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 Masters of Data Science

Tyler Saxton
School of Computer Science and Information Technology
Science, Engineering, and Technology Portfolio,
Royal Melbourne Institute of Technology
Melbourne, Victoria, Australia

October 10, 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 Dhirendra Singh.

Tyler Saxton
School of Computer Science and Information Technology
Royal Melbourne Institute of Technology
October 10, 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, Laura Pritchard, and Dr. Curtis Saxton, thank you for your encouragement, and your assistance with the editing process.

To-Do

- Revisit abstract to update the scope of results based on any other suburbs tested
- Repetitive language between Summary and Abstract "this thesis presents"
- I'm sure I had a better paper on road boundary detection, find it!
- Thesis examples have "Contribution" and "Organisation" sections
- Insert example GSV detection image
- Picture of some false positive examples
- Picture of the mask
- I have not explained detection confidence threshold
- Example papers using Canny-Hough
- How well do the summary and abstract tie to all four research questions?

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 local 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 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 local 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 lanes were 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 local 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

[Criteria 15: clear research questions/aims/hypotheses] ⇐

[Criteria 15: background knowledge] ⇐

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 identify and 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 it does not appear to be up-to-date. 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 investigate whether it is possible 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" machine learning model to detect locations where bicycle lanes are found. If a baseline map of bicycle lanes can be built in 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" machine learning model be used to identify on-road bicycle lanes in street scene images sourced from Google Street View?
- RQ2: Can the model then be used to detect and map bicycle lane routes across all streets in a local area with Google Street View coverage?
- RQ3: Can a similar process be applied 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 be re-used to visually survey other details about the infrastructure?

Chapter 2

Literature Review

[Criteria 15: literature review places research in context] ⇐

[Criteria 15: background knowledge] ⇐

2.1 Motivation

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 data 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:

- Presence and type of bicycle lane (dedicated, paved shoulder, none)
- Width of bicycle lane
- Presence of on-street parking
- Downhill or uphill grades
- Volume, speed, and vehicle type profile of motor vehicle traffic
- Quality of road surface (including drainage, tram tracks, etc.)
- Lighting

- 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 Applications of 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 where the objects of interest appear in each image, to an acceptable level of performance.

Tensorflow [14] [15] and PyTorch [16] 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:

- Ren et al., 2015 [17] R-CNN
- Liu et al., 2016 [18] SSD
- He et al., 2016 [19] ResNet
- Redmon et al., 2016 [20] YOLO (and subsequent variants)
- Sandler et al., 2018 [21] MobileNetV2
- Duan et al, 2019 [22] CenterNet

The Tensorflow 2 Model Garden [23] 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" [24] [25].

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 [26] and Zhou et al., 2019 [27] 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 [28] 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 [29] 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. Zhang et al., 2018 [30] 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 200m intervals [31]. Therefore, a Deep Learning model could be trained to detect these markings, similar to how Campbell et al., 2019 [29] trained a model to detect road signs, and Zhang et al., 2018 [30] trained a model to detect utility poles, using pre-trained models from the TensorFlow 2 Model Garden [23] as a starting

point.

The use of satellite imagery and aerial photographs were also considered. Li et al., 2016 [32] 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 wide 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.

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 [33] 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. The road markings were not visible.

Aside from formal bicycle lanes, cyclist safety can also be improved by an informal wide 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 [34] and a Hough transformation [35]. The OpenCV library provides a frequently used implementation [36]. Where 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 [37] 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 [38] 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 complimentary approach that might serve to help policy makers to identify where there is demand for better infrastructure.

2.3 Available 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 excludes roads where the cyclist may ride on a less formal paved shoulder. During the course of the research, it was found that some existing bicycle lanes are still marked as "planned", or not listed at all, and in some country areas the routes appeared to be paved shoulders rather than standard bicycle routes. 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" [39] is a source of crowd-sourced map data. It provides detailed information about road networks worldwide, and the attributes of each road segment. It supports "cycleway" tags where contributors can mark not just where a bicycle lane is, but also 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 crowd-sourced, the quality and availability of the data may vary by location.

