times smaller than standard convolution. Its accuracy has experienced a relatively small reduction. The introduction of two hyperparameters further decreases size and increases speed of the model. The width multiplier reduces the size of each network layer and the resolution multiplier reduces the size of the original input image. Both parameters quadratically reduce computational cost. Furthermore, MobileNetV3 uses linear bottleneck layers to avoid the destructive effects of nonlinearity on information retention. Figure 12 illustrates the superiority of the large third version over previous versions in terms of performance and complexity. Latency refers to the time taken to process one unit of data.

Similar adaptions with depthwise separable convolutions are made on the regular SSD algorithm to produce the mobile-friendly SSDLite (Sandler et al., 2018). These adaptions result in a drastic reduction from 14.8 million parameters in the original SSD algorithm to 2.1 million parameters in SSDLite.

The particular MobileNetV3 + SSDLite model used in this project has been trained on the COCO (Common Objects in Context) dataset which contains 330,000 images in 80 common object categories. The versatility of the model can be attributed to this broad range of object classes. This, in turn, allows for a wider range

Backbone	mAP	Latency (ms)	Params (M)	MAdds (B)
V1	22.2	228	5.1	1.3
V2	22.1	162	4.3	0.80
MnasNet	23.0	174	4.88	0.84
V3	22.0	137	4.97	0.62
V3†	22.0	119	3.22	0.51
V2 0.35	13.7	66	0.93	0.16
V2 0.5	16.6	79	1.54	0.27
MnasNet 0.35	15.6	68	1.02	0.18
MnasNet 0.5	18.5	85	1.68	0.29
V3-Small	16.0	52	2.49	0.21
V3-Small†	16.1	43	1.77	0.16

Figure 12: Performance vs Parameters MobileNet Models (A. G. Howard et al., 2017)

of device functionality. Included within these classes are 'people' and 'bicycles'. The distinction between 'bicycles' and 'cyclists' will be dealt with in further sections. As such, this model provides a strong foundation to build an object detection device tailored to the particular needs of the project.

5.1.3 Python & Jupyter Notebooks

The importance of the choosing the correct model and building a deep understanding of its mechanics cannot be overstated. Building a strong and appropriate foundation will significantly aid future work and, ultimately, dictate the success of the project. At this point, the most suitable object detection model has been chosen based on extensive research, bearing in mind the needs of the task at hand. With this, comes the capability to take in a single image as an input and to detect objects belonging to the specified class range with a level of accuracy comparable to other state-of-the-art algorithms. This basic function will be manipulated and adapted to form the basis of all future work and ensure the specifications of the project aims are met. Up until this point, most of the work has been **theoretical and research-based**. The task now becomes **creating something tangible** from this capability and harnessing the power of this efficient model.

A programming language needed to be chosen. Due to its popularity and its broad range of available libraries, Python was chosen over other potential choices such as R and MATLAB. Python has established itself as a leading language for machine learning tasks with libraries/packages such as Scikit-Learn, TensorFlow and Keras. The following work required a strong knowledge of Python and, as such, independent learning took much of the focus at this point. Jupyter Notebooks was chosen as a platform to implement the Python language due also to its popularity and user-friendly interface. Additional familiarity with using this platform was also required.

5.1.4 OpenCV

Numerous machine learning packages were mentioned in the previous section. These provide extremely useful functionality for a broad range of machine learning tasks. These more general libraries would suffice for the task at hand, however, it was decided to adopt a package tailored more towards computer vision. In this way,

one can make use of the specificity of the program, without worrying about the loss of functionality in less relevant areas. It was decided that OpenCV was the most suitable for the task. This cross-platform library provides real-time optimised computer vision tools and is compatible with Python. OpenCV is particularly useful for object detection but also finds application in augmented reality, facial recognition and robotics.

The first step involves importing OpenCV and other necessary supplementary libraries into the Jupyter Notebook environment. NumPy supports large, multi-dimensional arrays and matrices and boasts a large collection of high-level mathematical functions. It is used extensively throughout the code[1]. Next, a text file 'coco.names' containing an ordered list of all the object classes is read opened and parsed. These classes correspond to the 90 objects contained within the COCO dataset as discussed previously. Each individual class is stored in the variable 'classNames' having been separated by the newline character.

```
classNames= []
classFile = 'coco.names'
with open(classFile,'rt') as f:
    classNames = f.read().rstrip('\n').split('\n')
```

The MobileNetV3 + SSDLite model is then incorporated into the code using the 'dnn_DetectionModel' function which supports the most popular object detection algorithms including SSD. This function takes in two arguments: a text file containing the network (model) configuration and a binary file containing the trained weights for the model. These files are sources from an online GitHub repository. A set of parameters, dictating the necessary scale and pixel intensity of the inputted image, accompanies the creation of this function. This function returns 1. the **class indexes** of any objects detected within a single image, 2. a set of corresponding **confidence levels** for each detection and 3. a set of **bounding boxes**, localising each detection.

```
configPath = 'ssd_mobilenet_v3_large_coco_2020_01_14.pbtxt'
weightsPath = 'frozen_inference_graph.pb'
net = cv2.dnn_DetectionModel(weightsPath,configPath)
net.setInputSize(320,320)
net.setInputScale(1.0/ 127.5)
net.setInputMean((127.5, 127.5, 127.5))
net.setInputSwapRB(True)
```

A lot of the information necessary for a single image detection has now been extracted from the model. There still remains significant threshold and parameter choices and general manipulation design in order to achieve the desired results. The largest user-made function 'getObjects' deals with the majority of this. A key decision is to be made on the threshold of confidence level for each detection. If the confidence level of a

[1] An overview of the main features of the code is provided in this report. The more granular details are outlined within the code itself through comprehensive commenting.

particular detection falls below this threshold, then it will not qualify as a successful detection. Confidence levels range between 0 - 100, 0 meaning that the model is certain that the particular object within the class range is not in the image and 100 meaning that it is certain that the object in question is contained within the image. An initial value of 0.45 was chosen so as not to eliminate potential learning opportunities from poor or false detections. This provides the user with more data to work with. This value constitutes one of the arguments of the function 'getObjects'. Following the same logic, a low non-Maximum Suppression (nms) value is initially chosen (0.3) and inputted as an argument to the same function. Naturally, the main argument of the function is the image object itself, on which the detection model will work.

An additional argument 'objects' is inputted into the 'getObjects' function. This allows the user to specify any particular object the model should detect. All other objects will be ignored. In this case, 'person' and 'bicycle' were chosen as the desirable object classes. If this variable is left empty, then all original classes are able to be detected. As mentioned previously, the 'net' variable returns the class IDs of each object detected as one of its outputs. For example, if a 'person' is detected, then a class ID of 1 is returned as this is the first object in the 'classNames' variable derived from the 'coco.names' file. If the corresponding name of this object is also in the 'objects' variable, then the desirable object has been successfully detected. It is important to remember that this is also subject to the threshold and nms parameter values previously specified. The next task involves relaying this information to the user. For this the bounding box output from the 'net' model is used. This is an extremely important output which enables a lot of interesting functionality as seen in further sections. This variable is in the form of a NumPy array containing four benchmark pixel values of the resulting bounding box: 1. Top left corner x pixel value, 2. top left corner y pixel value, 3. width of the box and 4. height of the box. It is important to be clear that these are the coordinates of a bounding box around a particular object and not the coordinates of the object itself. As opposed to a standard Cartesian coordinate plane, the origin (0,0) is located in the top left corner of an image as seen in Figure 13. This means that the lower the object is located within the image, the larger the y coordinate value is.

These coordinates are used to display the original image with bounding boxes around each detected object as well as the class name and confidence level of the detected objects.

```
cv2.rectangle(img,box,color=(0,255,0),thickness=2)
cv2.putText(img,className.upper(),(box[0]+10,box[1]+30),
cv2.FONT_HERSHEY_COMPLEX,1,(0,255,0),2)
cv2.putText(img,str(round(confidence*100,2)),(box[0]+50,box[1]+60),
cv2.FONT_HERSHEY_COMPLEX,1,(0,255,0),2)
```

To run the above code, one simply needs to call the 'getObjects' function on an existing image. Images are read using the 'imread' function in OpenCV. Similarly, they are displayed on a separate output screen

