Mapping suburban bicycle lanes using street scene images and deep learning

A minor thesis submitted in partial fulfilment of the requirements for the degree of
Master of Data Science

Tyler Saxton
School of Computing Technologies
Science, Engineering, and Technology Portfolio,
Royal Melbourne Institute of Technology
Melbourne, Victoria, Australia

October 26, 2021

Declaration

This thesis contains work that has not been submitted previously, in whole or in part, for any other academic award and is solely my original research, except where acknowledged.

This work has been carried out since March 2021, under the supervision of Dr. Dhirendra Singh.

Tyler Saxton
School of Computing Technologies
Royal Melbourne Institute of Technology
October 26, 2021

Acknowledgements

First and foremost, I would like to thank Dr. Dhirendra Singh for inspiring this research and supervising me throughout the year. I also greatly appreciate the input provided by Dr. Ron van Schyndel and the "Research Methods" class of Semester 1 2021, as I worked to develop a detailed research proposal.

To Dr. Sophie Bittinger, Dr. Logan Bittinger, Amy Pendergast, Laura Pritchard, thank you for your encouragement, and your assistance with the editing process.

Summary

Many policy makers around the world wish to encourage cycling, for health, environmental, and economic reasons. One significant way they can do this is by providing appropriate infrastructure, including formal on-road bicycle lanes. It is important for policy makers to have access to accurate information about the existing bicycle network, in order to plan and prioritise upgrades. Cyclists also benefit when good maps of the bicycle network are available to help them to plan their routes. This thesis presents an approach to constructing a map of all bicycle lanes within a survey area, based on computer analysis of street scene images sourced from Google Street View or "dash cam" footage.

Abstract

On-road bicycle lanes improve safety for cyclists, and encourage participation in cycling for active transport and recreation. With many local authorities responsible for portions of the infrastructure, official maps and datasets of bicycle lanes may be out-of-date and incomplete. Even "crowdsourced" databases may have significant gaps, especially outside popular metropolitan areas. This thesis presents a method to create a map of bicycle lanes in a survey area by taking sample street scene images from each road, and then applying a deep learning model that has been trained to recognise bicycle lane symbols. The list of coordinates where bicycle lane markings are detected is then correlated to geospatial data about the road network to record bicycle lane routes. The method was applied to successfully build a map for a survey area in the outer suburbs of Melbourne. It was able to identify bicycle lanes not previously recorded in the official State government dataset, OpenStreetMap, or the "biking" layer of Google Maps.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The benefits of "active transport", such as walking and cycling, have been well documented in previous studies. Participants' health may improve due to their increased physical activity. There are environmental benefits due to reduced emissions and pollution. And there are economic benefits, including a reduced burden on the health system, and reduced transportation costs for participants [1] [2].

Federal and State government policy makers in Australia therefore wish to encourage cycling [3] [4]. However, the share of cycling for trips to work in Melbourne is only 1.5% [5]. For many commuters, a perceived lack of safety of cycling is a major barrier to adoption. Other significant factors are the availability of shared bicycle schemes and storage facilities, and the risk of theft [6]. Cycling infrastructure has a significant impact on real and perceived cyclist safety, and this research project will focus on that issue. Important safety factors include the presence and width of a bicycle lane, the presence of on-street parking, downhill and uphill grades, and the quality of the road surface [7] [8]. A comprehensive dataset of cycling infrastructure would help policy makers prioritize areas in need of improvement to safety.

In Victoria, Australia, the State government publishes a "Principal Bicycle Network" dataset to assist with planning [9], however, during an exploration of the dataset, it was found to be significantly out of date. (See section 2.3.) Individual Local Government Areas may produce their own maps of bicycle routes, but availability is inconsistent [10].

The aim of this research project was to construct a dataset or map of bicycle lanes in a local area, by collecting street scene images at known coordinates, and then using a "deep learning" model to detect locations where bicycle lane markings are found. Detection locations were matched to an existing dataset of roads in the area, to infer bicycle lane routes along stretches

of road where markings were consistently found. If a baseline map of bicycle lane routes can be built this way, then the process could be extended in future to gather information about other significant factors, such as how frequently the bicycle lane is obstructed by parked vehicles, or the presence of debris or damage to the road surface.

Google Street View has been chosen as a source of street scene image data due to its wide geographical coverage, and the accessibility of the data via a public API. However, a significant limiting factor is that the Google Street View images for any given location might be several years out of date. Therefore, the use of images collected from a "dash cam" was also explored. A local government that is responsible for building and maintaining bicycle lanes could use dash cameras to gather its own images, at regular intervals, for more up-to-date data.

1.1 Research questions

- RQ1: Can a "deep learning" model be used to identify bicycle lane markings in street scene images sourced from Google Street View, in at least 80% of images where those markings appear, with a precision of at least 80%?
- RQ2: Can the model then be used to correctly detect and map all bicycle lane routes across all streets in a survey area with Google Street View coverage?
- RQ3: Can a similar process be applied to correctly detect and map bicycle lane routes from street scene images collected from dash camera video footage in a survey area?
- RQ4: Can the approach used to map bicycle lane routes from dash camera video footage be re-used to visually survey other details about the infrastructure?

Models from the TensorFlow 2 Model Garden were first trained to detect bicycle lane markings in a custom dataset of Google Street View images. These initial models generalised well enough to detect bicycle lane markings in dash camera footage, so they were used to gather additional images from the dash camera to include in the dataset for further training and validation.

The remainder of this thesis is laid out as follows: Chapter 2 is a literature review, providing motivations for the research, and a background in relevant techniques; Chapter 3 describes the approach that was taken; Chapter 4 reports the results and discusses the limitations and findings; Chapter 5 discusses the overall conclusions of the research. Please see Appendix A for instructions on how to access and use the code and data assets produced as part of this research project.

Chapter 2

Literature Review

2.1 Motivating literature

Prior research has clearly shown health, economic, and environmental benefits from active transport. Lee et al., 2012 [1] analysed World Health Organization survey data from 2008, and showed that physical inactivity significantly increased the relative risk of coronary heart disease, type 2 diabetes, breast cancer, colon cancer, and all-cause mortality, across dozens of countries. Rabl & de Nazelle, 2012 [2] demonstrated that active transport by walking or cycling improves those relative risks for participants. Moderate to vigorous cycling activity for 5 hours a week reduced the all-cause mortality relative risk by more than 30%. They estimated an economic gain from improved participant health and reduced pollution, offset slightly by the cost of cycling accidents.

In Australia, Federal and State governments are committed to the principle of supporting active transport through the provision of cycling infrastructure, declaring their commitment through public statements on their official websites [3] [4]. Many other governments around the world have adopted similar policies.

Taylor & Thompson, 2019 [5] surveyed the use of active transport in Melbourne, to establish a baseline of current commuter behaviour. They found that cycling only accounted for 1.5% of trips to work in the area. It could therefore be argued that there is room for improvement.

Schepers et al., 2015 [11] produced a summary of literature related to cycling infrastructure and how it can encourage active transport, resulting in the aforementioned benefits. The paper found that providing cycling infrastructure that is perceived as being safer does encourage participation. Other papers such as Wilson et al., 2018 [6] agreed.

Other researchers have examined which factors affect the perceived and actual safety of cycling routes, in a variety of settings.

Klobucar & Fricker, 2007 [7] surveyed a group of cyclists in Indiana, USA, asking them to ride a particular route and rate the safety of each road segment along the route, then asking them to review video footage of other routes and rate the safety of those routes, too. A regression model was created to predict the cyclists' likely safety ratings for other routes. The creation of the model led to a list of road segment characteristics that were apparently most influential in the area.

Tescheke et al., 2012 [8] surveyed patients who attended hospital emergency rooms in Toronto and Vancouver in Canada, due to their involvement in a cycling accident. Details of the circumstances of each accident were gathered, along with the outcomes. The dataset was analysed to determine which factors increased (or decreased) the relative odds of a cyclist being involved in an accident.

Malik et al., 2021 [12] modelled cyclist safety in Tyne and Wear County in north-east England, a more rural setting.

The factors that contribute to cyclist safety vary by locality. For example, cyclists in one city might be concerned by the hazard of tram tracks, whereas this might not be a relevant concern in another city, or a less built-up area. The common themes among the aforementioned papers were: the presence and width of a bicycle lane; the presence of on-street parking; downhill or uphill slopes; the volume, speed and profile of motor vehicle traffic; the quality of the road surface; lighting; and the presence of construction work.

Most of these factors can be influenced by infrastructure and road design. Therefore, it would be valuable to quantify as many of these factors as possible in a dataset, to assist policy makers in deciding what changes ought to be made, and where, to provide a safer network of cycling infrastructure.

2.2 Relevant literature in machine learning

"Deep Learning" is a paradigm by which computational models can be constructed to tackle many problems, including visual object recognition. A Deep Learning model can be trained to perform visual object recognition tasks by supplying it with a "training" dataset of images where the objects it must recognise have been pre-labelled. During training, the model processes its training dataset over and over again, and with each iteration the weights in its

multi-layer neural network are refined through the use of a "backpropagation" algorithm [13]. With a sufficient number of training "epochs", the model hopefully develops the ability to detect if and where the objects of interest appear in each image, to an acceptable level of performance. See figure 2.1.

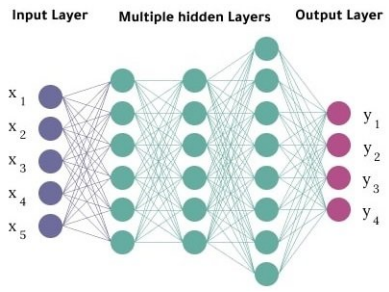


Figure 2.1: Example deep learning model, showing the multiple hidden layers that distinguish it from a "non-deep" neural network. Each arrow in the diagram represents a connection with a "weight". Weights are set and refined during the training process, via a "backpropagation" algorithm. Source: Lofqvist et al., 2020 [14], reproduced under Creative Commons licence.

Tensorflow [15] [16] and PyTorch [17] are two dominant frameworks for building, training, evaluating, and applying Deep Learning models. Within these frameworks, researchers have progressively developed new models for visual object recognition, typically with a focus on either improving the accuracy of the results, increasing speed to allow "real time" processing of video streams, or creating models that will work on low-cost hardware. Significant Deep Learning models considered within this research project include: R-CNN, proposed by Ren et al., 2015 [18]; SSD, proposed by Liu et al., 2016 [19]; ResNet, proposed by He et al., 2016 [20]; YOLO and subsequent variants, first proposed by Redmon et al., 2016 [21]; MobileNetV2, proposed by Sandler et al., 2018 [22]; and CenterNet, proposed by Duan et al., 2019 [23].

The Tensorflow 2 Model Garden [24] provides access to many of these models, pre-trained on the COCO 17 ("Microsoft COCO: Common Objects in Context") dataset. It therefore provides a convenient library of Deep Learning models that have already received a significant amount of training for the general problem of visual object recognition, and can be re-used to recognise new classes of objects through a process of "transfer learning" [25] [26].

Semantic image segmentation of street scene images is an important and active area of computer science research. "Deep Learning" models that can understand sequences of images in real time are essential for self-driving vehicles or similar driver-assistance systems, so many papers have focussed on that area. In order to train a Deep Learning model that specializes in understanding street scene images, a labelled dataset of street scene images is required.

Papers by Cordts et al., 2016 [27] and Zhou et al., 2019 [28] announced the publication of the "Cityscapes" and "ade20k" image datasets, respectively. These datasets each contain many street scene images, with objects of interest labelled in a format suitable for training deep learning models to understand similar on-road scenarios. Many papers have used these datasets in order to train new machine learning models. Chen et al., 2018 [29] is one example of a heavily-cited paper where "CityScapes" data has been used to train a "real time" model to understand street scene images. Unfortunately, neither of the datasets has cycling infrastructure labelled. The "CityScapes" dataset has labelled bicycle lanes under the "sidewalks" category. This is useful to train a car not to drive there, but a cyclist might not be legally allowed to ride on a sidewalk designed for pedestrian traffic.

In another branch of research, deep learning tools have been used to manage roadside infrastructure and assets. Campbell et al., 2019 [30] used an "SSD MobileNet" model to detect "Stop" and "Give Way" signs on the side of the road in Google Street View images, to help build a database of road sign assets. An application called "RectLabel" was used to label 500 sample images for each type of sign. Photogrammetry was used to estimate a location for each detected sign, based on the Google Street View camera's position and optical characteristics and the bounding box of the detected sign within the image. Zhang et al., 2018 [31] performed a similar exercise, detecting road-side utility poles using a "ResNet" model.

In Australia, the "Supplement to Australian Standard AS 1742.9:2000" sets out the official standards for how bicycle lanes must be constructed and marked. Generally, a lane marking depicting a bicycle should be painted on the road inside the bicycle lane within 15 metres before and after each intersection, and at intervals of up to 200 metres[32]. Therefore, it is proposed that a Deep Learning model could be trained to detect these markings, similar to how Campbell et al., 2019 [30] trained a model to detect road signs, and Zhang et al., 2018 [31] trained a model to detect utility poles, using pre-trained models from the TensorFlow 2 Model Garden [24] as a starting point.

The use of satellite imagery was also considered. Li et al., 2016 [33] showed that a road network could be extracted from satellite imagery using a convolutional neural network (CNN), and this is particularly useful in rural areas where maps are not already available. However the resolution of publicly available satellite image data would not be sufficient to identify a bicycle lane on a road, and it would not be able to distinguish a standard bicycle lane from a paved shoulder. Satellite imagery may have a role to play in detecting off-street bicycle tracks in "green" parkland area, but it was decided to exclude off-street routes from the scope of this research, and focus on on-road infrastructure.

Aerial photography may be useful, where it is available with sufficient detail. However it would require a different data source for every jurisdiction. Ning et al., 2021 [34] had success extracting sidewalks from local aerial photography using a "YOLACT" model. Areas of uncertainty caused by tree cover were filled in using Google Street View images. The model appeared to rely on a concrete sidewalk having a very different colour to the adjacent bitumen road. The approach might not be able to distinguish bitumen bicycle lanes.

Aside from formal bicycle lanes, cyclist safety can also be improved by a paved shoulder, or by creating lanes that are wide enough to safely accommodate a vehicle passing a cyclist. It may be possible to detect and map these arrangements by recognising lane markings and road boundaries. A common method of detecting lane boundaries is through the combination of a Canny edge detector [35] and a Hough transformation [36]. The OpenCV library provides a frequently used implementation [37]. If the Canny-Hough approach struggles with a poorly defined road boundary or "noise" from roadside objects, a Deep Learning approach may help: Mamidala et al., 2019 [38] successfully used a "CNN" model to detect the outer boundary of roads in Google Street View images.

During the literature review, one other paper was identified where the authors had applied Deep Learning techniques in the domain of cyclist safety: Rita, 2020 [39] used the "MS Coco" and "CityScapes" datasets to train "YOLOv5" and "PSPNet101" models to identify various classes of object (Bicycle, Car, Truck, Fire Hydrant, etc.) in Google Street View images of London. A matrix of correlations between objects was calculated. This was used to infer the circumstances where cyclists might feel the most safe. For example, there was a high correlation between "Person" and "Bicycle" which "suggests pedestrians and cyclists feel safe occupying the same space". This is a research did not involve creating maps.

2.3 Cycling infrastructure datasets and standards

The Victorian State Government in Australia started publishing an official "Principal Bicycle Network" dataset on `data.gov.au` in 2020 [9]. It is an official dataset that is intended to help policy makers with their planning. It includes "existing" and "planned" bicycle routes. The dataset only covers formal bicycle lanes, so it generally excludes roads with a simple paved shoulder. In some country areas, the routes did not appear to be formally marked as standard bicycle lanes. It was found that some existing bicycle lanes are still marked as "planned" in the dataset, or not listed at all. There is a field to record when each entry was last validated, but the most recent timestamp was in 2014. The data appears to be incomplete and out of

date. Its scope is limited to the State of Victoria.

"OpenStreetMap" [40] is a source of crowdsourced map data. It provides detailed information about road networks worldwide, and the attributes of each road segment. "Cycleway" tags can be used to mark not just where a bicycle lane is, but other interesting attributes, such as whether the cycleway is shared with public transport, whether there is a specially marked area for cyclists to stop at each intersection, and how wide the bicycle lane is. Given that the data is crowdsourced, the quality and availability of the data may vary by location.

"Google Maps" provides a "biking layer" [41]. This includes off-street bicycle paths and on-street bicycle lanes. It only gives a "yes" or "no" opinion about whether a route is especially suited to bicycles, with no further information provided. But that may be of assistance in scouting locations to use in a dataset of Google Street View images.