"Google Maps" provides a "bicycle layer". 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.

Other services such as "Strava" and "Trailforks.com" hold data that cyclists have recorded about their rides. These may be useful to assess the popularity of routes, and perhaps infer where upgrades would be welcomed, or where there might be an existing route that should be checked and added to a dataset.

Chapter 3

Methods

[Criteria 15: clear and accurate description of methods] ⇐

[Criteria 15: sufficient detail to allow reproduction of results] ⇐

[Criteria 15: awareness and critical evaluation of alternatives] ⇐

This section describes the methodology that was used to address each of the research questions.

[Comment on where to find github repository of code, where to find original source dashcam videos and GSV training image lists] ⇐

For detailed instructions on how to reproduce the results using the source code and data provided, please refer to the accompanying appendices.

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 [31] 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.

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.

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

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 Gathering 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 [31]. Therefore, intersections along known bicycle lane routes are the locations where example images are most likely to be found.

Campbell et al., 2019 [29] successfully trained a deep learning model to recognise "Stop" and "Give Way" signs with 500 example images of each. Therefore, an initial dataset of 500 bicycle lane marking example images was gathered, with an option to gather more images later, if necessary.

A Jupyter Notebook was created to allow an operator to efficiently review Google Street View images from intersections along known bicycle lane, and record which image files included the required examples. These images could then be "labelled", to record a precise bounding box around each bicycle lane marking in a format suitable for the TensorFlow 2 framework.

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 detail in section 2.3. The "Principal Bicycle Network" dataset was chosen, partially because it is the incumbent official dataset

for the local area, and partially due to its age. If it was last updated several years ago, then there is a much lower risk of attempting to sample an intersection, only to find that the bicycle lane was constructed too recently to appear in the available Google Street View data. However, the approach of using OpenStreetMap as a source of "known" bicycle lane routes would have the advantage that it generalises to other jurisdictions, and the process of using it ultimately proved to be less complex. Tools were created to generate lists of candidate intersections to sample in Google Street View using *both* approaches. For instructions on how to operate the tools, please refer to Appendix **??** and Appendix **??**.

Working from an XML extract of the OpenStreetMap database is the simplest approach. Data can be downloaded for an individual country via `https://download.geofabik.de` and then sliced into a smaller area, if necessary, using the "osmium" command line tool from `https://osmcode.org/osmium-tool/`. For more background on concepts in the OpenStreetMap data, please refer to Appendix A. Briefly, an OpenStreetMap XML extract file contains "ways", which usually represent road segments, and "nodes", which represent strings of coordinates that make up the "ways" and thereby describe the shape of the "way" on a map. A list of candidate intersections along "known" bicycle lane routes can be extracted from OpenStreetMap XML data, as follows:

- Use the "osmium" tool to reduce an OpenStreetMap database file for a country down to the required area. The tool can either reduce the map to a required bounding box, or it can be asked for data about an arbitrary shape, based on official geographic boundaries for a town or Local Government Area specified in a geojson file.
- Load the OpenStreetMap XML data into memory. Find all "way" objects that include a "cycleway" tag.
- For each "way" with a "cycleway" tag, check every "node" in the "way". If a node is any other "way" where the tags indicate that it is a road, and the "way" has a different name, then the "node" is an intersection. Include the "node" ID and its coordinates in the list of candidate intersections to sample.