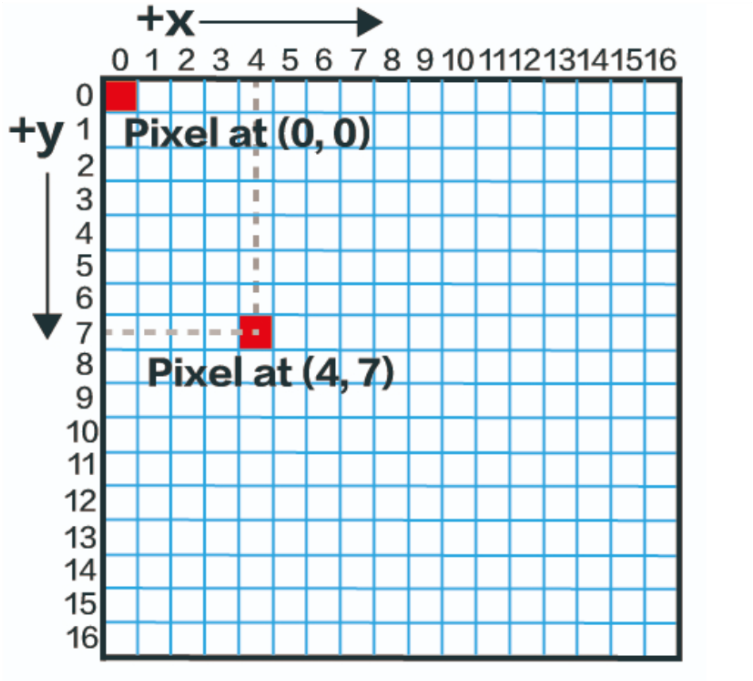


Figure 13: Pixel Coordinate Layout for Typical Image

using 'imshow'. 'imwrite' is used to save the resulting image containing detected objects to the current file directory. The 'waitKey' and 'destroyAllWindows' functions ensure that the output screens are correctly closed once the user presses any key.

```
image = cv2.imread("Image1.jpg")
...
result,objectInfo,c2,c3 = getObjects(image
,0.45,0.2,KNOWN_PEDWIDTH,KNOWN_CYCWIDTH, focalLength, objects=['person','bicycle'])
cv2.imshow("Output", image)
cv2.imwrite("distance_test.jpg", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

5.1.5 Multiple Objects & Video Extrapolation

To summarise, the model can now accurately detect user-specified objects within a still image, subject to certain parameters, and convey useful information to the user about the location and confidence level of each object. The 'net.detect' function was chosen so as to effectively and simply deal with the challenge of multiple objects. A typical example of what the model may have to deal with is seen in Figure 14.

Figure 14: Typical Image Containing Multiple Pedestrian and Cyclist Objects

```
classIds, confs, bbox = net.detect(img,confThreshold=thres,nmsThreshold=nms)
```

The function iteratively detects multiple objects, possibly of differing classes, within a single image. One of the most crucial questions still remains. How does one extrapolate the existing functionality to video/webcam inputs? The function 'VideoCapture' is used for this. This takes in either a video, sequence of images or webcam ID as its input and outputs individual frames and a binary variable signifying whether the function is operating correctly. The versatility of this function to cater for the aforementioned inputs means that one does not have to worry about dealing with videos and webcam inputs separately. This is important as the ultimate goal is to use a webcam input connected to a separate hardware device. As with the 'net' variable, multiple parameters are set initially for the 'VideoCapture' function. These refer to the width and height of outputted frames. A while loop is used to iterate over these frames and to call the 'getObjects' function at each iteration. Output display follows a similar structure to a single image input as mentioned in the previous section. The 'VideoCapture' object must be released once the user has pressed the 'q' key to ensure correct termination of the program.

```
cap = cv2.VideoCapture(0)
cap.set(3, 640)
cap.set(4, 480)
...
try:
    while True:
        success, img = cap.read()
        if success == 1:
            result,objectInfo,c1,c2 = getObjects(img
,0.45,0.2,KNOWN_PEDWIDTH,KNOWN_CYCWIDTH,focalLength,objects=[])
            vid.write(img)
```

```
            ...
            cv2.imshow("Output", img)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
        else:
            break
    cap.release()
    vid.release()
    cv2.destroyAllWindows()
```

An important part of any software-related project is dealing with bugs within the code and possible errors scenarios that may arise. The issues dealt with, and functionality added, as described in this and future sections, improves the **robustness** of the code. **Error handling**, such as the try and except block and the use of the binary 'success' variable, complement this robustness and ensure that the algorithm can deal with potentially unusual/harmful inputs.

5.1.6 Distance Measurement

Having dealt with webcam inputs and potential multiple objects within a single frame, the next step involves making design and functionality choices about the nature of detection for this particular application. Detecting pedestrians and bicycles on the road is one thing, but identifying dangerous situations involving pedestrians and bicycles is another. The device must alert the driver of potential hazardous situations based on the objects detected. The approach taken to solve this problem in this project, is to **evaluate the situation based on the distance of objects from the camera lens and, in turn, the vehicle itself**. This section deals with how these distances are constantly measured whereas the following sections deal with pinpointing dangerous situations from these measurements and also evaluating its accuracy.

An approach called **triangle similarity** is used to achieve accurate distance measurement. One must keep in mind the complexity of this task however. Measuring distances of objects from a single camera lens without the use of additional equipment such as sensors and trackers is a very difficult task. As such, the possibility of extremely accurate measurements is low. This is not too much of an issue for this project as discussed in the following sections. The theory behind the measurement is obtained from (Rosebrock, 2015). This technique is based on a more advanced method (Khan, Paul, Rashid, Hossain, & Ahad, 2020) which was deemed superfluous for the task at hand.

The method revolves around Equation (4):

$$D = \frac{W.F}{P} \tag{4}$$

Where D = distance from camera, W = width of object, F = focal length of camera and P = pixel width of detected object in image. This is reflected by the 'distance_to_camera' function in the code below.

```
def distance_to_camera(knownWidth, focalLength, perWidth):
    return (knownWidth * focalLength) / perWidth
```

The first step involved initial calibration to determine the focal length of the camera. To obtain this, 10 images containing a single object were captured by a camera. The distances of each object to the camera lens in each image were measured beforehand. Likewise, the width of each object was measured. To obtain accurate results, a large range of objects with high width and distance variability was used. The pixel width was extracted from the bounding box output of the 'net' variable. This corresponds to the third element in this NumPy array. Equation (4) was then rearranged and the above figures subbed in to obtain a focal length value for each image. These values were averaged to obtain a more accurate value. This value could now be used in subsequent measurements to obtain the unknown variable D of detected objects. Using this bank of images and corresponding table of variable values, one could now calculate the focal length of any camera.

Problems with inaccuracies arise from the W variable. A single value each for the width of pedestrians and cyclists was needed. Average values for the width of humans (males and females) and the width of bikes were obtained from various sources (Ross, 2002), (Reference.com, 2020), (Watson, 2018) and (in Architecture, n.d.). These values were taken as 68inches and 15inches respectively [2]. There exists huge variability in the widths of humans in particular, especially taking children into consideration. Different bicycle types and brands also yield a lot of different width values. Furthermore, as discussed in the 'Challenges' section, rotation and occlusion further exacerbate the problem. A pedestrian facing the direction perpendicular to the camera for example will have a significantly smaller width value and, as such, their distance value will be distorted. This issue could be resolved with multiple cameras or more equipment as previously mentioned. This topic is discussed towards the end of the report. Due to the scope of the project and its limitations as outlined at the outset, the distance measurements obtained were deemed acceptable. There exists a constant balance between complexity and functionality throughout the project. Striking this balance correctly is often difficult, however, understanding the theory behind the functionality and recognising potential methods to build upon it, alleviates some of the downsides of settling for a simpler method.

[2]This bicycle width value suits cameras facing out from the sides of the vehicle rather than out of the front window. Since this is a general purpose device and its designated orientation is not specified, this was deemed acceptable.

Finally, as with the class names and confidence levels, this information needs to be presented to the user. This is done in a similar fashion to before.

```
cv2.putText(img,"%.2f inches away" % dist,(box[0]+50,box[1]+300),
cv2.FONT_HERSHEY_COMPLEX,1,(0,255,0),2)
```

5.1.7 Cyclist Detection

The dataset the model is trained on contains an object class labelled 'person' and another labelled 'bicycle'. As is consistent with the aims of the project, a decision must be made on how to use this information to detect the object class 'cyclist'. A real-life situation where this distinction becomes necessary can be seen by the juxtaposition of Figure 15 and Figure 16.

Figure 15: Separate Person and Bicycle

Figure 16: Near Person and Bicycle i.e. Cyclist

Given the variables and functions already existing in the code, many methods exist to achieve this goal. The chosen method involves:

1. Locating the centre pixel coordinates of each 'bicycle' object detected within a single frame.
2. Storing each of these coordinates separately.
3. Locating the centre pixel of the first 'person' object detected within a single frame.
4. Measuring the Euclidean distance between this pixel and all stored 'bicycle' centre pixels.
5. If this distance lies below a particular user-specified threshold then change the label of the 'person' to 'cyclist' and remove any displayed information (bounding boxes, class name...) for the particular 'bicycle' object the pedestrian is close to.
6 . Repeat steps 3-5 for all 'person' objects within the image.

This is diagrammatically represented by Figure 17.

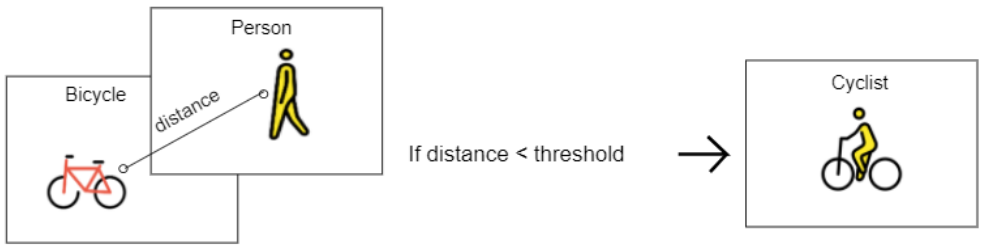


Figure 17: Method for Cyclist Detection

This procedure effectively checks whether a person is close enough to a bicycle to be sure that the combination of the two is, in fact, a cyclist. A general function is created to measure the Euclidean distance between two points.