Chapter 3

Methods

This section describes the methodology that was used to address each of the research questions. To assist with reproducibility, all code has been published on GitHub and FigShare, and the dash camera footage and data has been published on FigShare. For detailed instructions on how to access the code and data, and how to run the code via the provided Jupyter Notebooks, please refer to appendix A.

3.1 RQ1: Training a model to identify bicycle lanes in Google Street View images

A review of the relevant Australian standards for bicycle lanes [32] suggested that the best way to search for bicycle lanes in street scene images would be to focus on the bicycle lane markings that are painted on the road surface. See figure 3.1 for an example marking.

Figure 3.1: Example bicycle lane marking. Source: Google Street View, Oct 2019

This marking is universal across all standard bicycle lanes in Australia, and can distinguish a bicycle lane from other parts of the road. It may sometimes appear on a green surface, and occasionally it may be partially occluded due to limited space or wear and tear. Similar markings exist in many other jurisdictions outside Australia.

In order to train a deep learning model to recognise bicycle lane markings in street scene images, it was necessary to first create a dataset of labelled images for training and validation. This could then be used to apply a variety of pre-trained models from the "TensorFlow 2 Model Garden" to the problem, through a process of transfer learning.

3.1.1 Creating a labelled dataset of Google Street View images

The official Australian design standards for bicycle lanes specify that bicycle lane markings should appear within 15m of each intersection [32]. Therefore, intersections along known bicycle lane routes are locations where example images are likely to be found.

3.1.1.1 Identifying candidate intersections to sample

Two possible sources of information about "known" bicycle lane routes were considered for use in the sampling process: The "Principal Bicycle Dataset", and XML extracts from the OpenStreetMap database. These datasets are discussed in section 2.3, with further details of the structure of the OpenStreetMap data provided in Appendix B. The "Principal Bicycle Network" dataset was chosen, as the incumbent official dataset for local policy makers. Due to the age of the data, it was also less likely to suggest bicycle routes not yet captured in the Google Street View images. Using OpenStreetMap as a source of "known" bicycle lane routes would generalise to other jurisdictions, therefore tools were created to support both approaches, and their operation is described in appendix A.1.1 to A.1.3. Given a list of "known" bicycle lane routes from either of these sources, each route can be looked up in an OpenStreetMap database to identify its intersections.

OpenStreetMap data can be downloaded for an individual country via the "GeoFabrik" website [42] and then sliced into a smaller area, if necessary, using the "osmium" command line tool [43]. For more background on concepts in the OpenStreetMap data, please refer to Appendix B. Briefly, an OpenStreetMap XML extract file contains "ways", which represent lines on a map, such as road segments, and "nodes", which represent points with latitude/longitude coordinates that make up the "ways" and thereby describe the shape of each line on a map. A list of candidate intersections along "known" bicycle lane routes can be extracted from OpenStreetMap XML data by looking for "nodes" that are common to two or more "ways" that are roads and have distinct names.

Generating a list of intersections from the "Principal Bicycle Network" dataset involves additional work, because it only lists bicycle lane routes, and does not have any information

about intersecting roads. Each point in the dataset is "reverse-geocoded" to find the name of the road using an API call to a local instance of a "Nominatim" server, as described in appendix C.2. An OpenStreetMap XML extract for Victoria is then searched for all "ways" with a matching name, to find the corresponding road segments in the OpenStreetMap data. The bounding box around each "way" is compared to the bounding box of the "Principal Bicycle Network" route, to ensure they roughly overlap, and remove matches for other roads with the same name, in a completely different area. Once matching "ways" have been found for the route, they can be checked for intersections as per the OpenStreetMap process.

For every intersection identified, it is useful to calculate a "heading" for the road at the intersection, to help gather Google Street View images that are aligned to the direction of the road. The Python "geographiclib" library can calculate a heading from one point to the next. It is assumed that the heading at each intersection is the average of the heading from the previous "node" and the heading to the next "node".

3.1.1.2 Downloading sample Google Street View images for the dataset

Google provides an API for downloading Google Street View images of up to 640x640 pixels, at a cost of $7.00 USD per 1,000 images. Volume discounts are available, with a reduced cost of $5.60 USD per 1,000 image for volumes from 100,001 to 500,000 images. Beyond that, further volume discounts can be negotiated with Google's sales team [44].

Google Street View images were cached locally, to avoid unnecessary costs when requesting an image that has previously been downloaded.

The Google Street View API allows images to be requested for a location (latitude/longitude) with a desired heading, field-of-view angle, and camera angle relative to the "horizon". A Jupyter Notebook with an "ipywidgets" GUI was created to: Randomly select a sample location from the list of candidate intersections in Victoria; Download four Google Street View images at 0, 90, 180, and 270 degrees relative to the heading of the road, with a 90 degree field of view, and the camera angled downwards 20 degrees towards the ground to focus on any nearby road markings; Allow the operator to press buttons to record which images should be included in the dataset due to the presence of an interesting feature. Please see appendix A.1.4 for screenshots and detailed instructions on how to use the tool.

Bicycle lane markings are not always visible from the middle of an intersection, therefore, options were provided to allow the operator to move the camera up and down the road, 10 metres at a time, to get a better view of a marking that is just visible in the distance.

If the operator could not clearly tell that a white marking in an image was a bicycle lane marking, then the image was not included in the dataset.

The GUI allowed multiple candidate intersections to be assessed per minute. An initial set of 256 images was collected over the course of approximately 4 hours, based on a random sample of points across the State of Victoria.

3.1.1.3 Labelling the dataset

Once a suitable number of Google Street View images containing bicycle lane markings had been collected, they were copied to a single folder and then labelled using the open-source tool "labelImg" [45]. Please refer to appendix A.1.5 for more information on how to perform the labelling process. The "labelImg" tool allows the operator to examine each image in a folder, draw bounding boxes around objects of interest in each image, and assign a label to each bounding box to categorise its contents. The output of the process was one XML file per image, in a format that could be understood by TensorFlow 2 training tools and scripts.

3.1.2 Training and evaluating candidate models from the TensorFlow 2 Model Garden

Training and evaluation of candidate models was conducted on local infrastructure, as described in Appendix C.1. TensorFlow was configured for GPU-accelerated development, using CUDA and cuDNN libraries from NVIDIA.

The dataset that was collected in section 3.1.1.2 and labelled in section 3.1.1.3 was randomly split into "training" and "test" datasets according to an 80:20 ratio. There were 205 images in the "training" dataset, and 51 images in the "test" dataset.

The Jupyter Notebook described in appendix A.1.6 was used to download a candidate pre-trained model from the TensorFlow 2 Object Garden and configure it to search for bicycle lane markings based on the collected dataset. The Jupyter Notebook printed out commands that could be used from the command line, to train the model for a specified number of training steps against the "training" dataset, evaluate its performance against the "test" dataset, and create a "frozen" copy of the model as-at the current level of training.

Three significant factors were found to influence the performance of the model: The size of the dataset available for "training" and "test" purposes; the number of "epochs" or "training steps" spent training the model on the data; and the "confidence score " threshold, from 0%

to 100%, at which the model declares that a bicycle lane marking has been detected.

The initial size of the dataset was set based on previous experience in other papers such as Campbell et al., 2019 [30], where models had been successfully trained with around 500 images. An initial dataset of 256 images was gathered, and further images were added later in response to model performance.

During the training process, the TensorFlow 2 training script reported "loss" performance metrics every 100 training steps. This performance metric typically gradually improved as more training was conducted over the dataset, until a point of diminishing returns was reached. Every 2000 training steps, the training was paused, and an evaluation script was run to test the performance of the model against the independent "test" dataset that had been withheld from the training process. When training reached the point that further training steps did not significantly improve performance, training was halted.

The "confidence" threshold was adjusted manually, by reviewing the false positives and false negatives produced by the model, and adjusting it to provide a balance between over-confidence and conservatism. As part of the evaluation process, copies of each image were written to disk with an overlay showing any detected bounding box and its confidence score. These outputs were reviewed as part of the threshold tuning process. The boundary boxes of detections were also checked to ensure that the results were sensible.

Multiple pre-trained models from the TensorFlow 2 Model Garden were trialled, with selections guided by their benchmark "mAP" (mean Average Precision) metrics for processing of the "COCO 17" dataset. The process of building a map of bicycle lanes for an area can be considered a "batch" process, with no fixed time constraints, therefore mean Average Precision was prioritized over benchmark "speed".

The TensorFlow 2 Model Garden provided an efficient way to test multiple models via a single framework. There are popular object detection models, such as the "YOLO" series of models, that are not supported by it. For the purposes of this research, it was not considered necessary to perform an exhaustive search of all possible models across multiple frameworks. A different model (e.g. YOLOv3) or a different framework (e.g. PyTorch) could be substituted if required.

Please refer to Appendix A.1.6 for instructions on how to run the training process.

3.2 RQ2: Building a map of bicycle lane routes from Google Street View images in a survey area

In section 3.1, a model was trained to detect bicycle lane markings in one street view image at a time. To generate a map or dataset of bicycle lanes in an area, the first step is to determine a list of locations for which Google Street View images should be downloaded and processed by the model. A batch process can then used to process each image and record whether or not a bicycle lane marking was found. Finally, the list of sample locations where there was a "hit" must be correlated to geospatial data about the road network to infer routes, draw them on a map, and compare them to other sources.

3.2.1 Sampling strategy for Google Street View images

In section 3.1.1.2, it was noted that Google charges up to $7.00 USD per 1,000 Google Street View images, and that the most likely place to find a bicycle lane marker is in the area immediately surrounding an intersection. To reduce costs, a strategy of sampling only intersections and their immediate vicinity was adopted. Samples could also be taken at regular intervals along each road, if initial results suggested that this was necessary to compensate for missed detections.

The Jupyter Notebook described in appendix A.1.8 was used to load an OpenStreetMap XML extract file for the survey area into memory, and then traverse it to produce a batch file with the coordinates of every point within 20 metres of an intersection, at intervals of 10 metres. The length of the batch file was inspected, to assess potential costs, and then the required images were downloaded via the Google Street View API, if they were not found in the local cache.

It was discovered that some intersections would be missing on roads at the very margins of the survey area, if the intersecting road existed entirely outside the bounds of the OpenStreetMap XML extract for the survey area. A second extract, capturing bounding box with an extra margin of 200 metres around the survey area, was used to rectify the missing intersection issue. The 200 metres margin was found to be sufficient to ensure that the adjoining road was included in the extract by the osmium tool. Roads in the second extract were not considered part of the survey, but data about them was taken into account to ensure that intersections at the margin of the survey area were not missed.

Please see appendix A.1.9 and A.1.10 for detailed instructions on how to operate the tools

that were used to generate a list of sample locations in a survey area.

3.2.2 Applying a detection model to a batch of Google Street View images

The Jupyter Notebook described in appendix A.1.10 was also used to apply the detection model to the sampled Google Street View images, and produce an output CSV with the coordinates of every point where a bicycle lane marking was detected. If one or more bicycle lane markings were detected in an image, a copy of the image was saved to a folder of "hits" with a bounding box and confidence score overlay. An example image with detection overlay is shown in figure 3.3. If no bicycle lane marking was found, a copy of the image was instead placed into a "miss" folder. Partitioning the result images into "hits" and "misses" allows the operator to quickly browse the images to check for any false positives or false negatives.

Figure 3.2: Example bicycle lane marking with detection overlay. Source: Google Street View, Oct 2019

3.2.3 Inferring bicycle lane routes from Google Street View detection points

The result of the batch process in section 3.2.2 is a list of locations where a bicycle lane marking was found, linked to the OpenStreetMap "way" IDs and "node" IDs that they were sampled from. The next step is to use those detection points to infer contiguous routes that can be drawn as lines on a map.

Four Google Street View images were sampled at each sample point in the survey, providing a 360 degree view. An approximate heading was calculated for each road at each intersection, but occasionally that did not appear to match the retrieved images. Therefore, if a bicycle lane marking was detected at an intersection, it could be difficult to accurately decide which road it belonged to.

The proposed solution was to infer routes from the detection points based on the assumption that if markings were detected at two or more consecutive intersections along a named road, there is a bicycle lane along that segment. A configuration option was provided to allow for one or more intersections with "missing" detections before assuming that the bicycle lane route has been interrupted.

If a street shared an intersection with a road that had a bicycle lane, it would not usually be reported as having a bicycle lane unless it had another intersection with a bicycle lane at the next intersection.

A continuous road may be represented in OpenStreetMap as multiple "way" segments if any of its characteristics change from one segment to the next, such as a change of speed limit. Before inferring bicycle lane routes along a road from the detection points, it was necessary to link adjacent "way" segments from a single named road back up into a single chain. Otherwise, the option to allow for intersections with "missing" detections would not always work as intended.

Each inferred route was drawn as a line in a geojson file, which can be drawn on a map or measured as per section 3.2.4 below.

This approach was intended to draw a map of bicycle lane routes irrespective of which side of the road the bicycle lane is on. If additional information is required about roads where bicycle lanes are only present on one side of the road, the process would need to be refined to take into account the heading of the road, the heading of the camera angle, and exactly where in the frame the bicycle lane marking was detected.

3.2.4 Comparing results to other data sources

Once a geojson file has been produced to describe the detected bicycle lane routes, it can be drawn on a map in a Jupyter Notebook using the Python "ipyleaflet" library. See appendix A.1.11.

The OpenStreetMap XML extract for the survey area can be filtered down to "ways" that are roads with the "cycleway" tag, and then converted into an equivalent geojson file. The detected routes and the routes that are tagged in OpenStreetMap can be viewed side-by-side on a map.

To allow more detailed comparisons, the geojson files were drawn simulaneously, in a single process. For each road where a bicycle lane marking was detected, or tagged in Open-

Figure 3.3: Example map comparing detected bicycle lane routes to OpenStreetMap. Green lines represent routes that are found in both maps. Blue lines represent detected routes that are not recorded in OpenStreetMap. Red lines would represent routes that are recorded in OpenStreetMap but not detected, however there are no such routes in this example.

StreetMap, a process "walks" down "nodes" in order. If the nodes are part of a "way" that has been tagged as a "cycleway", their coordinates are added to a line in the "OpenStreetMap" geojson. If a detection point is encountered at two intersections in a row, then the intersection nodes and the nodes in between them are added to a line in the "detected routes" geojson. At the same time, additional geojson files are drawn to show the route sections where both sources agree that there is a bicycle lane, and route sections where one source registers a bicycle lane and the other does not.

The total length of all routes in a geojson file can be calculated by using the Python "geographiclib" library to measure the distance between each pair of points along the route. Therefore, the geojson files can not only be used to compare routes on a map, but to measure the similarities and differences in metres.

The "Principal Bicycle Network" dataset is available in a geojson format. This allows a quick visual comparison with OpenStreetMap and the detected routes. It is possible to align the "Principal Bicycle Network" dataset to the OpenStreetMap data to perform a detailed comparison with measurements, however this was not attempted. The "Principal Bicycle Network" dataset had significant gaps in the survey areas, so significant that quantifying the differences in metres would not have been very meaningful. Please see the results section 4.2.

3.3 RQ3: Building a map of bicycle lane routes from dash camera footage captured in a survey area

The use of Google Street View images for detection of bicycle lanes has both advantages and disadvantages to consider. Google Street View image data is readily available for a wide area across many jurisdictions internationally, and it can be collected without a physical presence on location. However, usage comes at a cost, which may be a significant factor if mapping is required across a large area. The image resolution is limited to 640x640 per image, though perhaps this can be worked around by requesting many more images per location, each with a tighter field of view, at additional cost. It appears that distinct images are only available every 10 metres. The images may be several years out of date. The position of the camera cannot be controlled, and it might miss service roads or the other side of a divided road. With only one image available for each location, a bicycle lane marking might be obscured due to the presence of other traffic.

A local authority might prefer to gather their own footage, to address these shortcomings. For the third research question, we explore whether bicycle lanes can be detected using dash camera footage instead, perhaps captured from vehicles that must regularly traverse the roads in the area to provide other services, such as waste disposal or deliveries.

3.3.1 Choice of dash camera hardware

For these experiments, the "Navman MiVue 1100 Sensor XL DC Dual Dash Cam" was self-installed inside a sedan, at a cost of $529 AUD. This model was selected because it provided location metadata in NMEA files accompanying the MP4 video recordings. For each minute of footage recorded, the device produced an MP4 file and a corresponding NMEA file. The NMEA file provided the latitude, longitude, heading, and altitude at one second intervals.