Working from the "Principal Bicycle Network" dataset is more complicated, because although it describes the shape of each bicycle lane route in geojson format, it does not have any information about intersections. The data must be matched to OpenStreetMap to list the intersections along each route. Another significant limitation is that it does not always list the name of the road for each bicycle lane route, and it never lists the town or suburb. To overcome these limitations, the following approach was taken:

- Explode each "Principal Bicycle Network" dataset record into a list of coordinates.
- For each coordinate, use a "Nominatim" server instance to determine the names of the road and its town or suburb, via a "reverse-geocoding" API call.
- For each coordinate and its named road and town orsuburb, use an OpenStreetMap XML extract of the area to identify "ways" with a matching name, and then look up all ways that intersect with it, as per the OpenStreetMap method. Use a bounding box around the original route coordinates from the "Principal Bicycle Network" dataset to exclude streets in the OpenStreetMap data that just happen to have the same name, in another town. E.g. "Main Street" might occur in many towns across the state.

A local instance of "Nominatim" was installed and used for this purpose, because the terms of use for public instances discourage bulk operations, and limit them to one per second. The local instance was configured on a machine as per Appendix B.1, following the official instructions [40]. The instance was loaded with data for Australia only, and used a solid-state drive for storage.

Once intersections were identified, it proved to be helpful to also calculate an approximate heading along the bicycle lane route at each intersection, to ensure that Google Street View images could be requested with the camera facing in a sensible direction. The Python "geographiclib" library can calculate a heading from one point to the next. Therefore, a heading was calculated from the previous "node" on the way to the intersection "node", and from intersection "node" to the two.

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, and beyond that, further volume discounts can be negotiated with Google's sales team [41].

The Python "`google_streetview`" library was used to facilitate access to this Google Street View API from Python code. In order to minimise costs, whenever an image was downloaded via the API, a copy of the image and its associated metadata was cached locally at a path that depended on the request parameters. If a request was made for the same image at a later date, it would be retrieve from disk instead of issuing a further request to Google, to minimise costs.

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 was created to:

1. Randomly select a sample location and heading from a CSV of candidate intersections anywhere in the State of Victoria.
2. Download four Google Street View images at 0, 90, 180, and 270 degrees relative to the desired heading, with a 90 degree field of view for each image, and the camera angled 20 degrees towards the ground to focus on any nearby road markings.
3. Allow the operator to quickly record which of the four images they see a bicycle lane marking in (if any) to a CSV file listing images ready to be labelled and included in the dataset.
4. Repeat.

It was found that the best location to look for a bicycle lane marking was not always right in the middle of an intersection, especially for large intersections. Therefore, the option was added to allow the operator to "browse" a desired number of metres before or after the intersection, along the route heading, to find a good image. If a bicycle lane marking appeared at the intersection, typically it could be found within 20-30m, except for major intersections where bicycle lanes may be terminated early by turning lanes.

Generally, an image was accepted for inclusion in the dataset if the bicycle lane marking was clear and unambiguous without needing to take cues from the overall scene context. If the symbol was so far away that it looked like a white blob on the road, and it could only be understood to be a bicycle lane marking based on its position relative to lane markings, then the image was not included, though the operator had the option of moving the camera closer for a better image.

The Jupyter Notebook allowed multiple candidate intersections to be assessed per minute, and an initial set of 500 images was collected over the course of approximately 4 hours.

Please see Appendix **??** for detailed instructions on how to use the supplied Jupyter Notebook to download Google Street View images for sample candidate intersections, and flag matching images in an output CSV file for inclusion in the labelled dataset.

3.1.1.3 Labelling the dataset

Once a suitable number of Google Street View images containing bicycle lane markings were collected, they were copied to a folder and then labelled using the open-source tool "labelImg" [42]. Please refer to the tool's official documentation for installation and usage instructions. The output of the tool 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 B.1. This task could also be performed on cloud-based infrastructure such as Google Collab, if required. For general directions on how to set up a local computer to enable TensorFlow machine learning that takes advantage of GPU acceleration, please search online for the latest tutorials and guides, as this may be subject to change over time. Appendix **??** provides instructions on how to verify that GPU acceleration is enabled.

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 "testing" datasets according to an 80:20 ratio, using a "bash" shell script.

A Jupyter Notebook was used to set up the experiment, by transforming the labelled image datasets into TensorFlow records, downloading and configuring a pre-trained model from the TensorFlow 2 Model Garden, and creating scripts to train and evaluate the model.