```
def distt(x1,y1,x2,y2):
    return np.sqrt(np.square(np.absolute(x2-x1)) + np.square(np.absolute(y2-y1)))
```

The centre pixel was naturally chosen as a good point that is representative of the location of each object. It is obtained using the bounding box variable. The x and y coordinates are found separately using the top left corner coordinates and adding half of the width or height of the box respectively.

```
for i in range(classIds.shape[0]):
            if classIds[i] == 2:
                counter = i
                box_temp = bbox[i]
                cyc_centrex = box_temp[0]+(box_temp[2]/2)
                cyc_centrey = box_temp[1]+(box_temp[3]/2)
```

The centre of a person's body and a bicycle will be very close if the person is physically on the bike, therefore, a reasonably small threshold value was chosen (more in 'Evaluation' section). Fortunately, the repercussions of incorrectly detecting a cyclist instead of a bicycle/person are quite minimal. In this case, two objects in close proximity will be detected as opposed to one which can still potentially lead to successful determination of a dangerous situation. Note, having changed the labels for previously detected 'cyclist' objects, these are no longer considered for future iterations in the code. This helps to deal with the issue of multiple objects within a single frame, prevalent throughout the project. Various other potential solutions to this problem were considered:

- Another possible approach would have been to retrain the model on a modified dataset including cyclist images instead of/as well as bicycle images. This was deemed extraneous as retraining an existing model, especially one as large as this one, involves significant alterations and a very large database of cyclist images. Additionally, as mentioned previously, the repercussions of unsuccessfully separating cyclists from bicycles are minimal.

- Having previously measured the distance of each object from the lens, this value could have been tracked over time and, hence, a value for the speed of each object could have been determined. One could then differentiate between faster and slower objects and infer that the former be labelled as cyclists and the latter as pedestrians. There are various problems with this approach. It significantly adds to the complexity of the algorithm with no real improvement in accuracy. Additional equipment would likely be required to obtain an accurate result. As mentioned previously, the stakes for this functionality are quite low. This method would also delay the decision output as at least two frames would be needed to calculate the speed.

- The simplest approach would have been to leave the detection as is and simply change all 'bicycle' labels to 'cyclist' labels. This relies on the assumption that there will be few bicycles left unattended on the road and falsely detecting those that are, would have few real consequences. This was viewed as a lazy approach, relying too heavily on the original model detection capabilities.

From the various code snippets and descriptions of the sections above, it is easy to see the usefulness of pre-existing functions within OpenCV. This highlights the importance of choosing the correct libraries/platforms and justifies the time spent on performing the necessary research to do so.

5.1.8 Flowchart

A flowchart of the entire code is given below. This diagrammatically sums up the sections above in a manner that is digestible for an uniformed audience.

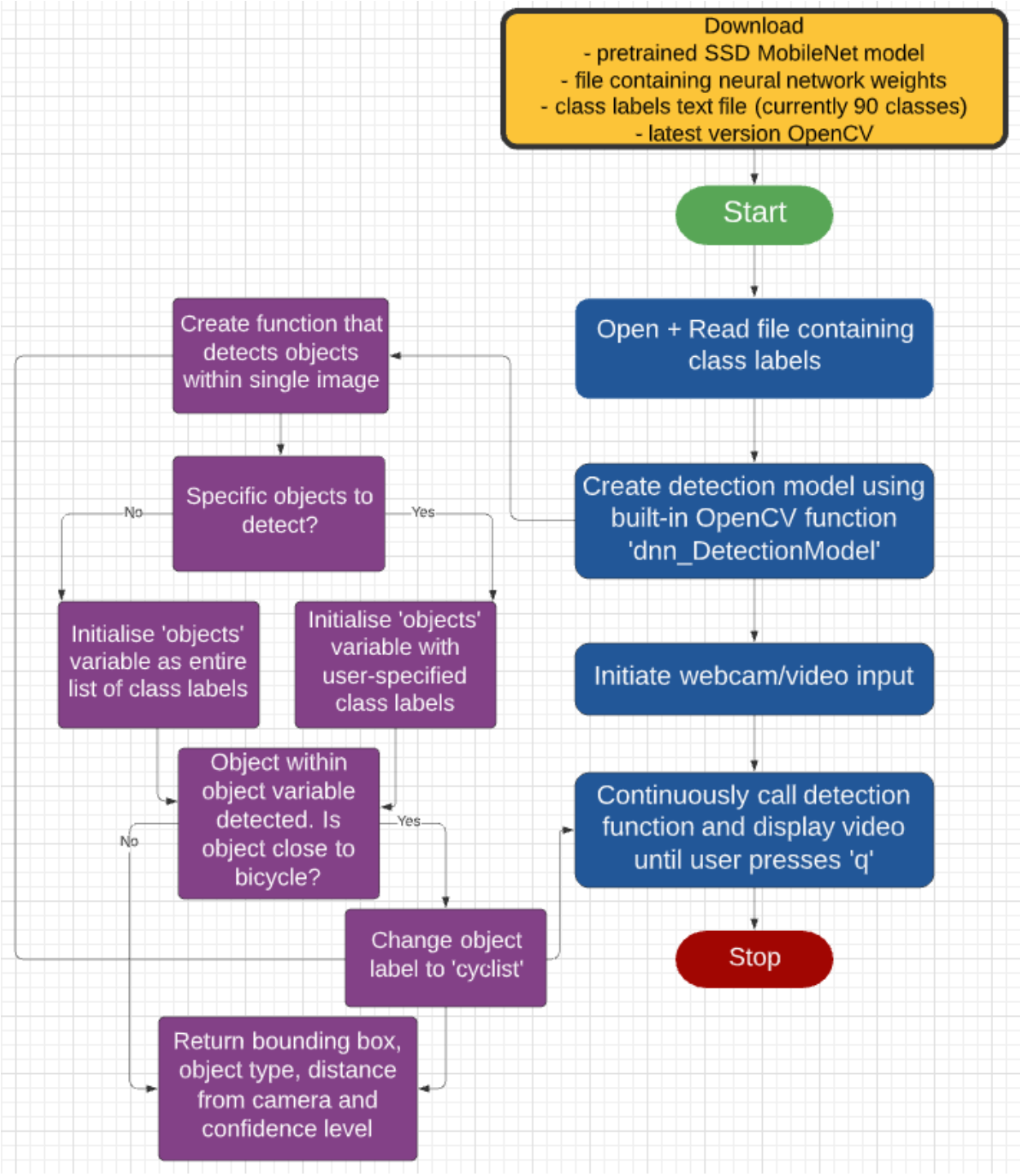


Figure 18: Code Flowchart

5.2 Hardware

5.2.1 Raspberry Pi

One of the main project aims was to test the limits of computationally inexpensive systems. These find application as embedded systems, components of a larger system such as an autonomous vehicle. This approach was chosen as it stays within the scope of a project of this nature and provides a more realistic basis for creating a multi-functional device given the resources available. Using less powerful hardware adds to the simplicity of the task at hand but at the expensive of potential performance. Choosing the right hardware can help alleviate the extent of this trade-off.

One of the most popular single board computers is the Raspberry Pi (Rpi). It is shown in Figure 19 and is effectively a tiny computer.

Figure 19: Raspberry Pi 4 Model B

Many models exist with slightly varying capabilities. The model used in this project is the Rpi 4 model B which uses the Linux operating system. This model boosts significant performance improvements over previous models as seen by Figure 20 and Figure 21. The Linpack benchmark is a measure of a system's computing power. More specifically, it measures how fast the computer can solve dense linear equations (Dongarra, Luszczek, & Petitet, 2003). The second figure deals with the computer's image editing capabilities which is particularly relevant to the image-heavy tasks of this project. It uses the GIMP software as a platform to compare models.

It is equipped with a 1.5 GHz 64-bit quad core ARM Cortex-A72 processor, multiple USB ports, a pair of 4k resolution HDMI ports and WI-FI and Bluetooth capability. It is still outperformed by personal computers but its compact and versatile nature finds many practical uses, for example as a radio transmitter, WI-FI booster and motion detector. In this project, the Rpi will facilitate the use of the MobileNetV3 + SSDLite object detection model. It can provide insight into the realistic implementable power of this state-of-the-art algorithm without the use of powerful GPUs and other computational resources. As such, it is useful to gauge the general feasibility of similar algorithms using restricted hardware.

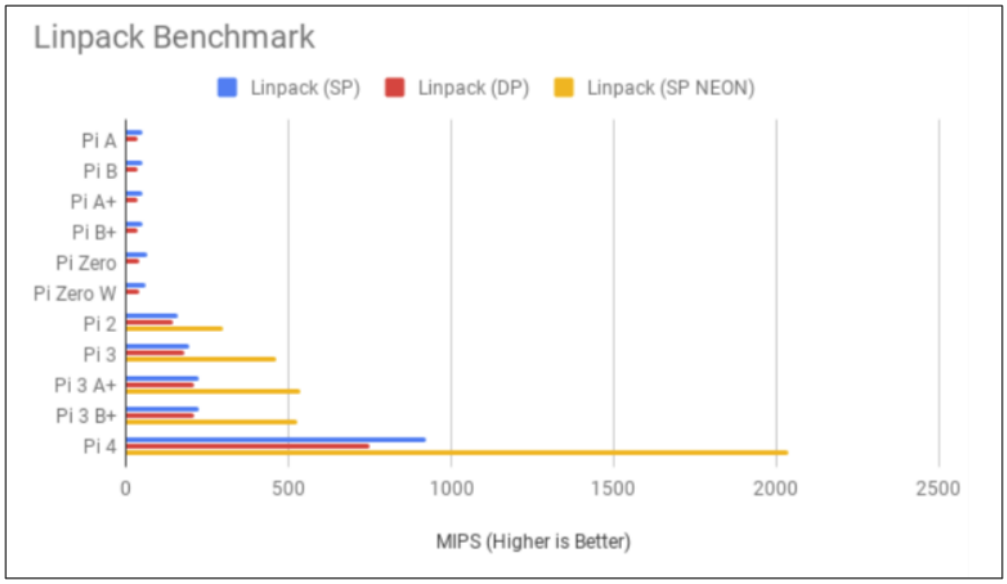


Figure 20: Model Comaprison: Linpack Benchmark

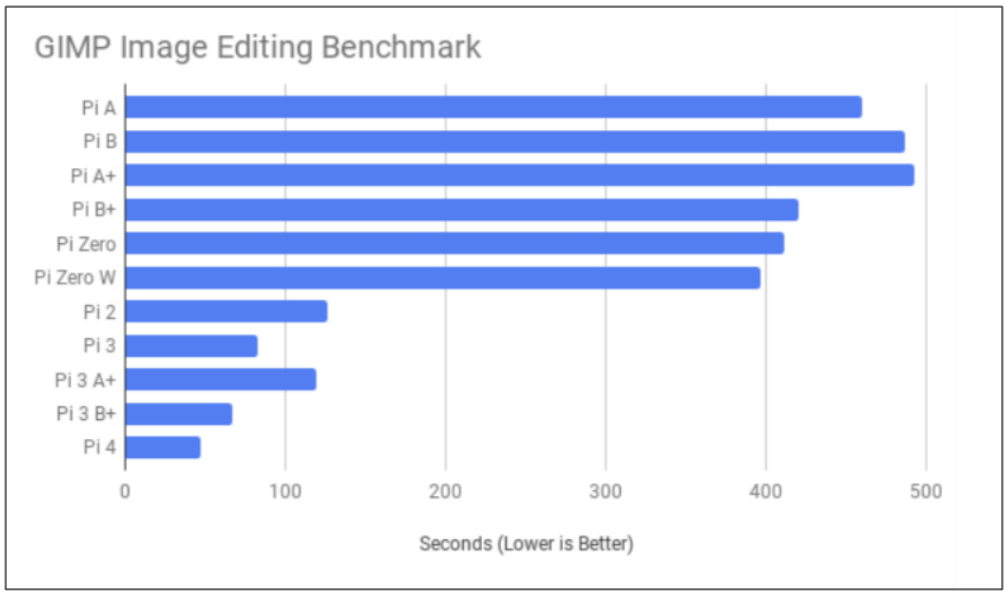


Figure 21: Model Comaprison: Image Editing

5.2.2 Setup

The Rpi uses the Raspbian operating system. The first step in the setup process was to install this operating system onto a microSD card. This involved using a specific adapter with a laptop and downloading the appropriate software from the Rpi website. Booting the Rpi up for the first time involved using an external keyboard and mouse, both connected through the available USB ports. The Rpi had to be connected to a power supply and a monitor. Basic settings such as region, password, WiFi needed to be configured following this. Much like a regular laptop, the Rpi environment contains a terminal to execute command lines. As mentioned previously, the format of these command lines mirror that of the Linux operating system. After

the initial setup, updates needed to be installed using the 'sudo apt-get upgrade' and 'sudo apt-get update' commands. Booting the system up each time using a keyboard and mouse would involve extra unneeded effort. There exists the option to remotely access the terminal of the Rpi using a laptop connected to the internet. This transfers the prior dependencies to the laptop's keypad and keyboard. Secure Shell Protocol (SSH) was used to achieve this. Windows 10's built in SSH Client and Server software was used to facilitate this operation. This required extracting the IP address of the Rpi and typing in the password chosen during the initial setup.

As with any new environment, the relevant programming packages and libraries need to be installed. To install OpenCV on Python the following commands were executed:

```
sudo apt-get install -y libhdf5-dev libhdf5-serial-dev
libatlas-base-dev libjasper-dev libqtgui4 libqt4-test
pip3 install opencv-python==4.0.1.24
```

When using OpenCV for a particular program it can be imported much the same as in the Jupyter Notebooks environment using: python3 -c import cv2.

The Rpi facilitates the use of an external camera module. Many different types exist, each varying in quality. The standard module was chosen. This epitomises the limitations of hardware used in this project and supports the idea that the limits of the hardware's capability will be tested. This is the root of the poor camera resolution challenge discussed in previous sections. The module is connected to the Rpi board using a ribbon cable and its use is facilitated through enabling the camera setting in the preferences tab (Rpi homescreen). This camera will be used to take in real-time images of the road. Its index is inputted as the single argument of the 'VideoCapture' function.

5.2.3 Software and Hardware Integration

The existing software must now be adapted for use on the Rpi. This is not as simple as just running the existing code in the Rpi terminal, some adjustments need to be made to the code and the manner in which it is run. The steps to do so are outlined below:

1. Update OpenCV to its latest version. This ensures that the highest performing version is being used, which could have potential implications on runtime.

2. The code itself, saved as an 'ipynb' file, must be transferred to the Rpi. This can be done via usual methods over the internet or by USB. The former was used in this project.

3. Any supplementary files needed to be transferred also. These included the 'coco.names' class labels text file, the network configuration file and the trained weights file of the model, the bank of images used for focal length calibration and all other images used for testing and evaluation.

4. The Python code could be implemented and run using the Rpi terminal. However, a text editor was used for this purpose. This allows for easier manipulation and debugging of the code as well as a generally more user-friendly interface. Many of these Rpi-specific editors exist each boasting slightly different features, layouts and even runtime variation. After thorough research, two text editors were chosen as potential platforms to run the code: Geany and Nano. Geany was subsequently chosen after side-by-side comparison.

5. Within the object detection code itself, some adjustments needed to be made. The file path names needed to be changed to the location within the Rpi Desktop. This is where the aforementioned files were transferred to.

6. The input argument of the 'VideoCapture' function was changed to the index (0) of the camera module attached to the Rpi board.

7. Focal length calibration needed to be reperformed on the Rpi to account for the different camera types used. This was done in the same manner as before, using the image bank transferred via the internet. All other parameter values were adjusted for this new environment as discussed in the Evaluation section.

8. In a similar way to the camera and SSH setup step, the General Purpose Input/Output (GPIO) functionality needed to be enabled (more on this in next section). This was done in the Preferences tab also.

5.2.4 Driver Alert

Having built software appropriate to the task at hand and successfully integrated it with suitable hardware, the next step involves deciding how the output of the algorithm should be conveyed to the driver. There exists many different paths leading to a sufficient solution to this problem. It is crucial to keep in mind the scope of this project. As embedding the object detection device into a larger system, such as an autonomous vehicle, does not form part of this project's scope, shining a light on the **versatility of the device and its potential to be added to other systems** is the main objective. Design and implementation choices would need to be made based on the specifics of the larger system and its associated environment of use. Some considerations for this task could include:

- Nature of connection to power supply.
- Available space.

- Existence of other similar devices.
- Detection objectives of specific vehicle.
- Desrable aesthetics.
- Visual vs audio driver alert.

For the purposes of this project, a basic alert distinguishing between cyclists and pedestrians is used. The whole objective is to aid driver observation and alert them of potential dangerous situations based on the locations of pedstrians/cyclists around them. Again, this leaves a lot of room for subjective choice. With relation to this project, the extent of danger is determined by the distance of objects from the vehicle. If an object is close enough to the vehicle, then it is deemed as a dangerous situation. Hence, the algorithm must discriminate between far and near objects.

There exists 40 General Purpose Input/Output (GPIO) pins on the Rpi model 4. These facilitate the use of external electrical circuitry, controlled through the Rpi terminal. It is easy to see that this opens up a whole new avenue for potential functionality of the device. The layout and assignment function for each pin is given in Figure 22.

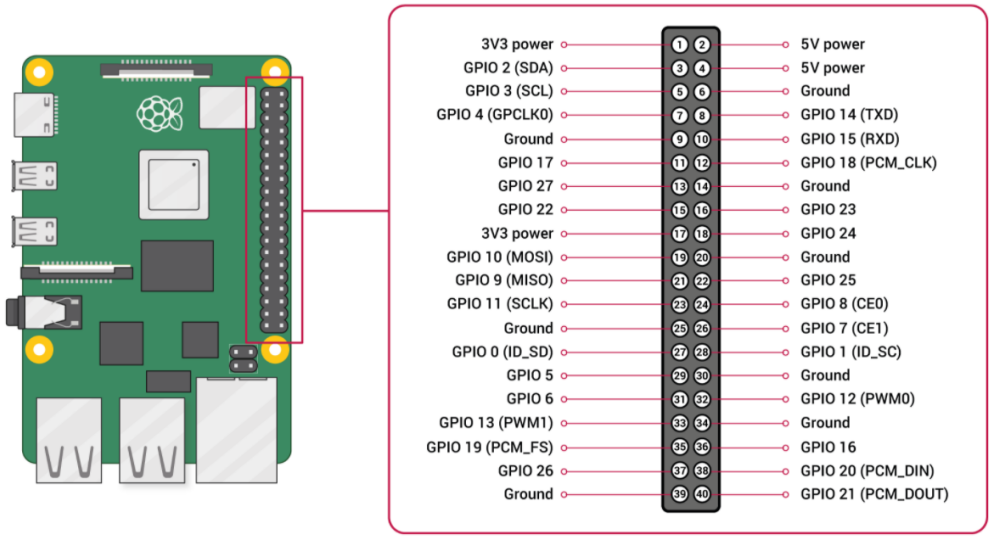


Figure 22: Raspberry Pi Model 4 GPIO Layout (Raspberrypi.org)

The two main candidates for driver alert signals were LEDs or buzzers i.e. visual vs audio. LEDs were chosen as they provide more versatility and clarity through the manner in which they signal different types of situations. Two separate LEDs were connected to the Rpi board as shown in Figure 23. The inside LED is connected to pin 8 through a 220 Ω resistor (anode) and a ground pin (cathode). The outer LED is also connected to the same ground pin (cathode) and pin 10 through a 220 Ω resistor (anode). Standard LED and resistors were used.

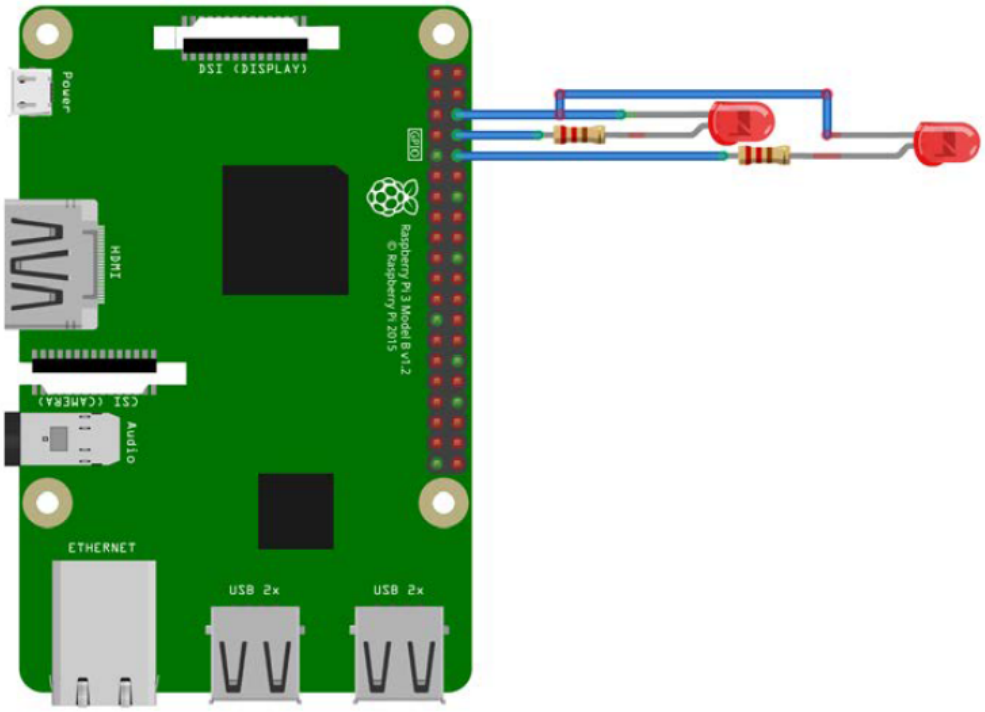


Figure 23: Two LEDS Connected to GPIO Pins

To access these GPIO pins a specific library must be imported into the Python code.