Video footage was recorded at 60 frames per second, at 1920x1080 resolution. Only images from the front-facing camera were used.

The camera features a "wide angle" lens to capture a good view of the road and its general surroundings, however optical specifications such as focal length were not available from the manufacturer's website.

3.3.2 Gathering dash camera footage

On the 3rd of October, 2021, Melbourne set the record for the longest "COVID lockdown" in the world [46]. Severe restrictions were placed on residents being more than 5km from home, gradually easing to 10km and then 15km for a period of many months during the course of the research[47]. Permitted reasons to leave home were restricted. Therefore it was only possible to experiment with dash camera footage from a confined area.

Approximately 100 minutes of footage was gathered by driving within a local area around the outer metropolitan suburbs of Mount Eliza, Frankston, Langwarrin and Baxter in Melbourne, Victoria, Australia. Not every road in the area was covered. Roads were selected for inclusion in the footage to include all known bicycle lanes in the area, based on local knowledge and a review of the OpenStreetMap data. A variety of other roads were included, to cover roads with and without a paved shoulder, major divided roads and small local roads, and a mix of residential and commercial areas.

The dash camera video footage and corresponding NMEA files were transferred to a computer and processed using the Jupyter Notebook described in appendix A.1.12.

For each pair of files, the NMEA file was loaded into memory, then the video file was split into a series of images. The 60 frame per second footage was reduced to 5 frames per second by only sampling every 12th image. Sampling images at a higher frequency than this did not appear to be necessary, due to the high similarity between adjacent frames. Each extracted frame was assigned a location by extrapolating the readings from the two nearest measurements in the NMEA file. An entry was then added to an output "batch" CSV file with the path to the image file, and its location metadata.

3.3.3 Training the bicycle lane detection model

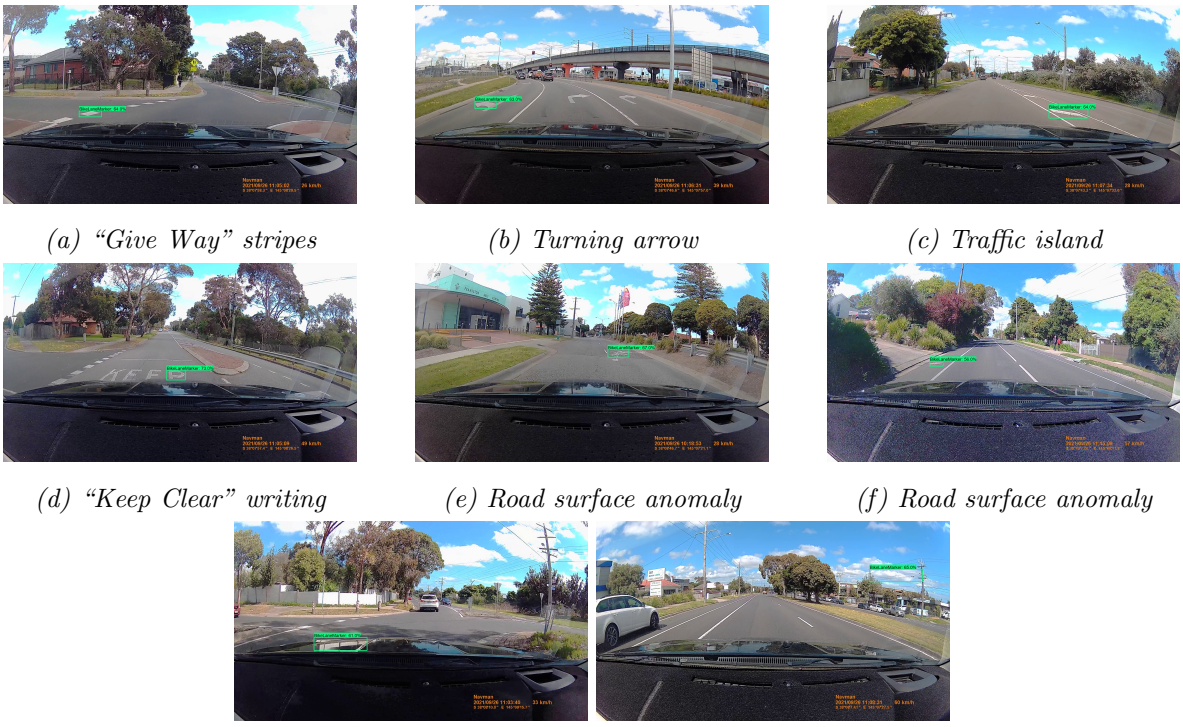
The dash camera footage was initially processed using the original model that was trained exclusively on Google Street View images in section 3.1, without any retraining based on dash camera images. The original model was able to detect every bicycle lane marking in the test footage, however further training and refinement was required to address some performance issues:

- Although the Google Street View-trained model was able to detect every marking during initial trials, it typically only did so when the marking was very close to the camera vehicle. Further training was apparently needed on footage from the dash camera in

order to allow it to detect markings that were further away, but still close enough to be clearly visible.

- There were also issues with false positives that had not been apparent when using the Google Street View model on Google Street View images. The model would sometimes yield false positives in the following situations:
  - Other white markings on the road, such as turning arrows, diagonal stripes to represent a traffic island, "keep clear" signs, or dashed lines advising traffic to "give way" at an intersection
  - Pale road defects or other anomalies such as manhole covers
  - Reflections from the bonnet of the camera vehicle
  - Clouds or random objects well outside the boundary of the road

See figure 3.4 for examples.



(a) "Give Way" stripes *(b) Turning arrow* *(c) Traffic island*

(d) "Keep Clear" writing *(e) Road surface anomaly* *(f) Road surface anomaly*

(g) Bonnet reflection *(h) White cloud*

Figure 3.4: False Positive Examples

A decision was made that the detection model needed to be trained with additional data from the dash camera footage. The footage was divided up into a "training area" consisting

of footage captured in the Frankston, Langwarrin, and Baxter area, and "test area" footage from Mount Eliza.

The Jupyter Notebook documented in appendix A.1.16 was used to process the dash camera footage from the "training area" with the initial detection model that had been trained exclusively on Google Street View images. Every image that yielded a detection – whether a true positive or a false positive – was then labelled and allocated to the "training dataset" and "test dataset" according to an 80:20 ratio. A total of 342 images were added to the datasets, supplementing the existing Google Street View images.

The original Google Street View training images were labelled with a single class "BikeLaneMarker". The new training images were labelled with the following additional classes, to encourage the detection model to think of many of the sources of false positives as something other than a bicycle lane marking: "ArrowMarker", "IslandMarker", "RoadWriting", "GiveWayMarker", and "RoadDefect".

The Google Street View images that had already been included in the dataset were *not* relabelled to consider the additional classes, though this was considered as an option if required to improve performance.

To reduce false positives due to reflections from the bonnet of the camera vehicle, or clouds and other random objects well away from the road, a detection mask was defined so that bicycle lane markings would only count as detections if they fell into an area on the left hand side of the road, in front of the bonnet of the car. See figure 3.5 for a visualization of this mask in an example image.

Figure 3.5: Detection mask to exclude car bonnet and right hand side of road.

The new images for the enhanced "training" and "test" datasets were gathered and labelled via the Jupyter Notebook documented in appendix A.1.13 and the "labelImg" tool. A new model was then trained and evaluated via the Jupyter Notebook documented in appendix A.1.14, and its predictions for the "training" and "test" datasets were analysed in detail

using the Jupyter Notebook documented in appendix A.1.15. Finally, the selected model was applied to the dash camera footage from the "test" area of Mount Eliza via the Jupyter Notebook documented in appendix A.1.16.

3.3.4 Mapping dash camera detections and comparing to other sources

Section 3.2.4 described the process that was used to produce geojson files to compare routes that were detected from Google Street View images to the routes that are recorded in OpenStreetMap. When sampling images from Google Street View, each sample location can be traced back to a "way" and a "node" in OpenStreetMap, where the latitude and longitude details came from. Therefore any detections are already precisely aligned to points in the OpenStreetMap extract.

Images that were collected from dash camera footage take their coordinates from the dash camera's GPS receiver, and they are sampled from locations anywhere along the road. To infer routes from bicycle lane markings detected in dash camera images, each detection location is mapped to the nearest intersection "node" in the OpenStreetMap data. Once this has been done, the rest of the process is the same as for detections from Google Street View images.

To find the nearest intersection node to a point, the Python "shapely" library is first used to find the nearest "way" to the point. The "shapely" library appears to use bounding boxes to accelerate this process: It was significantly faster than a brute-force search of all nodes in the OpenStreetMap data. Once the closest "way" is found, it is traversed to find the nearest "node" that is an intersection. At the same time, the process records the nearest "node" of any type, to help with traceability in case the detection occurred a significant distance away from the nearest intersection.

Wherever OpenStreetMap represented a highway or other divided road as multiple parallel ways, the above process was able to distinguish the side of the road that was closest to the camera position. It is important to gather footage from both sides of a divided road when measuring the differences between the generated map and OpenStreetMap. Otherwise, OpenStreetMap might count a bicycle lane on two "ways" for the divided road, where the generated map only counts one.

3.4 RQ4: Surveying other infrastructure details using dash camera footage

Further work was conducted to demonstrate how the approach of sampling images, processing them with a model, and correlating the results back to a map, might be re-used to survey other details of interest. A process was created to map any apparent "paved shoulders" on the side of the road in the survey area. Previous studies such as Klobucar & Fricker, 2007 [7] have observed that a paved shoulder can improve cyclist safety, even if it is not formally marked as a bicycle lane.

For a discussion of other areas of potential for future work, please refer to section 4.6.

To identify a paved shoulder, it is necessary to detect lane markings and the edge of the road on the side of the road that vehicles drive on. For the dash camera footage collected in this research project, this is the left-hand side of the road.

3.4.1 Correcting dash camera images for lens distortion

Lens distortion is where the optical properties of a camera's lens causes straight lines to appear curved in an image. The "OpenCV" library provides a method to correct distortion for a specific camera [48]. The camera is used to capture a range of images where a special calibration tool appears in frame. See figure 3.6 for some example images. An image of a "chessboard" pattern of known dimensions is held in front of the camera in different positions.

Figure 3.6: Example OpenCV camera calibration images

After the calibration images are collected, they are processed by OpenCV to create a mathematical model of the camera's lens distortion. With this model, OpenCV can apply a transformation to images from the same camera, to correct for lens distortion.

When searching for paved shoulders in a survey area, we are looking for lines in an image that are often straight. The dash camera being used to collect images was therefore calibrated, and in this exercise, all images were processed to correct for distortion. See figure 3.7 for an example raw image from the dash camera, and a corresponding corrected version.

(a) Uncorrected image *(b) Corrected image*

Figure 3.7: Example OpenCV camera calibration images

There was not much difference between the corrected and uncorrected images. The most obvious sign that the correction has been applied in figure 3.7 is that the top edge of the corrected image is bent downwards slightly, and the location/speed/time information that usually appears in the bottom right corner has been shifted. Although the operation to correct for lens distortion did not appear to have a significant impact when it came to detecting paved shoulders, it was retained in the pipeline to support any future photogrammetry work to estimate the widths of the lanes.

3.4.2 Detecting lane markings

A common solution to the problem of detecting lane markings involves applying a combination of the Canny edge detection algorithm proposed by Canny, 1987 [35] and the Hough transformation proposed by Hough, 1972 [36]. This general approach has been followed in many papers such as Li et al, 2016 [49] and Chai et al, 2014 [50], and it is frequently used in capstone projects in the self-driving car domain. The approach does not involve any machine learning or deep learning techniques.

The Canny-Hough approach typically works as follows:

a. Correct the image for lens distortion

b. Apply Canny edge detection to detect edges in an image [35]

c. Apply a mask to exclude edges outside a triangle representing the area immediately in front of the vehicle

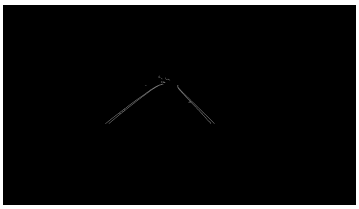
d. Apply a Hough transform [36] to convert these edges into lines, each line having a slope and an intercept.

e. Partition the lines into two groups: Lines that slope upwards when scanning from left to right, and lines that slope in the opposite direction. For each group of lines, take the average slope and intercept. These represent the assumed lane boundaries. Optionally redraw the image with the detected lines drawn as an overlay, to visualize them

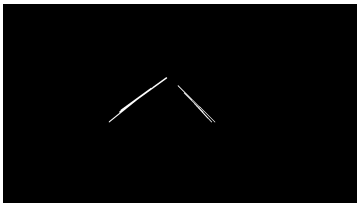
Please see figure 3.8 for example images to demonstrate the process.

(a) Corrected image

(b) Canny edge detection



(c) Apply mask



(d) Hough transform

(e) Average lines, superimposed on original image

Figure 3.8: Example sequence of Canny-Hough operations to detect own lane

The "OpenCV" library provides convenient tools to implement the required transformations.

Typically, the Canny-Hough process is used to detect the camera vehicle's own lane. In order to detect a paved shoulder, we first want to detect the boundaries of our vehicle's own lane, then perform a second pass at the image where the mask has shifted to focus exclusively on areas to further to the edge of the road.

A paved shoulder will be defined by two lines, each with a "slope" and an "intercept". One line will shared with a boundary of the camera's own lane,. The other line will be the edge of the road. To detect this extra line, we apply a mask to the Canny edge detection image just to the left of the camera's own lane boundary that was found in the first step. We then apply the same Hough transformation to detect lines, and average the ones with the expected "left" slope, to find a slope and intercept for the outer boundary of the paved shoulder. See figure 3.9 for an example.

If the Canny-Hough approach cannot find another line to the left of our the camera vehicle's

(a) Canny edge detection with previously detected lane

(b) Canny edges with new mask applied

(c) Image with all three detected lines

Figure 3.9: Example sequence of Canny-Hough operations to detect paved shoulder

own lane, it is a strong sign that there is no clearly defined paved shoulder on the road at that location. If a line is found, it could be a paved shoulder, or it could be a false positive caused by "noise" from something on the side of the road, but within the area included by the mask.

Please see figure 3.10 below for example images where lines were detected for a possible paved shoulder. In each of the example figures, two horizontal lines have been drawn. The lower horizontal line shows where the detected lines intersect a horizontal line just above the bonnet of the car, the nearest point in the frame with a clear view. The upper horizontal line shows where they intersect at an arbitrary point further into the distance. Vertical lines are also drawn to highlight the intersection points. Together, the horizontal and vertical lines are a visual aid to help assess the width and stability of any detected lanes, as the operator views the images quickly in sequence.

(a) A paved shoulder has been correctly identified

(b) A gutter has resulted in a very narrow shoulder area

(c) "Noise" from the straight lines in a fence

Figure 3.10: Example paved shoulder detections

Figure 3.10a shows a true positive match, where the boundaries of a paved shoulder have been correctly identified. Figure 3.10b shows a situation where a concrete gutter has resulted in a very narrow area being detected. This potential for this type of false positive can be mitigated by requiring a minimum width for any detected lane. Figure 3.10c shows a false positive match, where "noise" from a low wooden fence parallel to the road has caused the

average of lines to the left of the camera's old lane to appear to be a paved shoulder, off-road and in the grass. This phantom line is not "stable" and only appears for a few frames. This type of false positive can be reduced using metrics that require a paved shoulder to be consistently detected across most images in a road segment, and/or requiring that the slope and position of the line not vary too much from frame to frame.

3.4.3 Mapping paved shoulders across a survey area

The Canny-Hough lane detection method can be applied to all sample frames from the survey area, using a similar batch process to the bicycle lane detection process in section 3.3 and substituting a different detection model. The challenge is setting robust criteria for flagging a "detection" that will result in a sensible map. The simplest criteria would be to look for frames where lines were detected for both boundaries of a paved shoulder, but when it was applied to the training area footage, the results were very inconsistent. On roads where there was a paved shoulder or bicycle lane, the boundaries of that lane remained relatively stable from frame to frame, and the width between the boundaries of the detected area was relatively wide. There were few "skipped" frames where the boundaries were not detected. In contrast, on roads without a paved shoulder or bicycle lane, there may be many "skipped" frames, the slopes of the detected boundaries were inconsistent from frame-to-frame, and the detected boundary lines often had an intersection point much closer to the bottom of the frame.