The performance of each model depended on three key adjustable factors:

- The size of the dataset available for training and validation.
- The number of epochs spent training, where an "epoch" is a training iteration where a complete pass is made over the training dataset.
- The threshold confidence score, from 0% to 100%, at which the model assumes 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 [29], with the option to extend it later if necessary. In practice, it was

found that the initial size was sufficient to obtain extra results for detections from Google Street View images, and no further training images were required.

During the training process, the TensorFlow 2 training script showed "loss" performance metrics every 100 epochs. The performance reported by the training script gradually improved as more training was conducted over the dataset. Every few thousand epochs, training was paused, and the TensorFlow 2 evaluation script was run to test the performance of the model against the independent testing dataset that had been withheld from the training process. When training reached the point that further epochs did not significantly improve performance in either the training or the evaluation, training was halted.

As part of the evaluation process, image files were written to disk, containing the original image with an overlay where the bounding boxes and confidence score of any detections were drawn. These were examined to determine how many false positives and false negatives there were, and the confidence scores when the model returned any false result. The threshold confidence score was tuned in response to these findings, to find as many true positives as possible without significant numbers of false positives that might cause bicycle lane routes to be "detected" where they do not really exist.

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 advertised "Speed" and "COCO mAP" (mean Average Precision) metrics. 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 speed.

The TensorFlow 2 Model Garden provided an efficient way to test multiple models via a single framework. However, 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. However a different model (e.g. YOLOv3) or a different framework (e.g. PyTorch) could be substituted if required.

Please refer to Appendix **??** for detailed instructions on how to reproduce the training process.

3.2 RQ2: Building a map of bicycle lane routes from Google Street View images in an 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 marker 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 and draw them on a map.

3.2.1 Sampling strategy for Google Street View images

In section 3.1.1.2, it was noted that Google currently advertises a minimum cost of $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. During the sampling process, it was found that the Google Street View API would only return distinct images every 10 metres. Two sampling strategies were considered:

- Generating a list of sample points every 10 metres along every street in the area.
- Generating a list of sample points within a configurable distance of each intersection in the area, at 10 metre intervals.

Depending on the density of intersections in an area, limiting the samples to the immediate area around intersections could result in a significant cost reduction. The "intersection" strategy was chosen, with the option to switch to a strategy of sampling every 10 metres if the results showed that too many bicycle lanes were being missed in situations where they might have been detected with more samples.

An OpenStreetMap XML extract file for the area was loaded into memory and used to iterate over all streets and intersections in the area, resulting in a CSV batch file of sample locations to be downloaded from Google Street View and processed by the model. The size of the batch file was inspected to assess potential cost before proceeding with the download of images from Google Street View.

When loading and interpreting the OpenStreetMap XML extracts, it was important to understand that while a "way" object in the data typically represents a road segment, it may

instead represent the boundary of a reserve, a natural feature such as a coastline or creek, or a walking trail. A "way" is used to represent any line that is drawn on the map, in general. To avoid unnecessary requests to the Google Street View API for off-street locations, it was necessary to inspect the "tags" associated with each "way" to isolate actual road segments.

Another important consideration was how to handle roads that form the boundary of the area being surveyed. If the "osmium" tool is used to create an OpenStreetMap XML extract for a precise area according to the shape in a geojson file describing an official boundary, then the resulting extract may exclude intersections where the intersecting road exists entirely outside the boundary. This problem was addressed by allowing the operator to load a second OpenStreetMap XML extract file to cover a 200 metre margin around the area being surveyed. Streets in the margin would not be considered in the survey for possible bicycle lane routes, but they provided supplementary used only to identify intersections and ensure that none were missed.

Once the sample process has been run, the output is a CSV to document the coordinates of each point that needs to be downloaded from Google Street View and examined by the detection model. The CSV file includes additional columns to record the OpenStreetMap "way" ID and "node" ID associated with each point, so that any results can be linked back to the map. Other fields such as the street name were also included for traceability.