```
import RPi.GPIO as GPIO
from time import sleep
```

The first line of the code above allows a connection to be established to each GPIO pin. The second is an important feature used to dictate the timing of the LED signals. 'sleep()' produces a time delay of a given number of seconds. A few initial lines of code need to be run in order to ensure the appropriate setup of the connection.

```
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
GPIO.setup(8, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(10, GPIO.OUT, initial=GPIO.LOW)
```

The LED is checked to see if it works correctly. Subsequently, the 'setwarnings' function disables all warnings that may stop the flow of electricity to the pin. The 'setmode' function sets the pin addressing mode. To adhere to the layout shown in Figure 22, the mode GPIO.BOARD is chosen. 'setup' sets each appropriate pin as an output and tells the Rpi to send a LOW signal to each LED. This is the off state as opposed to HIGH which is the on state.

As mentioned before, the LED should signal a dangerous situation if either a cyclist or pedestrian is located close enough to the vehicle. A reasonable distance of approximately 5 meters is chosen. This is large enough

to account for the inaccuracies of the distance measurement. If an object is less than this distance away from the vehicle, an LED signal will be conveyed to the driver. The decision of how these signals should be arranged still needs to be made. A key feature of this alert should be to distinguish between cyclists and pedestrians. The code below shows how this is done for this project.

```
if dist < 200 and classIds[i] == 2:
    GPIO.output(8, GPIO.HIGH)
    GPIO.output(10, GPIO.HIGH)
    sleep(0.5)
    GPIO.output(8, GPIO.LOW)
    GPIO.output(10, GPIO.LOW)