The following solution was proposed:

- Each frame would be linked to its two nearest intersection "nodes" in the OpenStreetMap data. All frames associated with a single stretch of road between two intersections would be assessed as a group.
- If the model fails to find both boundaries of a paved shoulder in more than a threshold percentage of frames along a stretch of road, then it is assumed that there isn't one.
- Frames from locations within 30 metres of an intersection were excluded from consideration, to account for routine tapering of paved shoulders at intersections.
- A horizontal line is drawn across the frame towards a "horizon" as per figure 3.10. The width of the space between the boundary lines is measured at this height within the frame, in pixels. The mean width is calculated across all frames in the stretch of road where boundary lines were detected. If the mean is less than a threshold number of pixels, then the paved shoulder is too narrow, and it is assumed not to exist. This accounts for very narrow margins such a gutter that are of no use to a cyclist.

- If the model finds both left and right boundary lines for a possible paved shoulder, use the slopes and intercepts to calculate an (x, y) coordinate relative to the bitmap image where those two boundary lines intercept. Calculate the standard deviation of the x and y intersection values across all frames in the stretch of road where both boundary lines are defined. If the standard deviation in either the x or y dimension is greater than a threshold number of pixels, then the boundary lines are considered to be too inconsistent across images. The stretch of road is assumed not to have a paved shoulder.

The thresholds described above were tuned by analysing the dash camera footage from the "training" area and producing a CSV file with one record per frame, in chronological order. Each frame was mapped to the nearest "way" and the two nearest intersection "nodes" in the OpenStreetMap data, to associate it with a particular stretch of road, from intersection-to-intersection. It was also labelled with the name of the road, for traceability. The required summary statistics were calculated for each group of frames associated with a stretch of road, and these were attached to the CSV file as additional columns. Finally the footage was reviewed, and using the road names, changes of heading and "node" IDs as a guide, each record was labelled to record whether there was really a paved shoulder there or not.

A manual tuning process was performed, filtering the rows based on the column recording the "ground truth", to find thresholds that appeared to correctly predict the outcome as often as possible.

The model was then applied to the footage from the Mount Eliza "testing" area to see how it performed with the selected thresholds.

To construct a map of detected paved shoulders, the dataset was summarised to a list of distinct combinations of "way" ID and intersection "node" IDs for each road section where a paved shoulder was predicted. This information was cross-referenced to the OpenStreetMap data for each "way" ID, and a line was "drawn" in a geojson file for all "nodes" on the way between the two intersection nodes, including the intersection nodes themselves. The geojson file could then be drawn on map in a Jupyter Notebook using the Python "ipyleafelet" library as per sections 3.2 and 3.3.

In this exercise, lane detection was attempted from the dash camera footage only, as a demonstration of how the techniques developed to address the first three research questions could be re-used to gather other information visually. It could be applied to Google Street View images, however it was only attempted with the dash camera footage, where it is easier to ensure that each image is taken with a heading that is consistent with the direction of the road.

Chapter 4

Results and Discussion

4.1 RQ1: Training a model to identify bicycle lanes in Google Street View images

Initial performance targets of 80% recall and 80% precision were chosen for this research question, to provide a solid baseline before attempting to infer and map routes from the detection points as part of the subsequent phases of research. To infer routes from detection points with the method proposed in section 3.2, we need to be confident that bicycle lane markings will be detected at most intersections, if present, otherwise the missed detections would cause an incorrect discontinuity in the inferred route. A recall of 80% would mean that we can expect to detect the markings most of the time. We also want to limit false positives, because two adjacent false positives could cause us to infer a route where there is none. A precision of 80% would mean that only a small minority of our detections are false positives, and we would be very unlucky to infer a long route spanning multiple intersections that is not real.

An initial dataset of 256 Google Street View images was gathered, with the option to gather more, if necessary, to achieve acceptable performance for the first two research questions. They were labelled and split into 205 "training" images and 51 "test" images, according to an 80:20 split. One of the 51 "test" images did not contain any bicycle lane markings. It was initially included in the dataset in error, however it was retained because it proved useful for quick sanity checks, to ensure the model was not blindly flagging bicycle lane markings in every image.

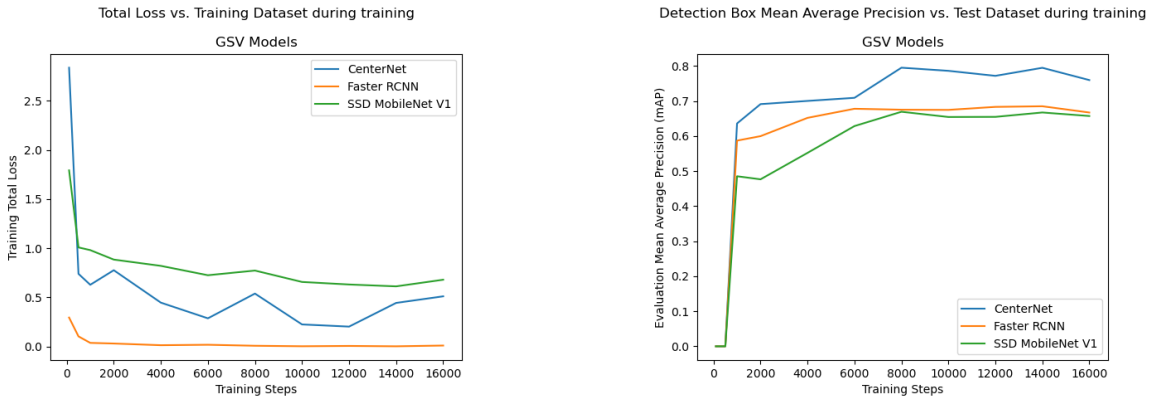
Five pre-trained object detection models from the TensorFlow 2 Model Garden were trained

and evaluated using these "training" and "test" datasets, through a process of transfer learning. Each candidate model was trained for 2,000 steps at a time, with training paused every 2,000 steps to evaluate performance against the "test" dataset. Training continued until it yielded no significant improvement to the mean Average Precision of bounding box detection when evaluated against the "test" dataset.

Table 4.1 lists the five pre-trained object detection models that were tested, and the mean Average Precision of their bounding box detection when evaluated on the "test" dataset after 2,000 training steps. The "EfficientDet D1 640x640" and "SSD ResNet101 V1 FPN 640x640" were unable to detect any bicycle lane markings even after 4,000 training steps, so training on them was ceased, to focus on the other three models that were producing more immediate results.

Pre-Trained Model	**Steps**	**Evaluation mean Average Precision**
CenterNet HourGlass104 512x512	2,000	0.691116
Faster R-CNN ResNet50 V1 640x640	2,000	0.476506
SSD MobileNet V1 FPN 640x640	2,000	0.599674
EfficientDet D1 640x640	n/a	n/a
SSD ResNet101 V1 FPN 640x640	n/a	n/a

Table 4.1: Bounding Box mean Average Precision per pre-trained model (2,000 training steps)



(a) 'Training "Total Loss"

(b) Evaluation "mean Average Precision"

Figure 4.1: Training and Evaluation of Detection Models based on Google Street View images

Figure 4.1a shows the "Total Loss" reported during training of the three remaining models. All three models reached a point of diminishing returns at 2,000 training steps.

Figure 4.1b shows the Bounding Box mean Average Precision during evaluation with the

"test" dataset. Continuing the training beyond 2,000 training steps yielded small improvements in that metric. However, the most important measure of success for this model was whether it could correctly identify whether a bicycle lane marking appeared in an image or not. It was not necessary to know precisely where the bounding box was, so improvements to the Bounding Box mean Average Precision are only useful up to a point. Using the tool described in appendix A.1.8, each of the three models were applied to the "training" dataset and the "test" dataset, to inspect its predictions for each image. The images were examined to confirm that the bounding boxes were drawn in a sensible location, and to confirm whether the model had successfully found a bicycle lane marking in each image. The results are recorded in 4.2.

Pre-Trained Model	**Train**		**Test**	
	False Pos^n	**False -ve**	**False Pos**^{n}	**False -ve**
CenterNet HourGlass104 512x512	0	11	0	13
Faster R-CNN ResNet50 V1 640x640	1	0	4	0
SSD MobileNet V1 FPN 640x640	8	30	4	9

Table 4.2: Errors in the detection of bicycle lane markings in the "train" and "test" datasets after 2,000 training steps. (The number of frames with a bounding box drawn in a false position, and the number of frames where the model came to a false negative conclusion.)

The "Faster R-CNN ResNet50 V1 640x640" model was able to successfully locate a bicycle lane marking in *every single image in "training" and "test" datasets where one appeared.* It was not fooled by a solitary image with *no bicycle lane markings* that had been included in the "test" dataset. It produced a bounding box in the wrong position for four images in the test dataset. It easily met the target performance metrics, with 100% recall and 92% precision.

The "Faster R-CNN ResNet50 V1 640x640" model, trained to 2,000 training steps, was chosen as the preferred model, and used to create a bicycle lane route maps for research question 2.

The initial dataset of 256 images appeared to be sufficient to get reliable detection results with this model, so no further images were collected from Google Street View for the "training" and "test" datasets at this stage. If better performance was required to obtain sensible results when drawing maps of bicycle routes in the next section, we would address that by increasing the number of training images available, add labels to common sources of false positive results, then re-assess the candidate models after further training.

4.2 RQ2: Building a map of bicycle lane routes from Google Street View images in a survey area

4.2.1 Mount Eliza

The model that was selected in section 4.1 was applied to a sample survey area in the vicinity of Mount Eliza. This area was chosen due to its accessibility: It was one of very few areas with bicycle lanes that was reachable within the COVID-19 lockdown restrictions that were in place at the time [46] [47]. It was therefore possible to validate the results via an in-person survey, in addition to comparing the results to OpenStreetMap and the "Principal Bicycle Network" dataset. It also allowed for comparison with results from dash camera footage in section 4.3.

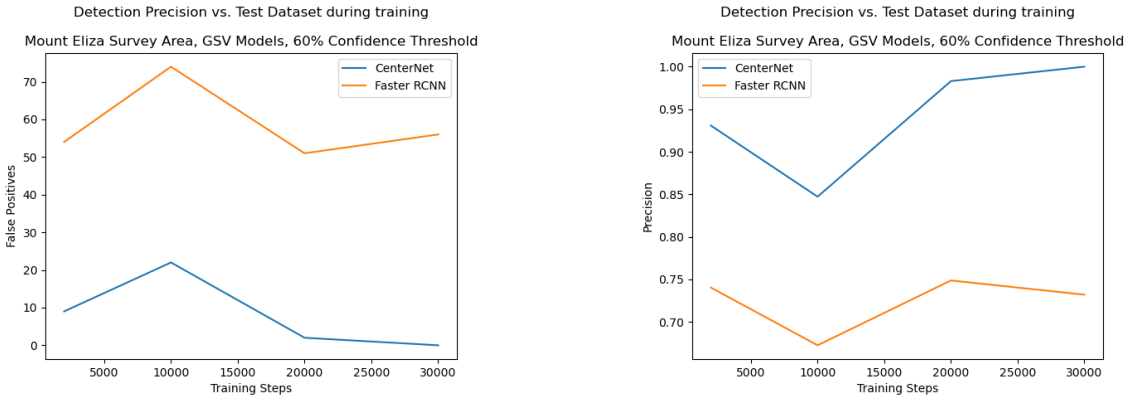
A total of 1,113 locations were sampled from the survey area, at points within 20 metres of every intersection. Four images were downloaded via the Google Street View API at each location, a total of 4,452 images at a cost of approximately $31 USD. The exact survey area within Mount Eliza was chosen to include a mix of roads with and without bicycle lanes, ranging from no-through roads to a highway.

Figure 4.2: Principal Bicycle Network existing and planned routes

Figure 4.2 is a screenshot from an interactive map that in the Jupyter Notebook described in appendix A.1.11, showing existing and planned bicycle routes in the survey area according to the "Principal Bicycle Network" dataset. The blue bounding box shows the survey area. The red lines show "existing" routes, and the blue lines show "planned" routes. The "Principal

Bicycle Network" dataset is out of date. It correctly records that there is an existing route on the Nepean Highway, in red. However, the long, straight "planned" route, Humphries Road, has existed for at least several years, and so has the route to the north-east of it, on Baden Powell Drive. The planned route on Baden Powell Drive to the west of Humphries Road does not yet exist.

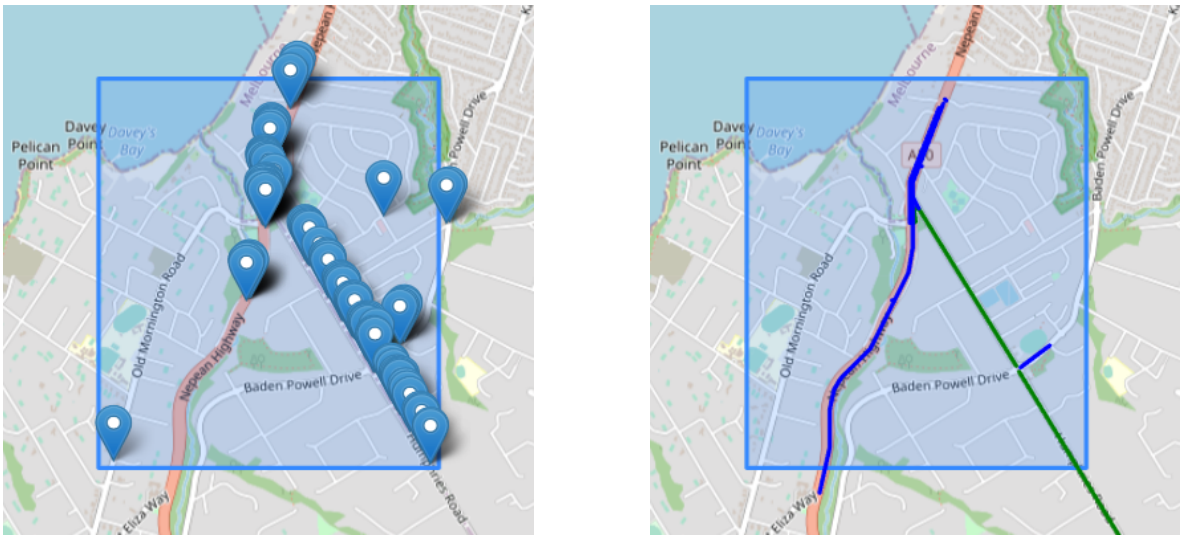
In section 4.1, the models were trained for 2,000 training steps, beyond which further training did not appear to yield any improvements when the model was evaluated against the 51 image "test" dataset. The Faster R-CNN model appeared to be the most promising. However, when that model was applied to Google Street View images from the Mount Eliza survey area, the initial results were poor. Precision was significantly lower than it had been in the initial evaluation, with many false positive detections for random objects such as bushes, or leaves on the road. The models received further training on the original "training" dataset, up to 30,000 training steps, and the CenterNet model had the best performance, producing zero false positive detections, and a correct map, after 30,000 training steps. It was therefore used to construct maps from the Google Street View images in the survey areas. See figure 4.3 for training statistics.



(a) False Positives *(b) Precision*

Figure 4.3: Extending model training steps to deal with false positives

Figure 4.5a shows the individual points that were detected by the model from Google Street View images sampled near intersections in the survey area. Figure 4.5b shows the bicycle lane routes that were inferred from those detections, according to the method described in section 3.2.3. The routes in figure 4.5b are colour-coded to highlight areas of agreement and disagreement between the detection model and OpenStreetMap. Where both the detection model and OpenStreetMap agree that there is a bicycle lane, routes are coloured green. They agree that there is a bicycle lane on Humphries Road. Routes that were detected by the



(a) Detection Points within survey area *(b) Detected routes c.f. OpenStreetMap*

Figure 4.4: Map of bicycle lanes in Mount Eliza based on Google Street View images

model but not recorded with a "cycleway" tag in OpenStreetMap are coloured blue. The OpenStreetMap data is missing the cycleway on the Nepean Highway, and the fragment of the Baden Powell Drive cycleway that was detected near the boundary of the survey area. Routes that are recorded in OpenStreetMap but were not detected would be coloured red, however there were no such routes in this survey.

Table 4.3 shows a comparison between the routes detected for Mount Eliza from the Google Street View images to what is recorded in OpenStreetMap, in metres. The two sources agreed upon 2,344 metres of bicycle lane routes. The detection model found an additional 3,217 metres that were not recorded in OpenStreetMap, including parts of the divided Nepean Highway where both sides of the road were counted due to them being mapped in OpenStreetMap separately. Distances are approximate due to the level of precision available when calculating distances between each point in the geojson files.