Please see Appendix **??** for detailed instructions on how to operate the tool that was used to generate a list of sample locations in a survey area.

3.2.2 Batch processing of Google Street View images

Given a CSV file listing sample locations to include in the survey of the area, a batch process works through the list, downloading any Google Street View images that have not already been cached locally. Then, the selected detection model is called to process each image in turn. If one or more bicycle lane markings are detected in an image, a copy of the image is saved to a folder of "hits", with the bounding box and confidence score included in the image as an overlay, to show where the detection was found. A record is also inserted into a CSV file of detection locations. If no bicycle lane marking is found in the image, a copy is instead placed in a separate "miss" folder. Partitioning the result images into separate "hits" and "misses" folders allows the operator to quickly browse the images to check for any false positives or false negatives.

Please see Appendix **??** for detailed instructions on how to operate the tools to download and

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

cache Google Street View images for a list of sample locations, then process them with the model.

3.2.3 Converting detection points into bicycle lane routes on a map

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

Google Street View image samples were taken 0, 90, 180, and 270 degrees from the assumed heading at each point, to provide 360 degree coverage. This allows detection of bicycle lane markings not just in the the forward and rear views, but also close alongside the camera vehicle, to improve the chances that an image will be found where the marking is clearly visible. The detection model has been trained to detect markings in any orientation, and many of the samples will be taken at intersections. Therefore, when drawing routes, proper consideration must be made about whether a detection belongs to the route being inspected, or an intersecting street.

Routes were inferred from the individual 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.

A continuous road may be divided up into multiple "way" segments in the OpenStreetMap data 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 written to a geojson file as a "LineString Feature", following the coordinates of all "nodes" in the original OpenStreetMap "way", from start to finish.

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.

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.

The OpenStreetMap XML extract for the survey area can be used to generate an equivalent geojson file of bicycle lane routes according to OpenStreetMap. The data is filtered down to "ways" that are roads and have a "cycleway" tag. Then a "LineString Feature" is written to an output geojson for each cycleway.

The detected routes geojson file and the OpenStreetMap-derived geojson file can be drawn on maps side-by-side for visual comparison, and the differences can be measured at a high level by comparing the total length of the routes in each file.

To drill down further into the detail of the differences, multiple geojson files were created:

1. A geojson file with all bicycle lanes from the detection model
2. A geojson file where a bicycle lane was detected but was not recorded in OpenStreetMap
3. A geojson file where a bicycle lane was recorded in OpenStreetMap but not detected
4. A geojson file where both the detection model and OpenStreetMap agree that there is a bicycle lane

Items 2-4 can be drawn as layers on a single map, each with their own color, to highlight where the bicycle lanes are, and where the sources agree and disagree. By measuring the distances in each geojson file, we can quantify how much agreement there is between the two sources, and how many metres worth of bicycle lane routes are in dispute.

Where there is a disagreement between the output of the detection model and what has been recorded in OpenStreetMap, the original images or footage can be reviewed to establish the truth, and whether the detection model was producing reasonable results for its input.

The "Principal Bicycle Network dataset" is available in a geojson format. A quick visual comparison is therefore possible, by drawing it on a map side-by-side with the geojson file of detected bicycle lane routes. To quantify the differences in more detail:

1. For each point in the "Principal Bicycle Network" dataset geojson file, find the closest point in the OpenStreetMap XML extract and its "node" ID. Exclude any points outside the bounding box of the survey area.

2. Use the coordinates of the matching "node" from the OpenStreetMap data, instead of the original coordinates from the "Principal Bicycle Network" dataset, to ensure that the sources are "aligned" on a common understanding of where the road network is, and eliminate tiny differences.

3. Produce four output geojson files to compare the "aligned" "Principal Bicycle Network" data to the detected bicycle lanes, following the same basic process that was used for the OpenStreetMap comparison.