else:
    GPIO.output(8, GPIO.HIGH)
    sleep(0.5)
    GPIO.output(8, GPIO.LOW)
```

This is why two LEDs were connected rather than one: Two lit LEDs suggests that there is at least one cyclist close enough and in the frame, one lit LED suggests that only pedestrians are close enough and in the frame. Therefore, if there is a cyclist close enough, regardless of how many or whether there are also pedestrian(s) close enough, two LEDs will light up. They will stay lit for 0.5 seconds after the cyclist has exited the dangerous threshold detection radius. If there are no cyclists close enough but a pedestrian, or multiple pedestrians, close enough then a single LED will light. The simplicity of this dichotomy is due to the rudimentary signal setup. One should remember that many choices of how to convey a dangerous situation to the driver can be made. The important thing is that **this device offers a platform for such widespread choice**, adding further potential functionality for the user. The method above simply demonstrates one basic way of doing so. Other possible ways explored include:

- Buzzers.
- Combination of both buzzer and LED.
- Screen showing pedestrians and cyclists in vicinity similar to rearview camera.
- Light up certain section of steering wheel corresponding to location of object relative to vehicle.
- More complex arrangement of LEDs, signalling more specific situations.
- Radar-style map on dashboard showing locations of objects relative to vehicle.

5.3 Evaluation

5.3.1 Distance Measurement Accuracy

The accuracy of the distance measurement needed to be evaluated. The simplicity of the triangle similarity method was bound to lead to errors due to oversimplification. Furthermore, the **extreme variability of real-world images of the road and the objects within them, adds significantly to the inaccuracies**. One redeeming factor is the absence of a need to very accurately measure distances due to a simple threshold value being used to determine dangerous situations. In other words, drivers do not need to know the exact distance a cyclist/pedestrian is away from the vehicle. He/she just needs to know whether they are reasonably close/within a certain radius of the car. To test the accuracy of this measurement, a bank of test images was accumulated. These images covered a variety of different settings and object arrangements. The actual distance of each object within these images was measured and stored. Each image was processed by the algorithm and the resulting distance measurements were compared to the actual distance measurements. This process is analogous to evaluating training vs test data for machine learning models. The **static pedestrian and cyclist width parameters were experimentally increased and decreased** to try and find a more accurate representative figure based on these results. Notably, the pedestrian width was increase to 20 inches. The results of this comparison can be seen in Figure 24.

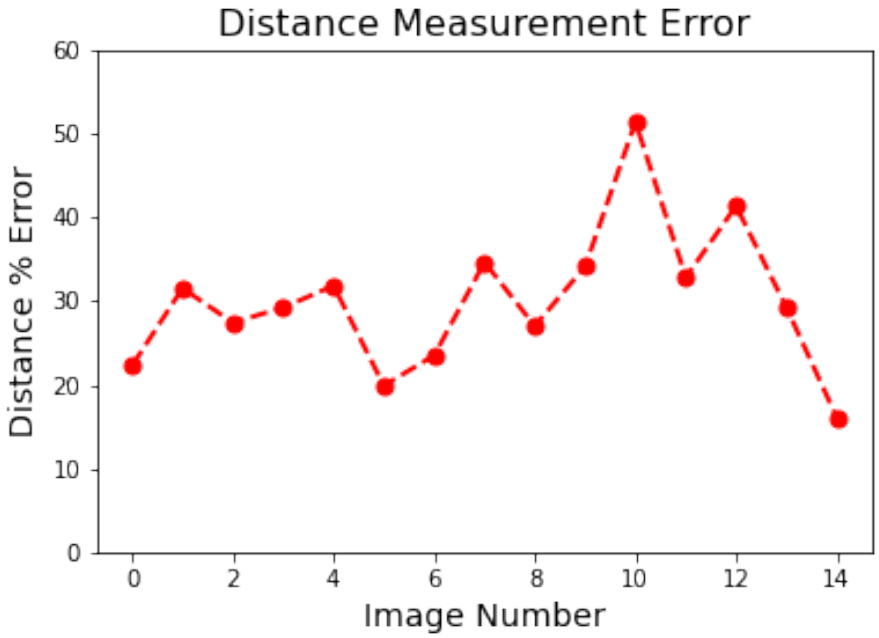


Figure 24: Distance Measurement Percentage Errors

The average error figure was 30.1%. This may seem quite large for an error figure but, as stated previously, an estimate of the correct distance only is required. The threshold for dangerous objects is 5m leaving a lot of room for larger values to be included as dangerous whilst not stopping very close objects from setting off the LED(s).

Stationary Webcam	Moving Vehicle	Still Image	Video	Occlusion
53.3%	49.8%	64.1%	60.9%	28.4%

Table 1: Mean Classification Accuracy Varying Environments

5.3.2 Performance Evaluation

The general performance of the algorithm needed to be tested and its parameters needed to be tweaked based on this assessment. The confidence level threshold value of 0.45 was deemed acceptable based on the examinations of previous sections. Three nms vales (0.1, 0.2, 0.3) were chosen and the resulting performances compared. 0.2 yielded the fewest incorrect multiple detections for a single object. Calculating the **mean classification accuracy** of the algorithm formed the basis of the evaluation. This is an important and informative figure that is often used to compare object detection models. It essentially quantifies how well the program detects particular objects in a given setting. Within the 'getObjects' function, two counters are used to compute this figure: 'counter2' counts the number of objects detected within a single frame and 'counter3' holds a running total of their confidence levels. These figures are return after each iteration of the while loop and an average is calculated.