Route source	**Metres**
Detection Model	5,216m
OpenStreetMap	2,344m
Both	2,344m
Detection model only	2,787m
OpenStreetMap only	0m

Table 4.3: Comparison of routes detected in Mount Eliza to OpenStreetMap, GSV images

Based on an in-person survey of the area, and inspection of the Google Street View images that were downloaded, the detection model correctly detected all bicycle lanes within the survey area, with no false positives. It identified two routes that had not been recorded in OpenStreetMap.

4.2.2 Heidelberg

After surveying Mount Eliza with both the Google Street View approach in this section, and the dash camera approach in section 4.3, the Google Street View approach was revisited to test it on a survey area in Heidelberg, Victoria. The purpose of this exercise was to assess how well the detection models generalised to a more heavily built-up area, closer to the city. It included both residential and commercial areas.

(a) Overview of survey area *(b) Magnified view*

Figure 4.5: Map of bicycle lanes in Heidelberg based on Google Street View images

The green lines represent routes where both the detection model and OpenStreetMap agreed that a bicycle lane route is present. The red lines represent routes that are recorded in OpenStreetMap but not detected. Any red lines outside the bounding box describing the survey area can be ignored. The Google Street View images for the red lines within the survey area were examined manually, to assess whether the model had produced sensible results based on the input. Zooming into the red diagonal line on Studley Road near the Heidelberg train station, it can be seen that the model did correctly predict that there was a bicycle lane, but it only marked it on one side of the road, where it is tagged separately on both sides of a divided road in OpenStreetMap. The remainder of the red lines are routes

where there is no bicycle lane visible in the Google Street View images, and the model has produced a sensible result. See figure 4.6 for example images from Cape Street and Stradbroke Street. It is possible that this is caused by out-of-date Google Street View images, however the images for Stradbroke Street were only six months old, and the streets did not seem wide enough to easily accommodate the addition of a bicycle lane. These routes might be errors in the OpenStreetMap data. An on-site survey would be required to confirm this.

The blue lines near the Austin Hospital and Heidelberg Station are false positives. They are caused by the road being close enough to see a bicycle lane marking on a nearby road.

(a) Cape Street, Heidelberg, May 2019

(b) Stradbroke Street, Heidelberg, March 2021

Figure 4.6: Example Bicycle lane routes that are tagged in Heidelberg but not apparent in the Google Street View images

There is a very short segment where OpenStreetMap records a cycleway on Lower Heidelberg Road near the intersection of Banksia Street. This route would be barely larger than the nearby intersection, and is likely to be an error in OpenStreetMap.

The detection model marked some routes on Lower Heidelberg Road, to the north and south of Banksia Street. These were investigated and found to be false positives, caused by large "Keep Clear" signs painted in white on the road. Keep clear signs are a known source of false positives that were added to the training dataset and labelled, to support the dash camera model in section 4.3. However there are not very many examples in the training dataset. This type of false positive error should be addressed by collecting and labelling more training data.

A comparison between the improved detection model results and OpenStreetMap in terms of route lengths can be found in table 4.4.

Route source	**Metres**
Detection Model	4,563m
OpenStreetMap	5,317m
Both	2,094m
Detection model only	469m
OpenStreetMap only	2,536m

Table 4.4: Comparison of routes detected in Heidelberg to OpenStreetMap, GSV images

The model was able to identify all bicycle lane routes that were recorded in OpenStreetMap and visible in the Google Street View images. It produced 469 metres worth of false positives, due to bicycle lane markings on one road being visible from multiple points on a nearby road. And it only drew a bicycle lane on one side of a divided road, where OpenStreetMap had recorded bicycle lanes on both sides, separately. An approach using dash camera footage could help to remove these issues, by providing footage for both sides of each road, with the camera reliably facing down the the road at each point, and masks to avoid detection of bicycle lane markings on nearby roads.

4.3 RQ3: Building a map of bicycle lane routes from dash camera footage captured in a survey area

4.3.1 Re-training the model for dash camera footage

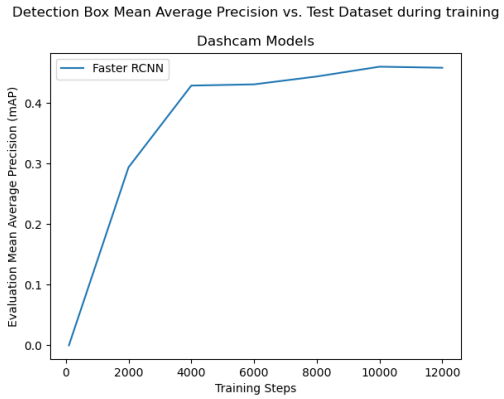
Due to COVID-19 restrictions [46] [47], dash camera footage could only be gathered in a circular area with a 15km radius, much of which was a body of water. Therefore, the range of areas available to gather dash camera footage for research question 3 was restricted to a handful of outer metropolitan suburbs. Footage from the Frankston, Langwarrin and Baxter area was used to train a the detection model to work with dash camera footage, and footage from Mount Eliza was used to test it.

Training was conducted as per the approach documented in section 3.3. Initially, the original model from section 4.2 that had been trained exclusively on Google Street View images was used to process the dash camera footage from the training area. The initial positive matches – whether true positives or false positives – were added to the “training” and “test” datasets

according to an 80:20 split. The resulting "training" dataset consisted of 205 Google Street View images and 252 dash camera images. The "test" dataset consisted of 51 Google Street View images and 85 dash camera images.

Another "Faster R-CNN ResNet50 V1 640x640" model was then trained and evaluated using the combined Google Street View + dash camera footage datasets. To deal with remaining false positives, the dash camera images were labelled with additional classes to distinguish common misleading features from bicycle lane markings, and a detection mask was applied to remove reflections from the bonnet of the car, and distractions from the right hand side of the road. Please see section 3.3 for a full discussion of these enhancements. The results observed from the model after every 2,000 training steps are recorded in figure 4.9 and table 4.5.

(a) 'Training "Total Loss"



(b) Evaluation "mean Average Precision"

Figure 4.7: Training and Evaluation of Detection Models based on dash camera images

Model	Steps	Train			Test		
		Total Loss	False +ve	False -ve	mAP	False +ve	False -ve
Faster R-CNN	2,000	0.0852	1	8	0.293945	3	9
Faster R-CNN	4,000	0.0985	0	8	0.428875	1	2
Faster R-CNN	6,000	0.0298	0	0	0.430906	3	3
Faster R-CNN	8,000	0.0222	0	0	0.443913	1	2
Faster R-CNN	10,000	0.0292	0	0	0.46007	2	2
Faster R-CNN	12,000	0.011	0	0	0.458306	0	1

Table 4.5: Training and evaluation results for the Faster RCNN model with dash camera footage

Figures 4.7a and 4.7b show that there were diminishing returns from further training beyond

4,000 to 6,000 training steps, in terms of both the "Total Loss" reported in the training process, and the "mean Average Precision" of the bounding boxes reported from the evaluation process against the "test" dataset.

The model was applied to the "training" and "test" images via the tool documented in appendix A.1.15 to examine the bounding boxes it chose to draw, and whether it believed there was a detection or not. The results of this review are found in the "False +ve" and "False -ve" columns reported in Table 4.5. After 6,000 training steps, the model was able correctly detect all bicycle lane images in the "training" data, with no false positives, false negatives, or misplaced bounding boxes. In the test data, there were a total of 3 false positives and 3 false negatives out of 136 test images. These false results came from a mix of Google Street View images and dash camera images. After 8,000 training steps, the performance improved further, in that there was 1 false positive and 2 false negatives when analysing the test data, and these were all for challenging Google Street View images. See figure 3.3.3.

(a) False Positive GSV image example

(b) False Negative GSV image example 1

(c) False Negative GSV image example 2

Figure 4.8: False Positive and False Negative results recorded when applying the preferred Faster R-CNN model to the combined Google Street View and dash camera "test" dataset, after 8,000 training steps

In figure 4.8a, the model has correctly identified two bicycle lane markings in the central business district of Melbourne, but it has also identified a white strip that was installed at a pedestrian crossing to assist people with impaired vision. A very high confidence of 95% was assigned to the false positive detection. To assist the model to deal with this situation better, more training images would be required, with these objects given a label to steer the model away from thinking of them as a bicycle lane marking.

In figure 4.8b, the model has missed a very faint bicycle lane marking that was alongside the camera vehicle, in the shadow of the vehicle. This was a very challenging image due the the

faintness of the marking, and it was common for the models tested in this research project to miss it.

In figure 4.8c, the model has missed a bicycle lane marking on the opposite side of the road, at a challenging angle and distance. In general, most models struggled with this test image as well.

Training was terminated after 8,000 steps, and the model was then used to conduct a survey from dash camera footage in the "test" are of Mount Eliza. At 8,000 training steps, the only incorrect predictions reported were for Google Street View images. For this section, the model was being applied to images from dash camera footage, where there were many more frames available to compensate for any missed detections in individual frames.

4.3.2 Generating and comparing maps

Figure 4.9 shows a comparison between the bicycle lanes in Mount Eliza that were detected based on dash camera images, and the routes that are recorded in OpenStreetMap as having bicycle lanes, for roads that were surveyed in the footage only. The green routes show where there is agreement between the detection model and OpenStreetMap. The blue routes show bicycle lanes that were detected but not recorded in OpenStreetMap. The red routes show bicycle lanes that are recorded in OpenStreetMap but not detected.

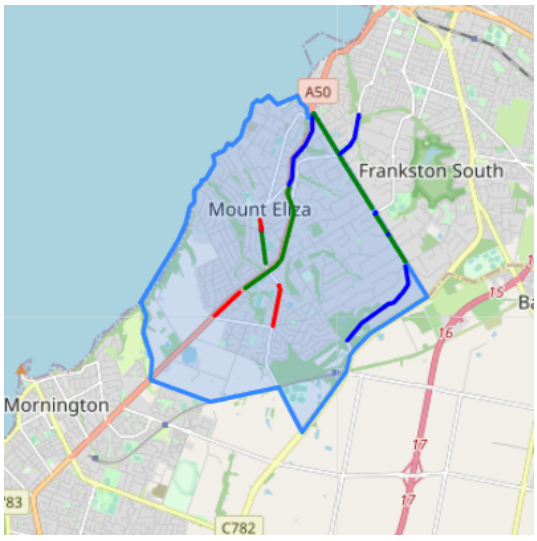


Figure 4.9: Detected bicycle lanes in Mount Eliza based on dash camera images

The model correctly detected three sections that are not recorded in OpenStreetMap. There is a cycleway on the Nepean Highway that has not been tagged in OpenStreetMap in the

northern section depicted on the map There is a narrow cycleway on Baden Powell Drive to the north-east of Humphries road. And there is a cycleway on Two Bays Road to the south-east of the survey area.

There is a short segment in the northern section of Mount Eliza Way where a bicycle lane is tagged in OpenStreetMap but was not detected by the model. The detection model appears to be returning a reasonable result in this instance, the bicycle lane does not extend beyond the roundabout at Kenaud Avenue.

There is a segment on Wooralla Drive where a bicycle lane is tagged in OpenStreetMap. This is an error, there is no bicycle lane on that road.

There is a longer segment on the Nepean Highway, leading up to the intersection with Tower Road, where the detection model failed to draw a bicycle route. This is the only part of the map where the detection model appears to have made an error. During the survey, the camera vehicle turned into Tower Road and missed that segment. If the vehicle had continued along Nepean Highway, it might have picked up the next bicycle lane marking and drawn a continuous route.

A comparison between the improved detection model results and OpenStreetMap in terms of route lengths can be found in table 4.6.

Route source	**Metres**
Detection Model	13,413m
OpenStreetMap	8,903m
Both	7,484m
Detection model only	5,795m
OpenStreetMap only	9,67m

Table 4.6: Comparison of routes detected in Mount Eliza to OpenStreetMap, dash camera images

Overall, the detection model performed well within the survey area. It is possible that multiple surveys spread over multiple weeks would have filled in the gap at the edge of the survey area, on the Nepean Highway. The model detected three route segments that are not recorded on OpenStreetMap, and correctly identified two route segments that appear to be recorded in OpenStreetMap in error. It missed one route segment that was recorded in OpenStreetMap, at the edge of the survey area, but perhaps repeat surveys or a survey route that extended further down the Nepean Highway may have caught this. Both OpenStreetMap and the detection model included two routes that are not listed in the "Principal Bicycle Network"

dataset as existing routes.

4.4 RQ4: Surveying other infrastructure details using dash camera footage

4.4.1 Tuning the Paved Shoulder Detection Model thresholds

The model proposed in section 3.4 was first applied to the training area footage from Frankston, Langwarrin, and Baxter, and then validated against the same test area footage from Mount Eliza as used for research question 3.

It was found that, when processing an individual frame, the proposed method was vulnerable to "noise" from objects just outside the paved area, and this led to the use of thresholds described in section 3.4.3. After analysing the observed data and ground truth for the "training" area, the following thresholds were set:

- A minimum of 80% of frames in a road segment must have a paved shoulder detected
- The mean width of paved shoulders detected in a road segment, measured at the upper top horizontal line drawn in the overlay, must be at least 75 pixels
- The standard deviation of the X and Y co-ordinates at which the paved shoulder boundary lines would intersect must be less than or equal to 50 pixels

These thresholds were intended to ensure filter out road segments with inconsistent detections, paved shoulders that are too narrow to be used by a cyclist, or where the detections are caused by "noise" that also causes the perceived paved shoulder boundaries to jump around wildly.

4.4.2 Results for the test area

The map of paved shoulders that was generated for the test area of Mount Eliza from the dash camera footage can be found in figure 4.10.

The map gives relatively robust results on Humphries Road and the Nepean Highway, where there is a bicycle lane, a special case of a paved shoulder. Part of the Nepean Highway was not detected, where the lane markings have faded and the "noise" from the foliage dragged the boundary of the lane detected for camera vehicle itself too far to the left.

Figure 4.10: Detected paved shoulders in Mount Eliza based on dash camera images

The other lines drawn on the map represent true paved shoulders, but they were easily interrupted by "noise" from the side of the road, or parked cars. Many of the interruptions in the paved shoulder route on Old Mornington Road, to the west of the the Nepean Highway, were attributable to parked cars.

The model did not present any false positives, and it appeared to find most roads with paved shoulders, but the routes drawn were often broken up.

4.5 Limitations

4.5.1 Use of Google Street View images

The use of Google Street View images to map bicycle lane routes in Mount Eliza was effective, and added value, finding routes that had not already been recorded in either the official "Principal Bicycle Network" dataset or the crowdsourced OpenStreetMap database. Repeating the exercise in the more heavily built-up suburb of Heidelberg was less effective: There were some false positives, and every real route that was identified was already included in the OpenStreetMap database.

The main limitations of the Google Street View approach are the cost of images and gaps in coverage they can provide.

The Google Street View API appears to be able to provide a distinct snapshot every 10 metres.

A town or suburb can easily have many kilometres of roads, and if every snapshot in a town were sampled, or multiple towns were surveyed, the amount payable to Google for the use of their API could add up quickly. The proposed solution attempted to mitigate this by only sampling the area immediately around intersections, and making assumptions that a bicycle lane route is continuous if only interrupted by missed detections at one or two intersections in a row. If API costs needed to be reduced further, then perhaps OpenStreetMap tags could be used to eliminate roads that are for "local traffic only", where bicycle lanes might not be as necessary. Or, the survey could focus on roads that are adjacent to known bicycle lane routes, and might therefore be a continuation waiting to be discovered.

One significant challenge is the recency of the available Google Street View images. For example, the latest available Google Street View images for the Mount Eliza survey area are dated October 2019. If any new bicycle lanes had been constructed in the past two years, the model would be unable to detect them. The Google Street View approach can only bring a dataset of bicycle lane routes up to date as far as Google's last visit to the area.

The Google Street View camera vehicle typically only provides a view from one lane, on one side of the road. Bicycle lane markings on the other side of the road may be difficult to spot, especially for a divided road with foliage or other barriers dividing the traffic. This is especially important if you want the bicycle lane route map to distinguish which side of the road a bicycle lane is on, if not both. It may be impossible to detect a bicycle lane in a service road running in the opposite direction to the lane from which the Google Street View snapshot was taken. Hopefully these challenges are sometimes mitigated by taking samples near intersections from multiple approaches, where some of the bicycle lane markings might be clear.

The model was able to detect bicycle lane markings even when they were moderately far into the distance, despite the 640x640 resolution limit. Sometimes, the bicycle lane marking was "smeared" by artefacts in the Google Street View image, but with enough training examples, the model was able to detect these correctly.

The approach of using a 360 degree view at each intersection to infer routes was vulnerable to false positives where bicycle lane markings were visible from a separate road, nearby. Using front-facing footage from a dash camera, where the camera is always aligned to the direction of the road, masks could be applied to resolve this issue.