3.3 RQ3: Applying the process to dash camera footage

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 bad luck with 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 detecded 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 passenger sedan in the manufacturer's recommended location behind the rear-view mirror, at a cost of $529 AUD. This model was selected because for each one minute of footage recorded in an MP4 video file, it also records an NMEA file with latitude, longitude, heading, and altitude at one second intervals. The NMEA metadata allows each frame in the footage to be linked to a location, by correlating the frame number to nearest NMEA measurements, and extrapolating readings across the 1 second interval as required.

Video footage was recorded at 60 frames per second, at 1920x1080 resolution, a significant improvement on the 640x640 resolution of Google Street View images, and the 10 metre intervals. 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 exact specifications such as focal length were not available from the manufacturer's website.

3.3.2 Gathering and processing dash camera footage

On the 3rd of October, 2021, Melbourne set the record for the longest "COVID lockdown" in the world [43]. Severe restrictions were placed on residents being more than 5km from home, gradually easing to 10km and then 15km [44]. 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 for processing via MicroSD card. Then a batch process was set up in Jupyter Notebook to process each pair of files in a directory.

For each pair of files, the NMEA file was loaded into memory, then the video file was split into a series of images via the "OpenCV" library. Reviewing the footage, it was decided that it was only necessary to process every 5th frame in order to get a clear view of every bicycle lane marking, effectively reducing the footage from 60 frames per second to 12 frames per second. The latitude, longitude, altitude, and heading for each frame was calculated or extrapolated from the NMEA records. Each frame was recorded to disk as a .png file for traceability, with a filename consisting of the original video filename, and a four-digit frame number. Browsing these files in alphabetical order preserved their original chronological sequence. For each frame, an entry was added to an output "batch" CSV file to record the path to the image file to be processed, and its associated location metadata.

The batch CSV file was processed by the detection model to yield another CSV file with detection results, as per the Google Street View process described in section 3.2. For images where at least one "BikeLaneMarker" was detected, a copy of the image with a green bounding box was written to a "hits" directory for manual inspection. For images where a "BikeLaneMarker" was not detected, a copy was written to the "miss" directory. Images in the "hits" and "miss" folders could be quickly inspected manually.

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, 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:
  - Short white dashes painted at intersections to advise drivers to "give way"
  - White arrows drawn on the road surface, such as turning lane arrows
  - Diagonal white stripes that were drawn on the road to represent traffic islands
  - Large white writing on the road, such as "KEEP CLEAR"
  - Pale road defects and anomalies
  - Reflections from the bonnet of the camera vehicle
  - Random objects or clouds well outside the boundary of the road

See figure 3.3 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 3.3: False Positive Examples

A decision was made to re-train the model with selected images from footage that was gathered in Frankston, Langwarrin, and Baxter, and to test on independent images gathered in Mount Eliza.

The footage from the Frankston/Langwarrin/Baxter training area was processed through the original Google Street View-trained model as-is, to see where that model was registering detections, whether they be true positives or false positives. A total of 342 images were selected for labelling so that they could be added to the existing Google Street View images in the dataset.

The original Google Street View training images had been labelled with a single class "BikeLaneMarker". The new training images selected from dash camera footage of the training area were labelled with additional classes, to encourage the model to think of them as something other than a bicycle lane marking:

- BikeLaneMarker
- GiveWayMarker
- ArrowMarker
- IslandMarker
- RoadWriting
- RoadDefect

Once these new images had been labelled, they were randomly split 80:20 and added to the Google Street View images in the original training and testing datasets. The models were then re-trained from scratch based on the combined training dataset, and tested against thew new combined test dataset.

To deal with the false positives that had triggered by reflections on the bonnet of the camera vehicle, and random objects on the side of the road, the process was updated to apply a mask to each image before passing it to the detection model. The mask was designed to exclude the bonnet of the vehicle and anything on the opposite side of the road, and focus attention on places in the frame where a bicycle lane marker might reasonably be encountered in the direction of travel. See figure 3.4.