```
for classId, confidence,box in zip(classIds.flatten(),confs.flatten(),bbox):
            ...
            counter2 += 1
            counter3 += confidence
...
while True:
        ...
        result,objectInfo,c1,c2 = getObjects(img
,0.45,0.2,KNOWN_PEDWIDTH,KNOWN_CYCWIDTH, focalLength, objects=[])
        counter1 += 1
        counter2 += c1
        counter3 += c2
        meanclass_accuracy = counter3/counter2
```

This figure was calculated for various situations as seen in Table 1 above. As there exists significant variation from one image to the next, a large number (30 in each case) of images were used to obtain an accurate figure. Equivalently, long video/webcam footage was used, having ensured there were plenty of pedestrians and cyclists in frame. The result for images containing occluded objects was understandably highly dependent on how occluded the object was. There were a few cases where no objects at all were detected. These were counted as 0% and therefore significantly reduced the overall average.

The figures for mean Average Precision (mAP) given by the makers of the model (A. Howard et al., 2019)

and (Sandler et al., 2018) were subject to more stringent evaluation, hence the discrepancy between them and the figures in this project. They are also calculated slightly differently to the mean classification accuracy above. Another important benchmark figure, **frames per second**, was also calculated. This gives a good indication of the speed of the program. The limited processing power of the Rpi was evident here. The device reached a maximum of 2 frames per second, meaning that although detections were reasonably accurate, they were slow. At this speed, the device barely qualifies as a real-time detector. Comparing this to the FAST YOLO model which can perform detection tasks at 155 frames per second, one can easily see the detrimental effects of using a smaller system.

The accuracy of 'bicycle' and 'cyclist' separation was also tested. As mentioned previously, it is not vital that the particular object is labelled correctly. The most important thing is that the device can detect whether there is an object close enough to the vehicle. A few false detections are acceptable. Using the same images and video footage from the mean classification accuracy calculations, a percentage error rate of 86% was calculated. This signifies how often the algorithm correctly identified a cyclist as a cyclist rather than both a bicycle and a pedestrian. An incorrect detection rate of 14% is definitely satisfactory when keeping in mind the scope of the project.

6 Results

In this project, the MobileNetV3 + SSDLite model has been successfully implemented on the Raspberry Pi to achieve basic pedestrian and cyclist recognition competency. The goals of this project have been met. The results from this object detection device are shown by the images below (further result images are provided in the Appendix). They are split into three sections: still images, video input and webcam input. It can be seen that in each case pedestrians and/or cyclists have been successfully located, classified and their distance measured. The algorithm accurately discriminates between bicycles and cyclists. One drawback is the speed with a frames per second rate of 2. This was, however, to be expected when using hardware with limited computational power.

Still Images

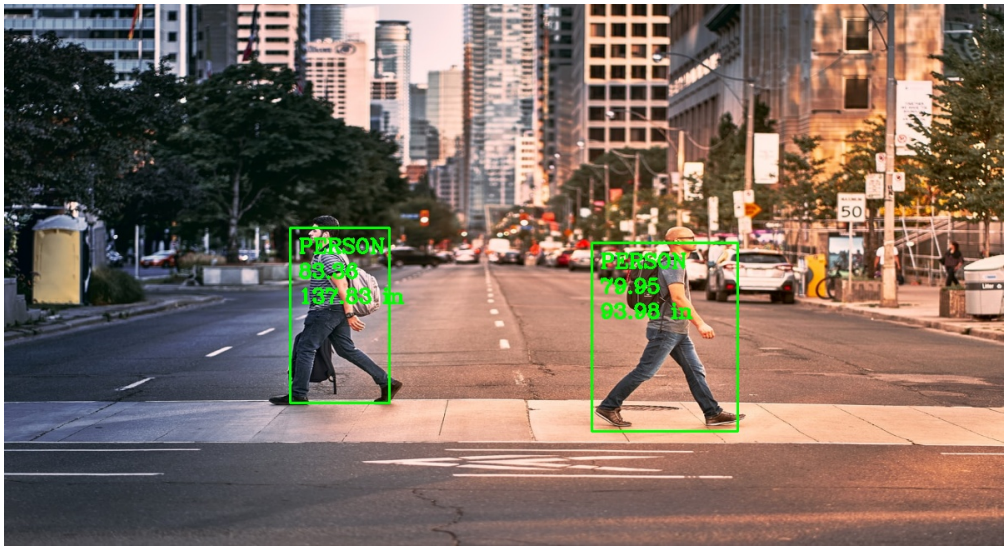


Figure 25: Still Image Results 1

Video Input

Figure 26: Video Input Results 1

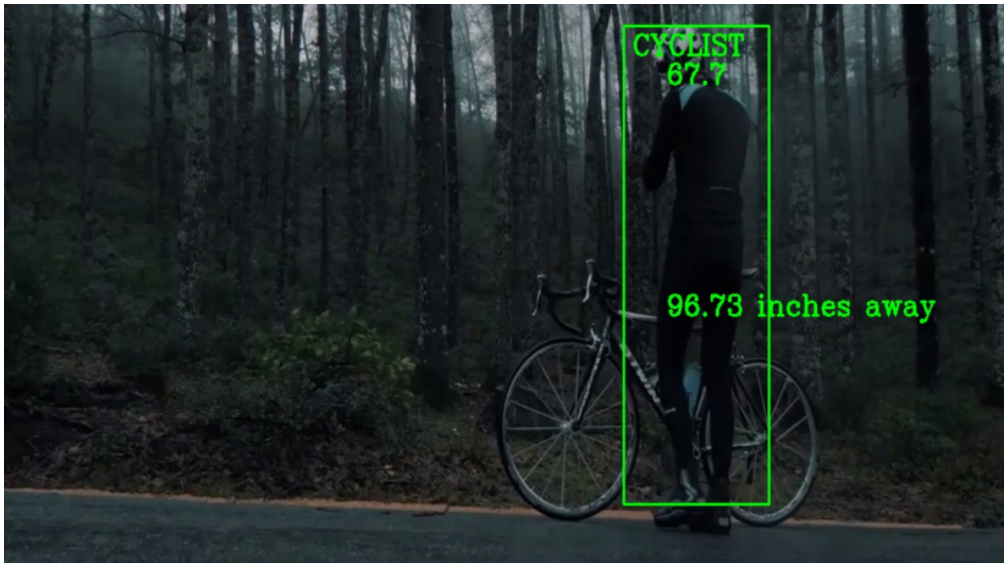


Figure 27: Video Input Results 2

Webcam Input

Figure 28: Webcam Input Results 1

7 Discussion

Throughout this project, it was always important to keep in mind the project aims and scope. It is easy to stray from these guidelines, given the enormity of the field of machine learning and object detection. Recognising that competing with commercial object detection systems was infeasible, an initial attainable

goal was set out: to build and program a pedestrian and cyclist detection device with useful functionality on computationally-restricted hardware. Compared to state-of-the-art devices with abundant computational resources, this device falls short as expected. Mean classification accuracy is approximately 55% whereas for practical, real-time systems it would ideally need to be significantly higher. Likewise, a speed of 2 frames per second would need to be drastically improved to cope with the variability and data-heavy tasks associated with real-time detection. Detection speed and precision lose out due to the lack of computational processing power. The achievement of the project lies in successfully creating an object detection device in spite of this.

7.1 Possible Future Work

This project provides a strong platform for others to potentially build upon in the future. With the advancement of hardware capacity (Moore's Law) and newly developed techniques in object detection, one could see dramatic increases in performance of similar devices in the future. Outlined below are a few ideas for possible future work relating to this project.

- The dataset for the MobileNetV3 + SSDLite model could be modified to include more/only images of pedestrians and cyclists. In doing this, the model will improve its detection capabilities having been given more (specific) initial data to be trained on.
- Variations and different combinations of models could be used instead of/alongside the aforementioned model. It has already been seen in this project that different algorithms can complement eachother to benefit overall performance. Additionally, advancements in theoretical understanding will undoubtedly aid all promising models in the future.
- This device could be coupled with other equipment such as sensors, trackers and other object detection devices to supplement its functionality. This would provide a more comprehensive detection setup than the one seen here.
- The processing power of the Rpi could be boosted by additional hardware such as the Coral USB Accelerator peripheral. This would be particularly helpful in speeding up the detection capabilities of the device.
- There exists a lot of potential to adapt the driver alert function. In this project, a basic signal was conveyed to the driver but, depending on the application, there exists many ways to perform this task and to determine which situations are, in fact, dangerous.
- One could expand the class range to other objects to supplement the recognition of dangerous situations on the road.
- The functionality and design of the device could be modified to suit a particular use or system. The versatility of the Rpi and coding languages, such as Python, facilitates this. The device built in this project provides a strong foundation for future manipulation and advancement.

7.2 Further Implications

Progress with regards to this project can have wide-reaching implications beyond the primary objective of improving pedestrian and cyclist safety. Theoretical and practical advancements in the field of computer vision can provide a relevant, positive impact in society with the recent proliferation in AI-related areas such as:

- **Autonomous vehicles:** Pedestrian and vehicle tracking, lane selection, path planning and pedestrian behaviour prediction are some of the uses of object recognition relating to autonomous driving (Lu, Tang, Wang, Zhu, & Wang, 2019).
- **Video Surveillance:** Object recognition is used to minimise human interaction in managing the surveillance process and to efficiently detect abnormal behaviour (Xu et al., 2018).
- **Robotics:** Creating human-like machines proves to be one of the most challenging applications of object detection. Even the simplest tasks require a high level of precision with the use of wide a range of object classes (Coates & Ng, 2010).
- **Medical Application:** Object recognition has huge potential to advance medical imaging and aiding the visually-impaired (Lee, Su, & Chen, 2012).

8 Conclusion

Following the discussions in the previous sections, a real-time object detection device has been successfully built. This device can detect pedestrians and cyclists on the road, measure their distances from a moving vehicle and alert the driver of a potentially dangerous situation. The originality of the project stems from working within the constraints of small, inexpensive hardware and from the unique combination of the resulting device's functionality. The intended application for this device is to reduce accidents relating to VRUs on the road. Given the device's capabilities, this is certainly a fitting application. The hope is to improve the statistics given in the introduction as well as overall road safety. The work completed in this project provides a strong means to do just that.

9 Appendix

Further pedestrian and cyclist detection results.

Still Images

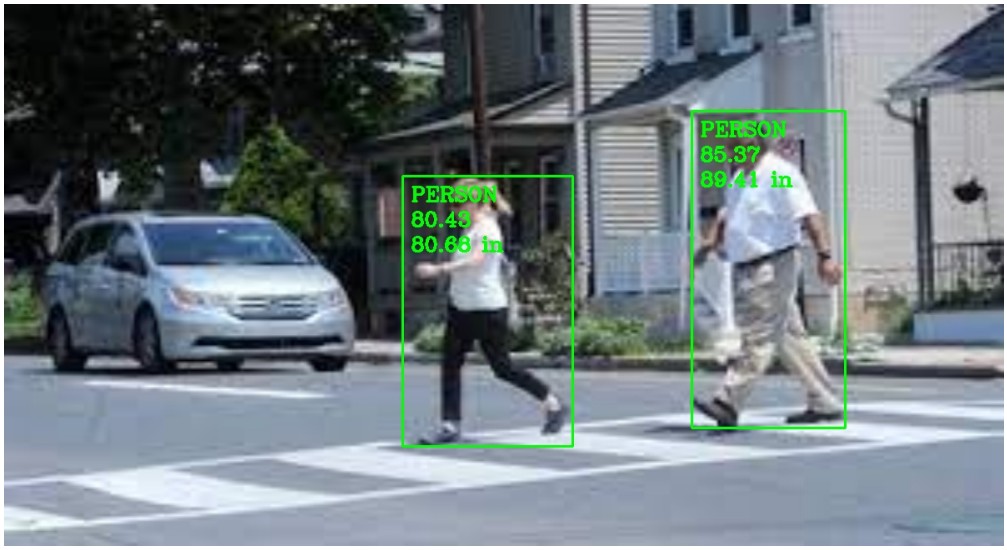


Figure 29: Still Image Results 2

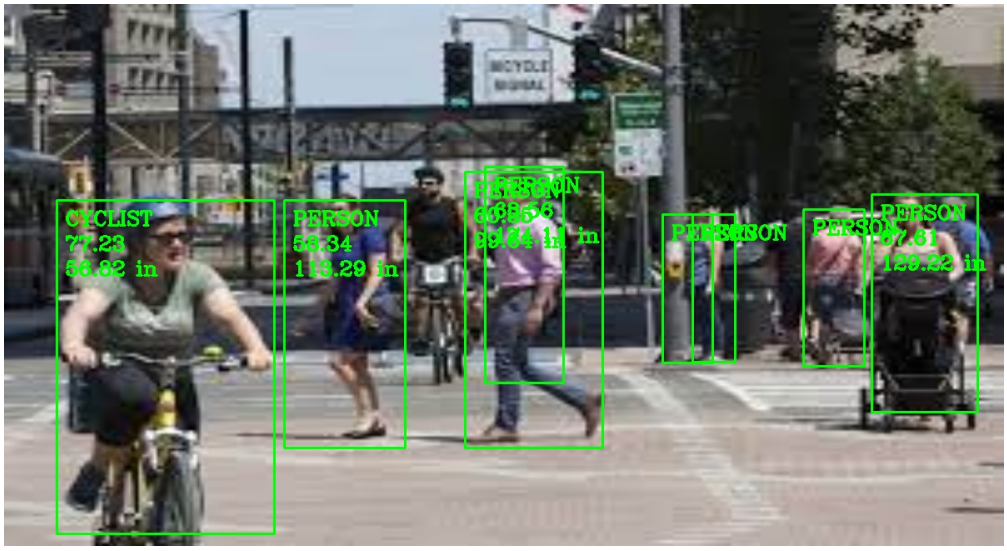


Figure 30: Still Image Results 3

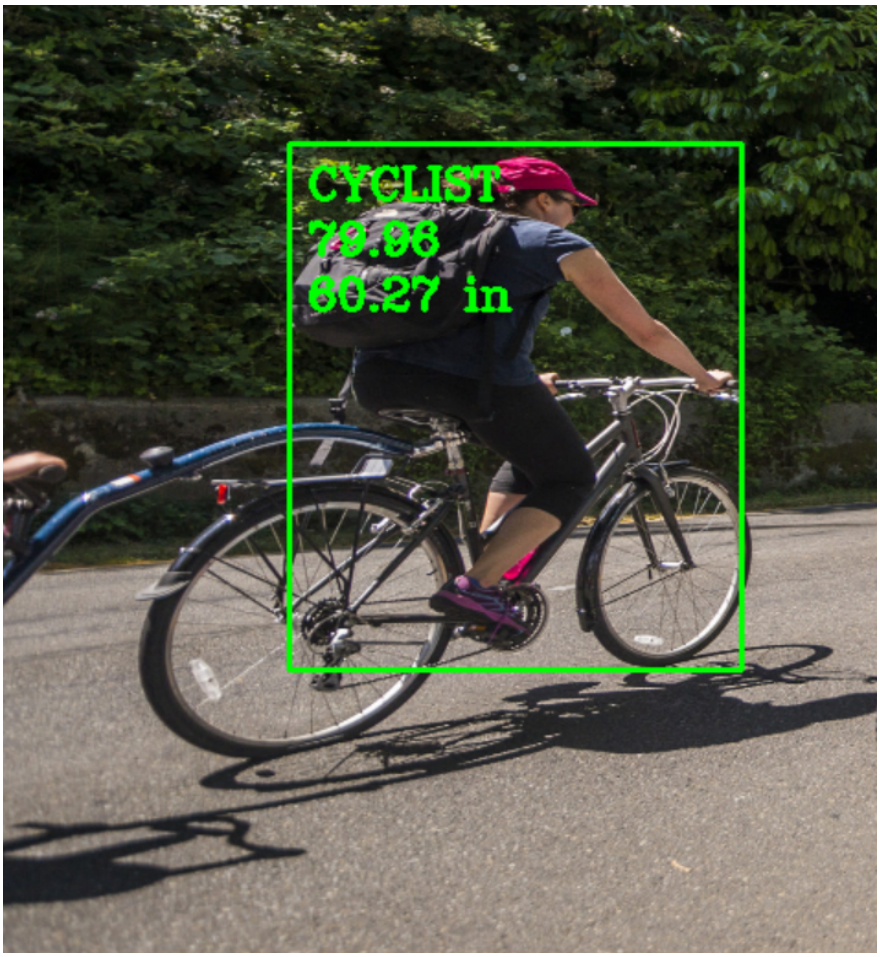


Figure 31: Still Image Results 4

Webcam Input

Figure 32: Webcam Input Results 2

Figure 33: Webcam Input Results 3

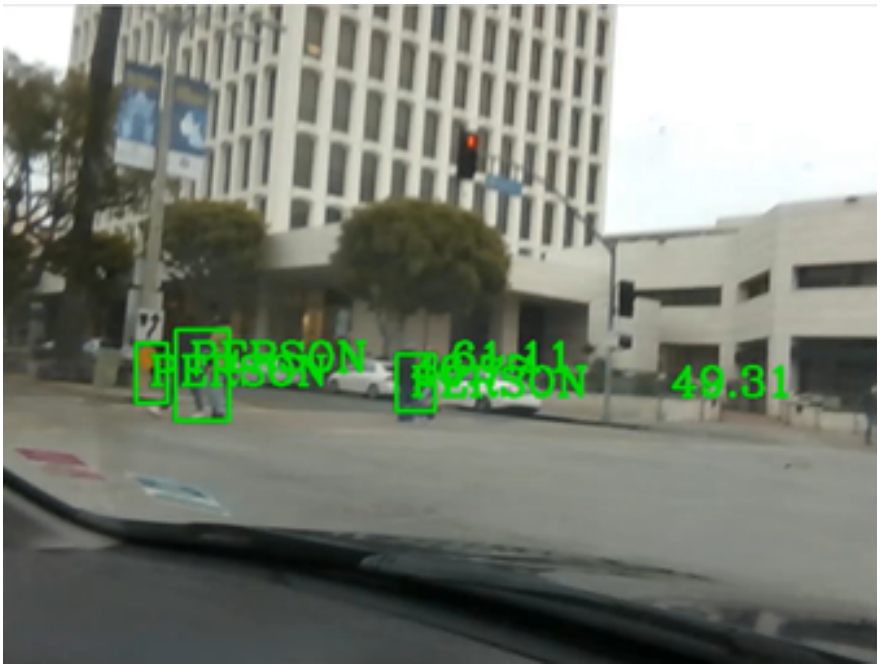


Figure 34: Webcam Input Results 4

References

Ahmed, S., Huda, M. N., Rajbhandari, S., Saha, C., Elshaw, M., & Kanarachos, S. (2019). Pedestrian and cyclist detection and intent estimation for autonomous vehicles: a survey. *Applied Sciences*, *9*(11), 2335.

Coates, A., & Ng, A. Y. (2010). Multi-camera object detection for robotics. In *2010 ieee international conference on robotics and automation* (pp. 412–419).

CSO. (2018). *Transport omnibus 2018.*

Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. In *2005 ieee computer society conference on computer vision and pattern recognition (cvpr'05)* (Vol. 1, pp. 886–893).

Dongarra, J. J., Luszczek, P., & Petitet, A. (2003). The linpack benchmark: past, present and future. *Concurrency and Computation: practice and experience*, *15*(9), 803–820.

Eurostat. (2018). *Road accidents: regions with highest fatality rates.*

Geronimo, D., Lopez, A. M., Sappa, A. D., & Graf, T. (2009). Survey of pedestrian detection for advanced driver assistance systems. *IEEE transactions on pattern analysis and machine intelligence*, *32*(7), 1239–1258.

Girshick, R. (2015). Fast r-cnn. In *Proceedings of the ieee international conference on computer vision* (pp. 1440–1448).

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 580–587).

Harris, C. G., Stephens, M., et al. (1988). A combined corner and edge detector. In *Alvey vision conference* (Vol. 15, pp. 10–5244).

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 770–778).

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., ... others (2019). Searching for mobilenetv3. In *Proceedings of the ieee/cvf international conference on computer vision* (pp. 1314–1324).

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

in Architecture, F. (n.d.). *Average male and female dimensions / heights.*

Khan, M. A., Paul, P., Rashid, M., Hossain, M., & Ahad, M. A. R. (2020). An ai-based visual aid with integrated reading assistant for the completely blind. *IEEE Transactions on Human-Machine Systems*, *50*(6), 507–517.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436–444.

Lee, C.-H., Su, Y.-C., & Chen, L.-G. (2012). An intelligent depth-based obstacle detection system for visually-impaired aid applications. In *2012 13th international workshop on image analysis for multimedia interactive services* (pp. 1–4).

Li, X., Li, L., Flohr, F., Wang, J., Xiong, H., Bernhard, M., ... Li, K. (2016). A unified framework for concurrent pedestrian and cyclist detection. *IEEE transactions on intelligent transportation systems*, *18*(2), 269–281.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). Ssd: Single shot multibox detector. In *European conference on computer vision* (pp. 21–37).

Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Proceedings of the seventh ieee international conference on computer vision* (Vol. 2, pp. 1150–1157).

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, *60*(2), 91–110.

Lu, J., Tang, S., Wang, J., Zhu, H., & Wang, Y. (2019). A review on object detection based on deep convolutional neural networks for autonomous driving. In *2019 chinese control and decision conference (ccdc)* (pp. 5301–5308).

Moravec, H. P. (1981). Review of deep learning algorithms for object detection. In *Ijcai* (Vol. 81, pp. 785–790).

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

Neubeck, A., & Van Gool, L. (2006). Efficient non-maximum suppression. In *18th international conference on pattern recognition (icpr'06)* (Vol. 3, pp. 850–855).

Phung, V. H., Rhee, E. J., et al. (2019). A high-accuracy model average ensemble of convolutional neural networks for classification of cloud image patches on small datasets. *Applied Sciences*, *9*(21), 4500.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 779–788).

Redmon, J., & Farhadi, A. (2017). Yolo9000: better, faster, stronger. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 7263–7271).

Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.

Reference.com. (2020). *How long is a bike?*

Ren, S., He, K., Girshick, R., & Sun, J. (2016). Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE transactions on pattern analysis and machine intelligence*, *39*(6), 1137–1149.

Rosebrock, A. (2015). *Bicycle parking info.*

Ross, A. (2002). *Find distance from camera to object/marker using python and opencv.*

RSA. (2018). *Cyclist injury trends 2006-2018.*

RSA. (2020). *Provisional review of fatal collisions.*

RSA. (2021). *Provisional review of fatal collisions.*

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 4510–4520).

Schmid, C., & Mohr, R. (1997). Local grayvalue invariants for image retrieval. *IEEE transactions on pattern analysis and machine intelligence*, *19*(5), 530–535.

Watson, K. (2018). *What's an average shoulder width?*

Williams, H. (2019). *What is artificial intelligence all about anyway?*

Xu, R., Nikouei, S. Y., Chen, Y., Polunchenko, A., Song, S., Deng, C., & Faughnan, T. R. (2018). Real-time human objects tracking for smart surveillance at the edge. In *2018 ieee international conference on communications (icc)* (pp. 1–6).

Zou, Z., Shi, Z., Guo, Y., & Ye, J. (2019). Object detection in 20 years: A survey. *arXiv preprint arXiv:1905.05055*.