The survey area of Mount Eliza is in an outer metropolitan area, and the COVID-19 lock-down restrictions have triggered political debate about whether it should even be considered "metropolitan" at all. Most residential roads in the area do not have pedestrian footpaths.

When the model is applied to higher-density suburbs, such as Heidelberg, it is likely to need more images in the "training" and "test" datasets to cover the additional distractions that might appear on the side of the road. Detection masks might also assist. With more traffic, it might be much more difficult to detect a bicycle lane marking on the other side of a road or intersection, and there might therefore be a much higher reliance on detecting markings immediately in front of and to the left of the camera vehicle, where it is less likely that another vehicle could be blocking the view.

Difficulties in heavily built-up areas might matter less if the bicycle lane routes in those areas have already been well-mapped due to their priority and a large number of interested cyclists who could contribute to OpenStreetMap.

4.5.2 Use of dash camera images

The use of dash cameras to gather images would overcome many of the limitations of the Google Street View data. If the latest available images are too old, then there is freedom to go and collect more at any time. Images can be gathered from both sides of the road, from any lane or service road required. The driver can drive conservatively, to limit the degree to which the view is obstructed by the vehicle in front. Many more snapshots are available: If the camera vehicle is driving at 50kmph, that equates to just under 14 metres per second. A camera recording at 60 frames per second would record a frame every 23cm, compared to the 10 metre intervals provided between snapshots by Google Street View. The limiting factor becomes how much data is meaningful to store and review, and missed detections may be compensated for in nearby frames or repeat surveys.

The results obtained from the dash camera survey were at least the equal of the Google Street View survey of a smaller sample area. There is reason to believe that the dash camera approach has the potential to be more robust and able to deal with the technical challenges that limit the Google Street View approach.

The challenge with the dash camera approach is with logistics. With Google Street View, you can survey as wide an area as you can afford to download. With the dash camera approach, someone must install and maintain a dash camera, drive the required routes, and send the data off for processing. This could be done by interested volunteers for individual areas, or by individual local governments who are aware that they have significant deficiencies in their data. Perhaps the most promising prospect would be to "incidentally" gather footage from local government vehicles that regularly drive most routes for another purpose. For example, garbage trucks would cover all residential streets on a regular basis. The business

case for collecting, managing, and processing the footage could be justified not only by the value of mapping bicycle lane routes, but by the value of any other information that could be gathered visually and correlated to a location. For example, a local council might be interested in knowing whether the garbage truck *really* missed someone's bin, or whether the truck is being sent back because the resident actually forgot to put their bin out. Or they may be interested in monitoring local roads for defects that need to be repaired.

To scale the dash camera approach across multiple cameras, or to reproduce the research, it should be noted that the optical characteristics, resolution, and frame rate of each camera may differ, and placement of the camera in different positions, angles, and vehicles may materially affect the perspective of the images. These factors might result in the need to gather additional training and test images to ensure that the solution generalises well across cameras. Any model that depends on a model to correct for lens distortion should go through its own calibration process, as per section 3.4.1.

The method that was used to quickly "bootstrap" a set of set of dash camera images for training and validation based on the Google Street View-trained model worked well in this instance. But the process of randomly splitting those images into "training" and "test" datasets would not have ensured the absolute independence of the "test" images. None of the "test" images were included in the "training" dataset, however some of the "training" images may have been very similar due to them being taken from a nearby location. If the model had failed to generalise well despite good performance in its evaluation, this may have needed more attention.

It is likely that more "training" and "test" images would be required for inner metropolitan areas, but it was not possible to test this hypothesis.

4.5.3 Detection of paved shoulders

The model that was applied to the problem of detecting paved shoulders had significant limitations. It was intended as a "proof of concept" to show that the general technique used to map bicycle lanes from the bicycle lane markings could be applied to other problems where information can be collected visually.

The model relied on explicit programming of an algorithm, and manually selected thresholds applied to summary statistics based on observations in a training area. It did not attempt to apply any of the more recent advances in machine learning or deep learning, where the model "learns" to solve the problem for itself from the data. It was able to create an approximate

map, without false positives, but the detected routes were often interrupted by parked cars or roadside "noise".

These results were achieved by using training data from a suburb that was adjacent to the survey area, and they most likely shared many characteristics. It might be difficult or impossible to tune the model to work well across a wider variety of settings.

A more robust solution might be provided by applying deep learning techniques to identify the road surface, with a training dataset that is diverse enough to cover all conditions that may be encountered in the survey area. If the model is also able to identify vehicles and their bounding boxes in each image, then it might be better able to deal with parked vehicles, both in assessing whether a paved shoulder is there, and assessing its usefulness in light of the obstructions.

The application of the Canny-Hough approach to lane detection across a variety of environments is challenging, and could be considered a separate research project in itself. To get a more consistent detection of the road boundary in the face of road-side "noise", it would be worth exploring deep learning image segmentation techniques to train a model that can detect the road surface, similar to the approach taken by Mamidala et al., 2019 [38]. A deep learning model could also be trained to detect parked cars, and take them into account: A parked car should not be allowed to interrupt a detected paved shoulder route, but if there are many parked cars on the shoulder, is it worth recording the route at all?

4.5.4 Survey area constraints

COVID-19 lockdown restrictions imposed significant constraints on the location and variety of survey areas. Access to dash camera footage was especially limited, while for the Google Street View approach, it constrained the areas in which an absolute "ground truth" could be established via an in-person survey. Further work would be required, once lockdown restrictions are eased, to show how well the dash camera solutions work in a greater variety of environments.

During this research, surveys were generally conducted at a scale of one town or suburb at a time. OpenStreetMap data for an area was parsed and then cached in memory in Python "dict" objects for quick access. It seems reasonable to partition this work by town or Local Government area, and the methods used worked well at that scale. Some operations might not scale well if it were applied to a whole State or Country at a time. For example, for every point sampled in dash camera footage, we find the closest "way" and then the closest

"node" in OpenStreetMap, so that the data is aligned to the OpenStreetMap version of the shape of each road. This allows us to identify and measure any differences in routes. If the process was applied at a State level instead of a Suburb level, the search for the closest point in OpenStreetMap might take significantly longer.

4.6 Opportunities for future research

Future work could examine how well the solutions proposed generalise to other survey areas.

If the cost of using the Google Street View API is a concern, then other strategies to reduce API consumption could be investigated. For example, the current solution takes four images at each location. How much impact does it have if we only have the front and rear views? If we approach an intersection from multiple angles, and the angles are not perfectly square, that might result in unnecessary requests for images with headings that are only a few degrees different from ones we already have. How much can we save by restricting this? Can we make assumptions about which classes of road we should survey, based on their tags or other information in the OpenStreetMap database? For example, a "tertiary" road might be a good candidate for a bicycle lane, but a minor road designed for local traffic only might not be. How frequently are bicycle lanes tagged in the OpenStreetMap database for each class of road? Would it be useful to limit our search to roads that are adjacent to known bicycle routes, where they might provide a useful but unrecorded continuation?

This research focussed on surveying one aspect of the road infrastructure that is known to influence bicycle safety and uptake: The presence of a bicycle lane. There are many other factors that could surveyed visually. With a more robust way to detect the edge of the road and the each lane, we could not only survey the paved shoulders, but also measure the width of lanes. We could use another deep learning model to detect vehicles that are obstructing the bicycle lane, or rows of parked cars that present a hazard due to the chance that a door will suddenly open into the bicycle lane, or the car might suddenly pull out. We could correlate the data to topographical maps or GPS altitude readings to identify road segments with a significant uphill or downhill slope. We could look for road surface defects or debris in the bicycle lane that might suggest that maintenance is required.

The solution that was implemented has contributed some general approaches that could be reused for other applications beyond the domain of bicycle safety. It provided: a method to sample Google Street View images for a list of locations, and review them for possible inclusion in a labelled dataset for the purposes or training or testing a deep learning model;

a method to sample Google Street View images for roads or intersections in a survey area; a method to infer routes on a map from points where a model has detected something of interest; and a method to align a route from dash cam footage and its GPS metadata to the closest route in OpenStreetMap to make comparisons. Any application where there is potential value in recognising something visually, counting incidences, and drawing them on a map, could re-use some of the techniques used in this research project.

Chapter 5

Conclusion

A "deep learning" model was successfully trained to detect bicycle lane markings based on a dataset of 256 Google Street View images that were randomly sampled from intersections along known bicycle lane routes across the State of Victoria, Australia. The model was based on the "Faster R-CNN ResNet50 V1 640x640" model that is published in the TensorFlow 2 Model Garden, pre-trained on the COCO 17 dataset. It achieved 100% recall and 92% precision when evaluated against 51 test images that had been withheld during its training. This easily satisfied the performance targets proposed in research question 2.

The model was applied to Google Street View images across a survey area in the outer metropolitan suburb of Mount Eliza, Victoria, Australia. A process was created to correlate the detection points to OpenStreetMap data about the road network to infer and map bicycle routes. All bicycle routes in the area were successfully detected, including routes not previously identified in OpenStreetMap or the "Principal Bicycle Network" dataset, satisfying the aims of research question 2. There were no false positive routes. When the same process was re-applied to the heavily built-up suburb of Heidelberg, the results were not as strong: Not all known bicycle routes were detected, and some of the inferred routes were not real. Additional training data would be required to improve performance, and future research should examine whether better results could be obtained using dash camera footage.

The initial model that was trained on Google Street View images generalised well enough to detect many bicycle lane markings in dash camera footage collected in a training area. This was used to gather images from the dash camera to be labelled and included in the training and test datasets, to further improve the performance of the model when processing dash camera footage. The final model was also able to correctly identify all bicycle lane routes in the survey area, from dash camera footage, including routes not previously recorded, with

no false positive routes. It therefore satisfied the aims of research question 3, and shows promise when working in outer metropolitan areas where bicycle lane routes are not already well mapped. It was not possible to test the model in more heavily built-up areas due to movement restrictions in place as part of the government response to COVID-19.

Finally, it was found that the approach that was used to map bicycle lane routes from dash camera footage could be re-used to visually survey other details about the road infrastructure. As a demonstration, a basic Canny-Hough model was created and tuned in an attempt to detect paved shoulders on the side of the road, suitable for a cyclist to use to maintain some separation from other traffic. The model was able to create and draw a map, though there were some issues with gaps in the routes drawn, due to interference from parked cars and other roadside "noise". With future research, it would be possible to plug in a deep learning model to improve performance, or plug in a different model to detect something else, for another application. This satisfied research question 4.

Appendix A

Process documentation

To assist with reproduction of results, code is provided at GitHub at:

`https://github.com/tylerrmit/minor_thesis.git`

Data necessary to reproduce the results, including dash camera footage collected during the experiments, can be found in an archive on `figshare.com` under the Digital Object Identifier (DOI):

`10.25439/rmt.16862518`

The code consists of a series of Jupyter Notebooks for high-level operation of the process, supported by Python classes for implementation details. The Jupyter Notebooks are numbered to help the user through the proper sequence, and they are documented below.

Dash camera MP4 footage and NMEA metadata that was collected during the experiments are provided on FigShare, along with calibration footage and other files. These are intended to help reproduce the results of the dash camera experiments, however a full process is provided for anyone who wishes to assemble their own equipment and collect data in a different area.

A.1 Jupyter Notebooks

All Jupyter Notebooks listed below follow a convention that immediately after a title and description, there is a single "Config" cell where *all* customization or configuration for the notebook can be made. E.g. there may be a parameter that can be adjusted to survey a different town or suburb. Shortly after the "Config" cell, any required Python modules are

imported, and then then any variables that are derived from the "Config" variables (e.g. paths to files or directories) are defined. To run the Jupyter Notebooks in general:

- Review the description and documentation
- Review and update the "Config" cell
- Run all cells in the notebook, or run the cells one at a time if specifically directed by the notebook

Whereever an operation is expected to take a long time, progress bars are provided.

The notebooks are designed with the assumption that the "jupyter notebooks" command will be run from the "minor_thesis" top-level directory that was checked out from GitHub as the current working directory. They expect to find a "data_sources" subdirectory for a lot of the data and configuration files, and a "jupyter" subdirectory where the notebooks live.

A.1.1 01_Reverse_Geocode_PBN

We create a dataset of Google Street View images containing bicycle lane markings by examining images of intersections along known bicycle routes, because the markings are consistently found at intersections. To identify candidate intersections to survey for sample data, we can use either the "Principal Bicycle Network" dataset or OpenStreetMap "cycleway" tags as our source of known bicycle routes.

If we wish to use the "Principal Bicycle Network" dataset as the basis for our sampling strategy, we use notebooks 01 and 02. If we wish to use OpenStreetMap data, we can use notebook 03 instead.

The purpose of notebook 01 is to apply "reverse-geocoding" to all points in the "Principal Bicycle Network" dataset, to help identify intersections where we might look to gather Google Street View images containing bicycle lane markings, for our dataset. The process takes latitude/longitude coordinates, and finds the street name and town/suburb/city, which we can use to search for intersections in the next step.

It requires a copy of the "Principal Bicycle Network" dataset from `data.gov.au` in geojson format, and connection details for a local Nominatim server as per appendix C.2.

It will create the file "pbn_exploded.geojson" as its output. This notebook can be skipped by using OpenStreetMap as the driver for sampling intersections along known bicycle lane routes, via notebook 03.

A.1.2 02_Identify_Candidate_Intersections_PBN

This notebook takes the reverse-geocoded "Principal Bicycle Network" data from notebook 01, and correlates it to OpenSteetMap data to find the intersections. It outputs a file "pbn_intersections.csv" for use in the sampling process in notebook 04.

This notebook requires a local Nominatim server. It also requires an OpenStreetMap XML extract for Victoria – see the notes at the top of the notebook for instructions on how to obtain that. It can be skipped by using OpenStreetMap as the driver for sampling intersections along known bicycle lane routes, via notebook 03.

A.1.3 03_Identify_Candidate_Intersections_OSM

This notebook creates a list of intersections along known bicycle lane routes using OpenStreetMap "cycleway" tags as its source of information. It is an alternative to using notebooks 01 and 02 that should generalise well for other states and countries, where the "Principal Bicycle Network" dataset does not provide coverage.

It requires an OpenStreetMap XML extract for Victoria – see the notes at the top of the Jupyter Notebook for instructions on how to obtain that. It outputs a file "osm_intersections.csv".

This notebook is provided as an option to help reproduce the results outside of Victoria, however it was the "Principal Bicycle Network" approach in notebooks 01 and 02 that was actually used.

A.1.4 04_Gather_Dataset_GSV

This notebook provides a GUI for sampling Google Street View images at intersections along known bicycle routes (the output of notebook 02 or 03).

Update the "Config" to give it the name of the list of locations to sample from, usually "pbn_intersections.csv" or "osm_intersections.csv". Then run it from top to bottom. The last cell contains a GUI using "ipywidgets".

You will need to provide a Google Street View API key that is linked to your Google Account and billing info, as per `https://developers.google.com/maps/documentation/streetview/get-api-key`. The API key is stored in a text file "apikey.txt" in the parent directory of the directory from which Jupyter Notebooks was run, usually the directory that also contains a copy of the "minor_thesis" directory that was checked out from GitHub.

You may need to enable ipywidgets in Jupyter Notebooks before running the "jupyter notebooks" command. Please find instructions on the commands to do that online, they depend on whether Anaconda is being used, or MacOS, etc.

See figure A.1 for a screenshot of the GUI.

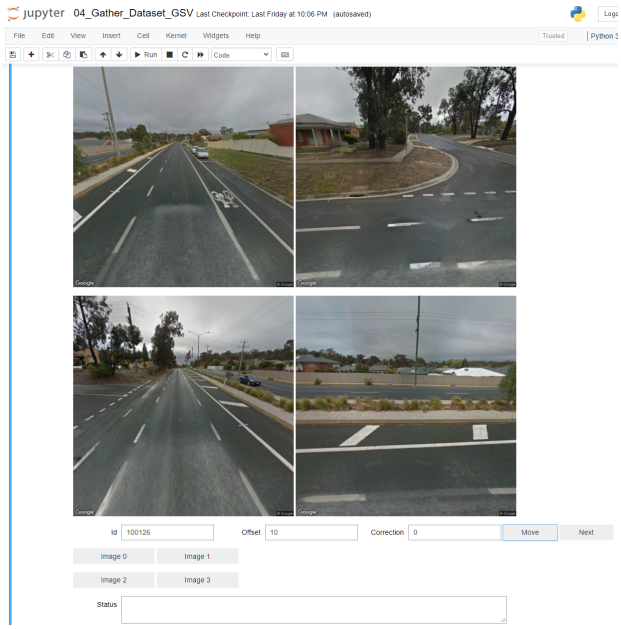


Figure A.1: GUI for gathering Google Street View image dataset

Four images are arranged on screen, with corresponding buttons "Image 0" through "Image 3". The top-left image is "Image 0" and represents the forward-facing view at the sample location. The bottom-right is "Image 2" and represents the rear-facing view. The other two images show what appeared alongside the camera vehicle.