The detection model that had been re-trained to include dash camera images from the training area of Frankston/Langwarrin/Baxter was then evaluated against the independent footage from the Mount Eliza area.

Figure 3.4: Detection mask to exclude car bonnet and right hand side of road.

3.3.4 Mapping dash camera detections and comparing to other sources

Processing a batch of images from the dash camera yields a CSV file with detection locations. The location coordinates in these results come from NMEA file, and they may not exactly match "node" coordinates from the OpenStreetMap XML data extract. To produce a map of bicycle lane routes detected from the dash camera footage, we first "align" the detection co-ordinates with "node" coordinates found in the OpenStreetMap data. This allows us to more easily compare routes and measure the differences.

For every detection location logged by the model, we find the nearest "way" in the OpenStreetMap XML extract that belongs to a named road. The Python "shapely" library helps us to perform this search as efficiently as possible, using bounding boxes for each way to avoid a brute force search of all points. Once we have found the nearest "way", we check every "node" in the way to record the nearest *intersection* "node", and the nearest "node" of *any type*. We consider there to be a "hit" at the coordinates of both nodes: the point on the OpenStreetMap route closest to the detection – which may a bicycle lane marking that occurs at intervals of up to 200 metres along the road [31] – and the closest intersection along the road.

Once the detection coordinates have been aligned to the OpenStreetMap coordinates, geojson files can be "drawn" and compared to other sources using the same techniques as for the Google Street View images. This process is based on the nearest intersection for each detection, but preserving the co-ordinates of the nearest "node" of any type in the output helps with traceability.

3.4 RQ4: Surveying other infrastructure details using dash camera footage

Once dash camera footage had been used to map bicycle lane routes in a survey area, some further work was conducted to demonstrate how the approach might be re-used to survey other details of interest. To demonstrate future potential, a process was created to map any paved shoulders on the side of the roads in the survey area that cyclists might ride on achieve physical separation from motor vehicles. 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 if it is not explicitly marked. We are interested in the edge of the road on the side of the road that traffic usually drives on. In this research, dash camera footage was captured in Australia, so we are interested in the left hand side. The following section therefore refers to the edge of the road on the side that traffic usually drives on as the "left".

3.4.1 Correcting 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 [45]. The camera is used to capture a range of images where a special calibration tool appears in frame. See figure 3.5 for some example images. An image of a "chessboard" pattern of known dimensions is held in front of the camera in different positions.

Figure 3.5: 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 its 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 3.6 for an example raw image from the dash camera, and a corresponding corrected version.

(a) Uncorrected image

(b) Corrected image

Figure 3.6: Example OpenCV camera calibration images

The most obvious sign that the correction has been applied in this example 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.