The GUI will automatically open with the first randomly-selected sample point. Examine the four images, and for each image where a bicycle lane marking appears, press the corresponding button to record the "hit" in the output CSV "hits.csv". This will mark the image for inclusion in the dataset in the next stage of the process, when images are "labelled". In the example in figure A.1, a bicycle lane marking is clearly visible in the top-left image, and the operator would hit the "Image 0" button to record it.

By default, the "Offset" and "Correction" text boxes are set to zero.

If you see a bicycle lane marking off in the distance in the forward direction, and you want to move the camera closer to it, enter "10" in the "Offset" text box, and then press "Move". The "Status" box will briefly mention that the next image is being loaded, and then the images

will update to a camera position 10 metres down the road. To move in the reverse direction, enter a negative offset.

Moving forward or backward down the road depends on having the camera set at the correct heading for the road in the first place. If the camera appears to have been placed so that the "forward" direction is at an angle to the road, correct it by entering a number of degrees in the "Correction" text box, and then pressing "Move". The orientation will shift. Once you are happy the orientation is roughly aligned with the road, you can then enter an offset to move the camera to a better position.

When you are finished trying to obtain sample images from this location, press the "Next" button to move to a new sample location.

With practice, multiple sample images can be obtained per minute. If there is no sign of any bicycle lane marking in the area, even in the distance, then you can quickly skip to the next location by pressing "Next". It is not uncommon for the "Principal Bicycle Network" dataset to flag a route in a country town such as Warrnambool as an existing on-road bicycle route, only to find that all Google Street View images appear to show an informal paved shoulder, with no bicycle lane markings anywhere along the road. If you can see bicycle lane marking in the immediate area, hit a button for each matching image and move on. If you can see one off in the distance, move the camera 10 or 20 metres to capture it.

Every set of images that is downloaded from Google Street View will be cached in the "gsv" directory under "data_sources".

The output of this notebook is a "hits.csv" file in "data_sources" containing a list of cached Google Street View images where suitable bicycle lane markings were found. You can monitor the size of "hits.csv" as you go along, to keep track of how many images you have found for your dataset. Once you have enough images, proceed to notebook 05 for the labelling process.

A.1.5 05_Copy_GSV_Images_For_Labelling

In notebook 05, we take the Google Street View images previously identified, copy them to a single folder, label them with the "labelImg" tool, split them into "training" and "test" folders.

The key "Config" items are a version suffix to give to the dataset, e.g. "V1", and the percentage of the dataset that should be held in reserve for "testing" of models and not included in the "training" data, e.g. "20" for 20%.

Do not run all cells of this notebook at once. You will need to pause in the middle to run the external "labelImg" tool, from `https://github.com/tzutalin/labelImg`, or a suitable alternative.

In the first part of the notebook, the code will take every image that was flagged in "hits.csv" during the notebook 04 process, and copy them all into a single "dataset" folder.

After those cells have run, the notebook will tell you to stop and run the "labelImg" tool to "label" the data. This involves telling "labelImg" where to find the new "dataset" folder with all the images, and then, one image at a time, you draw a bounding box around any object of interest, and give it a label, e.g. "BikeLaneMarker" for the bicycle lane markings we are training the model to look for. See the instructions in the notebook and the official documentation for "labelImg" for more details.

Once the labelling is done, you can proceed with running the final cell in the notebook, which will randomly select a percentage of the labelled images in the "dataset" file and move them to a "test" directory, with the version suffix specified in the "Config". The remainder will be moved to an equivalent "train" directory. The labelled images in the "train" and "test" directories are ready to be turned into "tfrecord" files suitable for TensorFlow processing in the next notebook.

A.1.6 06_Model_Training_Setup

The purpose of notebook 06 is to download and compile the required code to support TensorFlow Object Garden development, and to set up "tfrecord" files for training and validation of the models.

This notebook is worth running one cell at a time, to watch for errors about missing dependencies, and address them with "conda install" or "pip install" as required.

Cells 4 and 5 will download the required TensorFlow Object Garden code from GitHub, and then run the required "protoc" commands to set it up.

Cell 6 will run a validation script to check that TensorFlow development has been set up correctly. Not all errors will necessarily mean that the setup is broken, but watch its output to make sure you are not missing any dependencies.

The final two cells will create a "label map" file, which must include all the labels that were used in "labelImg", and then convert the image and label data in the "train" and "test" dataset directories into "tfrecord" format, for use with TensorFlow.

A.1.7 07_Model_Training_GSV

Notebook 07 allows you to download a pre-trained model from the TensorFlow 2 Object Garden, and configure it for training on the custom dataset.

`https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md`

In our case, we ran this notebook to configure multiple candidate models from the garden, before settling on the "CenterNet HourGlass104 512x512" model based on its performance.

In the "Config" section, give the URL and Name of the pre-trained model you wish to download, from the table on the above "TensorFlow 2 Detection Zoo" webpage. You can also specify the name you'd like to give to a model when it has been trained and you want to create a "frozen" model from it, e.g. "centernet_V1".

Then run the notebook from top to bottom. The notebook will download the model, and update its configuration to look for our custom objects, instead of the "COCO 17" objects it was originally trained to detect. In the results of the very last cell, it will print three commands that you can run manually, from a command prompt. It is possible to update the notebook to run the commands from the notebook, but generally you get better, immediate feedback about the progress if you run them yourself from a command prompt.

Run the first command to train the model on our "train" dataset for 2000 steps. Every 100 steps, it will give you statistics how long it is taking to perform each step, on average, and "loss" statistics to show how well it thinks it is performing with the "train" data. You can re-run this command with a higher number of training steps to continue the training from where it left off.

The second command allows you to evaluate how the model performs with the "test" data that was withheld from training. It can give you statistics such as "mean Average Precision" (mAP) and "Total Loss". How did its predictions compare to the actual labelled bounding boxes in the test data?

The third command allows you to make a smaller "frozen" copy of the model. A "frozen" model is useful if you want to:

- Keep a smaller copy of the model as-at a certain number of training steps
- Copy the model to a different system where it will be applied

Use notebook 07 to download a pre-trained model and set it up for training, then run the training commands recommended by it until you are happy with the evaluation statistics and think it's ready to apply. If the model does not perform well, you might need to review the dataset (possibly adding more images) or increase the number of training steps (if that is improving the performance) or switch to a different pre-trained model from the "zoo".

The next notebook can help show how the model is performing beyond the statistics, and give insight into what it is struggling with.

A.1.8 08_Apply_Model_to_Train_and_Test_GSV

The "evaluation" script in notebook 07 will take a model and apply it to the "test" data, and produce performance statistics as output. If you want to *see* how it is performing, notebook 08 can help. It processes the data from the "train" and "test" directories, and outputs images that have been sorted into "hits" and "miss" directories depending on whether a bicycle lane marking was detected, and it will draw a bounding box and confidence score around any detection.

In the "Config" section, give it the name of the model, which will either be the "pre-trained_model_name" from notebook 07, or the "frozen_name" if you created a "frozen" copy of the model. Give it the version suffix of the dataset, e.g. "V1". And then the "confidence threshold" at which the model should assume that an object has been detected, e.g. "0.55" means it will score a "hit" if its confidence is 0.55 or greater.

Run it from top to bottom, and it will produce output in `detections/train_XXX` and `detecions/test_XXX` where `XXX` is the version suffix of the dataset. Within those directories, there will be a `hits` folder and a `miss` folder, containing output images. You can review the `hits` to see if it accurately placed bounding boxes around the detect objects. You can review the `miss` folder to see the images where it failed to make a detection, where one was expected. Were there any unusual challenges with the image? Is it reasonable that the object might not have been clearly visible? Would it help to add more images like this to the dataset?

A.1.9 09_Filter_OSM_to_Local_Area

Notebooks 01 to 08 prepare and train a model to detect bicycle lane markings. Next, we need to choose a "survey area". We will use OpenStreetMap data for that area to identify locations

to sample – near each intersection – and then we will download the images and process them with our model.

The purpose of Notebook 09 is to create a geojson file describing the shape of a local area, such as a town or suburb, based on government-issued data. We can then use this to extract OpenStreetMap data for that specific area, and use it as our "survey area".

In the "Config" section, choose a suburb name as the "locality_name". It will expect to load the government-issued "LGA_boundaries_VIC.geojson" file from `data.gov.au` and reduce it down to a geojson name for the selected suburb.

Download the OpenStreetMap data for Australia from `download.geofabrik.de` as instructed in the notebook.

If you run the notebook from top-to-bottom, it will output two suggested "osmium" commands. These are designed to reduce the full "country-level" extract of OpenStreetMap data down to data for the survey area. The first command will reduce the data to the exact official shape of the suburb. The second command will create an extract for a slightly bigger bounding box, to help identify intersections near the margin on the survey area.

Install the "osmium" tool and run the commands suggested by the notebook to create the required OpenStreetMap extracts.

A.1.10 10_Apply_Model_to_Survey_Area_GSV_Images

This notebook will identify sample points near intersections in the survey area, download Google Street View images, apply the model, and record any detected bicycle lane markings in the output "detection_log.csv".

If you wish to control Google Street View API costs, please run this notebook one cell at a time.

In the first phase, it will use the OpenStreetMap extracts for the survey area, from notebook 09, to create a list of sample points within a certain number of metres of each intersection. The "margin" parameter lets you specify how many metres either side of each intersection you want to sample. If it is set to 0, the process will only take samples from the middle of each intersection. If it is set to 10, it will also take samples 10 metres either side of the intersection. The recommended setting is 20, because the Australian standards specify that a bicycle lane marking should be within 15 metres of every intersection, and the Google Street View API apparently only gives a distinct image every 10 metres.

At the end of the first phase, the notebook will report how many sample points were identified for margins of 0m, 10m and 20m. The process will be taking four images per sample point, so multiply the proposed number of sample points by four to get an estimate of the cost. If there are too many sample points, consider using the "osmium" tool with its "–bbox" option to extract a smaller rectangular area within the suburb as a sample, instead of processing the whole town.

If you proceed with the rest of the notebook with the proposed sample points, it will download the required images from Google Street View, process them with the model, record the detections to a subdirectory for the "locality_name" within the "detections" directory, and create geojson files to be used with the maps in notebook 11. It will infer detected routes from the detection locations according to some rules, and compare these to what OpenStreetMap said with its "cycleway" tags. As part of the comparison, it will tell you how many metres of bicycle lane routes were detected, how many metres were tagged in OpenStreetMap, how many metres they agree, and how many metres each source had a route that the other one did not.

The "detections" folder for the "locality_name" will also have "hits" and "miss" folders where output images (with bounding box overlays) can be examined. It can be helpful to quickly flick through the images in the "miss" folder to see if there are many clear bicycle lane markings that were missed. It can also be helpful to flick through the "hits" folder to see if there were many false positives, and what might have been causing them.

A.1.11 11_Map_Bicycle_Lanes_GSV

Notebook 11 produces interactive maps to view and compare bicycle lane routes for the survey area:

- Map 1 shows each detection as a point on the map
- Map 2 shows the bicycle lane routes that were inferred from the detections
- Map 3 shows bicycle lane routes that are tagged as "cycleway" routes in OpenStreetMap
- Map 4 shows a comparison between the detected routes and OpenStreetMap, with different colours to represent where they agree, and where one had a route section that the other did not.

Set the survey area name in the "Config" section, and run the notebook from top to bottom. You can zoom in and out and pan around each map as required. The "Config" section allows you to set a default zoom level.

A.1.12 12_Split_Dashcam_Footage

The next group of notebooks are dedicated to detecting bicycle lane routes from dash camera footage instead of Google Street View images.

Notebook 12 expects to find a folder within "data_sources" containing MP4 video files and associated NMEA files. These came from a "Navman MiVue 1100 Sensor XL DC Dual Dash Cam" that records one MP4 video and one NMEA file per minute.

In the "Config" section, specify the name of the folder containing the footage, this is effectively the "survey area". You can also specify how many image frames you want to sample, up to the number of frames in the original footage. E.g. you can reduce 60fps footage down to 5 frames per second.

Run this notebook from top to bottom. It will split the video files into images on disk, and create a file "metadata.csv" with the location for each image based on the NMEA data.

The first progress bar shows how many of the video files have been processed. Subsequent progress bars show progress through each individual video file as it loads.

A.1.13 13_Copy_Dashcam_Images_For_Labelling

The first time you are processing dashcam footage, it is recommended that you skip straight to notebook 15 and attempt to detect bicycle lane markings with an existing model that was trained with Google Street View images. That model will not necessarily have ideal performance on the dashcam images, which are taken from a different perspective, with different equipment, at a different resolution. But running a first pass with the GSV model can help to quickly identify images that could be taken from the dashcam footage and added to the training and validation dataset.

Use notebook 15 to process footage from a "training" area. Then come back to this notebook 13.

In the "Config" directory, set the version suffix for the previous GSV dataset that you wish to add to, e.g. "V1". Then set the version suffix for the new dataset version you wish to

create e.g. “V2”. Then start running the first few cells.

Any “hits” from the initial training – whether they are true positives or false positives – will be read from the “detection_log.csv” file by notebook 13, and copied to a “dataset” folder.

Stop running the notebook, and label the images in the “dataset” folder with “labelImg” as directed. To avoid false positives, it was found to be helpful to label additional classes such as turning arrows, traffic islands, and give way markings, to avoid confusing their simple white markings with bicycle lane markings. See the notebook instructions for further details.

Once the images in the “dataset” directory have been labelled, continue running the notebook. Existing “train” and “test” images from the previous dataset will be copied into this new version, according to their original splits. Then any new images from the dashcam will be split into the “train” and “test” directories to join them. Finally, “tfrecord” files will be created from them, for TensorFlow to use.

A.1.14 14_Model_Training_Dashcam

Once new “train” and “test” datasets have been constructed, with dashcam images supplementing the original Google Street View images, we can train a new model.

This notebook 14 is equivalent to notebook 07, but for the new dataset that includes dashcam images. We select a pre-trained model that we want to work from, download it, and run the recommended training, evaluation, and model-freezing commands.

Beware that if you give exactly the same “pretrained_model_name” as you did for GSV training, it will try to continue training that model where you left off. You may prefer to give a different “pretrained_model_name” to start the training from scratch.

A.1.15 15_Apply_Model_to_Train_and_Test_Dashcam

This notebook 15 is equivalent to notebook 08, but for the new dataset that includes dashcam images. Run this notebook if you want to visualize the results of applying the model-in-training to the “train” and “test” datasets.

A.1.16 16_Apply_Model_to_Dashcam_Footage

This notebook 16 is equivalent to notebook 09, instead of sampling Google Street View images near intersections, we are processing every image we extracted from the video footage

in notebook 12.

Based on the findings from cycling the "training area" footage through notebook 15 -¿ 13 -¿ 14 in a loop, some enhancements were made to the model to reduce false positives. See the notebook and "Methods" section of this document for further details.

In the "Config" section, you can specify the folder/location being surveyed, and the minimum confidence score required to flag a detection. Specify the name of the model or frozen model you wish to use, and the dataset version prefix to ensure the correct label map file is used.

The "mask" option allows you to specify which part of the frame to run detections on, instead of the whole frame. This will exclude detections in areas where we do not expect to see a bicycle lane marking, and avoid some potential false positives. The mask boundary will be drawn on the output images in the "hits" and "miss" folders, so you can see what was considered by the detection model.

There are some further options to control how many detections are required in a general area before a detection is counted for the purposes of inferring a route. This is to avoid false positives that occur in a single frame with nothing detected in adjacent frames. By default, each detection requires two further detections within 50 metres, but a minimum of 10 metres away to avoid duplication when the camera is motionless.

The end result of this notebook is a "detection_log.csv" with all detection points, and "detection_log_filtered.csv" where one-off detection points that are not supported by nearby detections are excluded.

A.1.17 17_Convert_Detection_Log_to_GeoJSON

In notebook 17, we take the "detection_log_filtered.csv" output from notebook 16, and correlate it to the OpenStreetMap data, infer bicycle lane routes, map them, and compare to the routes that are tagged as "cycleway" in the OpenStreetMap data. Differences and agreement between routes are measured in metres, and geojson files are created in order to produce maps in the next notebook.

Check the "Config" section to make sure the correct dashcam image folder has been specified for the survey area, along with the "locality" name for the OpenStreetMap extract. Then run the notebook from top to bottom.

A.1.18 18_Map_Bicycle_Lanes_Dashcam

This notebook 18 is equivalent to notebook 11, except it is producing maps to view and compare routes that were detected from dash camera footage, instead of Google Street View images.

- Map 1 shows the bicycle lane routes that were inferred from the detections
- Map 2 shows bicycle lane routes that are tagged as "cycleway" routes in OpenStreetMap, where the route was covered by the dash camera footage
- Map 3 shows a comparison between the detected routes and OpenStreetMap, with different colours to represent where they agree, and where one had a route section that the other did not.

A.1.19 19_Calibrate_Dashcam

The final group of notebooks were used to demonstrate how the general technique of applying a model to images and correlating the result to existing geospatial data could be used to build a dataset about road infrastructure. In this example, we looked at whether we could detect a "paved shoulder" to the left of the camera vehicle's lane, where a cyclist might ride their bike with some separation from other vehicles. Future work could be looking at estimating lane widths, detecting parked cars or other obstacles in the bicycle lane, or detecting hazardous road surface defects.

Notebook 19 is used to calibrate a model to correct images from the dash camera to account for optical distortion. Follow the instructions in the notebook (and the reference material linked in the notebook description) to print out a calibration tool that looks like a chessboard. Record footage of this calibration tool being held up in front of the dash camera at different positions in the frame, at different distances from the camera.

Update the "Config" to point to the folder where the calibration videos can be found, and record the number and size of the squares as printed on the calibration tool. Then run the notebook.

The notebook will split the video into images at 1 frame per second, then follow a standard OpenCV calibration process to produce a model. The model will then be saved as a "dashcam_calibration.yml" file in the "data_sources" directory.

At the bottom of the notebook, you will see an example of an original image from the dashcam, and a corrected image where the model has been applied to make straight lines appear straighter.

A.1.20 20_Detect_Paved_Shoulders_Dashcam

This notebook will re-process the video footage from a survey area, just as notebook 12 did. But this time:

- The model to correct for optical distortion, from notebook 19, will be applied, then...
- A "lane detection" model will be applied, to detect the camera's own lane and then the next lane to the left, if any
- The detected lane lines will be overlaid onto output images so that they can be visualized
- For each road segment, from intersection to intersection, an assessment is made as to whether there seems to be a paved shoulder, based on whether one was found in most images, whether it was wide enough, and whether the boundaries were stable enough across the frames in the group. See the "Methods" section and the notebook for further explanation.

The first significant output of this process is the "metadata_with_summary.csv" file, which is recorded in the "split" subdirectory under the footage folder. It contains one record per frame, with the summary statistics for the road segment that are used to decide whether there might be a paved shoulder along that segment, or not.

Then, a geojson file is created, to allow us to draw on a map where the paved shoulder detection criteria was met in "metadata_with_summary.csv". This is mapped in notebook 21.

A.1.21 21_Map_Paved_Shoulders_Dashcam

This notebook draws the detected "paved shoulders" from notebook 20 on a map, so that they can be visualized. Most bicycle lanes should also appear as paved shoulders, so it is useful to compare this output to the other maps in notebooks 18 and 11.

A.1.22 22_Apply_Model_To_Video_Stream

This notebook is used to create demonstration videos where a dash camera video is loaded, a model is applied to all frames, and then an output video is created with a detection overlay. Sample outputs are available in the FigShare archive under the "demos" directory.

Appendix B

OpenStreetMap XML Concepts

The use of OpenStreetMap XML extract data is fundamental to the solutions proposed in this research project. In this appendix, we briefly describe some key concepts in the data, and provide example XML data to demonstrate them.

B.1 Ways

A "way" in the OpenStreetMap XML data is a line that can be drawn on a map. Typically, it will represent a road segment, as per the example in figure B.1. However, it could also be the path of a natural feature such as a creek or coastline. It could be an off-street walking track or bicycle trail. Or it could be the boundary of a reserve or an estate.

Each "way" has a unique "id", followed by a list of "node references" and "tags".

The "node reference" links to a "node", which has a latitude/longitude position. Therefore, the node reference list effectively describes path of the "way" if it were drawn on a map.

The "tags" describe characteristics of the way. In figure B.1, the "way" is a "tertiary" road named "Humphries Road" where the speed limit is 60 (kmph) and there is a bicycle lane and a sidewalk.

A "way" can be only a *part* of a longer road. If different sections of a road have different properties, the road will be broken up into multiple "ways". E.g. a change of speed limit would split the road into two "ways".

```
<way id="26662301" version="32" timestamp="2020-08-27T04:26:02Z">
  <nd ref="30204323"/>
  <nd ref="638346068"/>
  ...
  <nd ref="2117131454"/>
  <nd ref="638346153"/>
  <tag k="cycleway:left" v="shared_lane"/>
  <tag k="highway" v="tertiary"/>
  <tag k="maxspeed" v="60"/>
  <tag k="name" v="Humphries Road"/>
  <tag k="sidewalk" v="right"/>
  <tag k="surface" v="asphalt"/>
</way>
```

Figure B.1: Sample OpenStreetMap XML "way"

B.2 Nodes

In a separate section of the OpenStreetMap XML file, each of the "nodes" are listed. A "node" is a point on a map, usually just with a latitude and longitude, but occasionally they can have tags to describe special properties, such as the presence of traffic signals as in figure B.2.

```
<node id="30204322" version="18" timestamp="..." lat="-38.1655191" lon="145.1016428"/>
<node id="30204323" version="21" timestamp="..." lat="-38.1667063" lon="145.1017474">
  <tag k="highway" v="traffic_signals"/>
</node>
<node id="30204324" version="18" timestamp="..." lat="-38.1674697" lon="145.101785"/>
```

Figure B.2: Sample OpenStreetMap XML "nodes"

B.3 Intersections

If a "node" is shared by two more "ways", it may be an intersection. By checking the name of each "way" we can see whether it is an intersection, or just a "node" at which single road was split into two ways due to a change in a characteristic.

Therefore, to find "nodes" that are intersections, look for "nodes" that are shared by "ways" with multiple distinct names.

B.4 Cycleways

For the purposes of this project, if a "way" has a tag that begins with "cycleway", then OpenStreetMap believes that it has some sort of bicycle lane on at least one side of the road. The "bicycle" tag typically appears to mean that bicycles are *permitted*, but it may not mean that there is a dedicated bicycle lane.

Appendix C

Computer Environment

C.1 Machine Learning workstation

This workstation was used for all machine learning activities. TensorFlow 2.3 GPU acceleration was enabled via NVIDA cuDNN and CUDA drivers.

- Windows 10
- Anaconda 3
- Intel(R) Core(TM) i9-9900K CPU @ 3.60GHz
- 32GB RAM
- NVIDIA GeForce RTX 2080 Ti (12288MB dedicated, 16341MB shared
- 1TB SSD

Details of the Anaconda environment, including the packages and versions used, can be found in the GitHub repository in the "environment.yml" file.

C.2 Nominatim server

This Linux server ran on a separate machine, as a virtual host. Its sole purpose was to run a Nominatim server for the reverse-geocoding and geocoding services required by some Jupyter notebooks.

- Ubuntu 20.04 running on Windows 10 under Hyper-V
- AMD Ryzen 7 3700X 8-core processor 3.6GHz
- 16GB (32 GB for windows host)
- 100MB SSD

The server was loaded with OpenStreetMap data for Australia sourced from the GeoFabrik website [42] according to the Nominatim installation instructions [51].

Bibliography

[1] I.-M. Lee, E. J. Shiroma, F. Lobelo, P. Puska, S. N. Blair, and P. T. Katzmarzyk, "Effect of physical inactivity on major non-communicable diseases worldwide: an analysis of burden of disease and life expectancy," *The Lancet*, vol. 380, no. 9838, pp. 219–229, 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140673612610319 1, 3

[2] A. Rabl and A. de Nazelle, "Benefits of shift from car to active transport," *Transport Policy*, vol. 19, no. 1, pp. 121–131, 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0967070X11001119 1, 3

[3] "Active transport - walking and cycling," Sep 2019. [Online]. Available: https://www.infrastructure.gov.au/infrastructure/pab/active_transport/index.aspx 1, 3

[4] "Walking and cycling," Dec 2020. [Online]. Available: https://transport.vic.gov.au/getting-around/walking-and-cycling 1, 3

[5] M. Taylor and S. Thompson, "An analysis of active transport in melbourne: Baseline activity for assessment of low carbon mobility interventions," *Urban Policy and Research*, vol. 37, no. 1, pp. 62–81, 2019. [Online]. Available: https://doi.org/10.1080/08111146.2018.1437031 1, 3

[6] O. Wilson, N. Vairo, M. Bopp, D. Sims, K. Dutt, and B. Pinkos, "Best practices for promoting cycling amongst university students and employees," *Journal of Transport & Health*, vol. 9, pp. 234–243, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214140517309222 1, 3

[7] M. S. Klobucar and J. D. Fricker, "Network evaluation tool to improve real and perceived bicycle safety," *Transportation Research Record*, vol. 2031, no. 1, pp. 25–33, 2007. [Online]. Available: https://doi.org/10.3141/2031-04 1, 4, 23

[8] K. Teschke, M. A. Harris, C. C. Reynolds, M. Winters, S. Babul, M. Chipman, M. D. Cusimano, J. R. Brubacher, G. Hunte, S. M. Friedman, M. Monro, H. Shen, L. Vernich, and P. A. Cripton, "Route infrastructure and the risk of injuries to bicyclists: A case-crossover study," *American Journal of Public Health*, vol. 102, no. 12, pp. 2336–2343, 2012, pMID: 23078480. [Online]. Available: https://doi.org/10.2105/AJPH.2012.300762 1, 4

[9] "Principal bicycle network (pbn)." [Online]. Available: https://vicroadsopendata-vicroadsmaps.opendata.arcgis.com/datasets/39588a362a4d4336a3af534b128994ff_0 1, 7

[10] "Bicycle and walking route maps," Aug 2021. [Online]. Available: https://www.vicroads.vic.gov.au/traffic-and-road-use/cycling/bicycle-route-maps 1

[11] P. Schepers, E. Fishman, R. Beelen, E. Heinen, W. Wijnen, and J. Parkin, "The mortality impact of bicycle paths and lanes related to physical activity, air pollution exposure and road safety," *Journal of Transport & Health*, vol. 2, no. 4, pp. 460–473, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214140515006842 3

[12] F. A. Malik, L. Dala, and K. Busawon, "Deep neural network-based hybrid modelling for development of the cyclist infrastructure safety model," *Neural Computing and Applications*, Mar 2021. [Online]. Available: https://doi.org/10.1007/s00521-021-05857-3 4

[13] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015. [Online]. Available: https://doi.org/10.1038/nature14539 5

[14] M. Lofqvist and J. Cano, "Accelerating deep learning applications in space," 07 2020. ix, 5

[15] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 265–283. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi 5

[16] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga,

S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," 2016. 5

[17] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," 2019. 5

[18] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015. [Online]. Available: http://arxiv.org/abs/1506.01497 5

[19] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 21–37. 5

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 5

[21] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 5

[22] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 5

[23] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, and Q. Tian, "Centernet: Keypoint triplets for object detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019. 5

[24] H. Yu, C. Chen, X. Du, Y. Li, A. Rashwan, L. Hou, P. Jin, F. Yang, F. Liu, J. Kim, and J. Li, "TensorFlow Model Garden," https://github.com/tensorflow/models, 2020. 5, 6

[25] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision –*

ECCV 2014, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Cham: Springer International Publishing, 2014, pp. 740–755. 5

[26] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010. 5

[27] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 6

[28] B. Zhou, H. Zhao, X. Puig, T. Xiao, S. Fidler, A. Barriuso, and A. Torralba, "Semantic understanding of scenes through the ade20k dataset," *International Journal of Computer Vision*, vol. 127, no. 3, pp. 302–321, Mar 2019. [Online]. Available: https://doi.org/10.1007/s11263-018-1140-0 6

[29] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 834–848, 2018. 6

[30] A. Campbell, A. Both, and Q. C. Sun, "Detecting and mapping traffic signs from google street view images using deep learning and gis," *Computers, Environment and Urban Systems*, vol. 77, p. 101350, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0198971519300870 6, 13

[31] W. Zhang, C. Witharana, W. Li, C. Zhang, X. Li, and J. Parent, "Using deep learning to identify utility poles with crossarms and estimate their locations from google street view images," *Sensors*, vol. 18, no. 8, 2018. [Online]. Available: https://www.mdpi.com/1424-8220/18/8/2484 6

[32] "Supplement to australian standard as 1742.9:2000," 2015. [Online]. Available: https://www.vicroads.vic.gov.au/-/media/files/technical-documents-new/traffic-engineering-manual-v2/tem-vol-2-part-29--as17429-bicycle-facilities.ashx 6, 9, 10

[33] P. Li, Y. Zang, C. Wang, J. Li, M. Cheng, L. Luo, and Y. Yu, "Road network extraction via deep learning and line integral convolution," in *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, July 2016, pp. 1599–1602. 6

[34] H. Ning, X. Ye, Z. Chen, T. Liu, and T. Cao, "Sidewalk extraction using aerial and street view images," *Environment and Planning B: Urban Analytics and City Science*, vol. 0, no. 0, p. 2399808321995817, 0. [Online]. Available: https://doi.org/10.1177/2399808321995817 7

[35] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986. 7, 24

[36] R. O. Duda and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures," *Commun. ACM*, vol. 15, no. 1, p. 1115, Jan. 1972. [Online]. Available: https://doi.org/10.1145/361237.361242 7, 24

[37] K. Bapat, "Hough transform using opencv," May 2021. [Online]. Available: https://learnopencv.com/hough-transform-with-opencv-c-python/ 7

[38] R. S. Mamidala, U. Uthkota, M. B. Shankar, A. J. Antony, and A. V. Narasimhadhan, "Dynamic approach for lane detection using google street view and cnn," in *TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON)*, 2019, pp. 2454–2459. 7, 47

[39] L. Rita, "Become a better cyclist with deep learning: Using yolov5 to identify cyclists' risk factors in london," Aug 2020. [Online]. Available: https://towardsdatascience.com/become-a-better-cyclist-321a209d78d8 7

[40] M. Haklay and P. Weber, "Openstreetmap: User-generated street maps," *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, Oct 2008. 8

[41] "Google maps platform - bicycle layer." [Online]. Available: https://developers.google.com/maps/documentation/javascript/examples/layer-bicycling 8

[42] "Openstreetmap data extracts." [Online]. Available: https://download.geofabrik.de/ 10, 72

[43] "Osmium tool." [Online]. Available: https://osmcode.org/osmium-tool/ 10

[44] "Google street view static api usage and billing." [Online]. Available: https://developers.google.com/maps/documentation/streetview/usage-and-billing 11

[45] Tzutalin, "tzutalin/labelimg: labelimg is a graphical image annotation tool and label object bounding boxes in images." [Online]. Available: https://github.com/tzutalin/labelImg 12

[46] J. Boaz, "Melbourne passes buenos aires' world record for time spent in lockdown," Oct 2021. [Online]. Available: https://www.abc.net.au/news/2021-10-03/melbourne-longest-lockdown/100510710 19, 32, 37

[47] A. News, "The 5km travel limit returns. here's where that gets you - and the reasons you can go further," Jul 2021. [Online]. Available: https://www.abc.net.au/news/2021-07-15/whats-within-5km-your-home-victoria-interactive-tool-covid/100297252 19, 32, 37

[48] K. Sadekar, "Understanding lens distortion: Learnopencv," May 2021. [Online]. Available: https://learnopencv.com/understanding-lens-distortion/ 23

[49] Y. Li, L. Chen, H. Huang, X. Li, W. Xu, L. Zheng, and J. Huang, "Nighttime lane markings recognition based on canny detection and hough transform," in *2016 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, 2016, pp. 411–415. 24

[50] Y. Chai, S. J. Wei, and X. C. Li, "The multi-scale hough transform lane detection method based on the algorithm of otsu and canny," in *Materials Science and Intelligent Technologies Applications*, ser. Advanced Materials Research, vol. 1042. Trans Tech Publications Ltd, 11 2014, pp. 126–130. 24

[51] "Nominatimn basic installation." [Online]. Available: https://nominatim.org/release-docs/latest/admin/Installation/ 72