3.4.2 Detecting lane markings

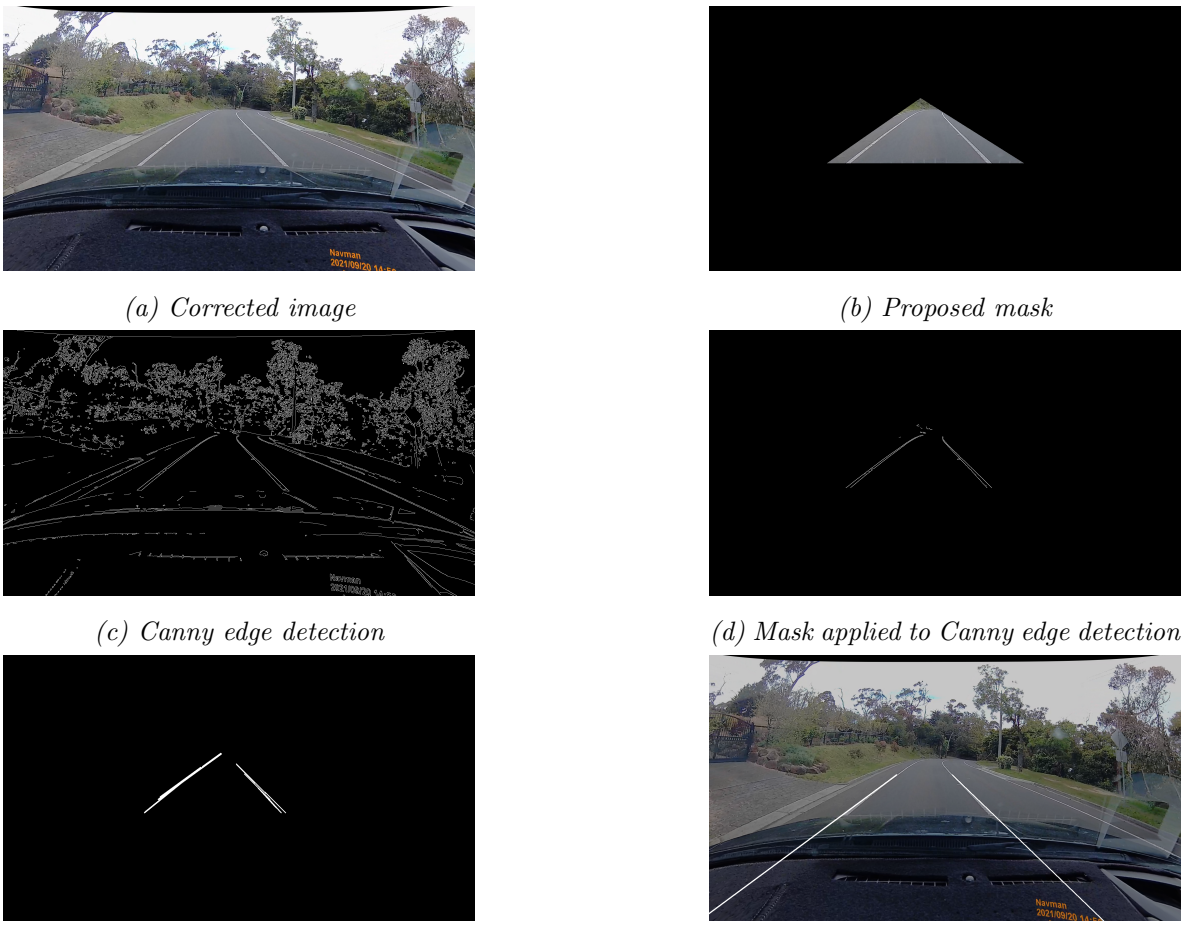
A common solution to the problem of detecting lane markings involves applying a combination of the Canny edge detection algorithm proposed by Canny, 1987 [34] and the Hough transformation proposed by Hough, 1972 [35]. This general approach has been followed in many papers such as Li et al, 2016 [46] or Chai et al, 2014 [47] and it is frequently used in capstone projects in the self-driving car domain. At its most basic, the approach does not involve any machine learning or deep learning techniques.

The Canny/Hough approach typically works as follows:

- Convert the image to greyscale
- Apply Canny edge detection to detect edges in an image [34]
- Apply a mask to exclude edges outside a triangle representing the area immediately in front of the vehicle
- Apply a Hough transform [35] to convert these edges into lines, each line having a slope and an intercept.

- Partition the lines into two groups: Lines that slope upwards from left to right, and lines that slope downwards from right to left.
- For each group of lines, take the average slope and intercept. These represent the detected lane boundaries.
- Use the slope and intercept of each detected line to overlay a line on the original image, extending from the bottom of the image to a suitable "horizon" somewhere around half-way to the top of the image. These straight lines representing the lane boundaries can be visualized by superimposing them over the original image.

Please see figure 3.7 for example images to demonstrate the process.



(a) Corrected image

(b) Proposed mask

(c) Canny edge detection

(d) Mask applied to Canny edge detection

(e) Hough transform

(f) Average lines, superimposed

Figure 3.7: 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 be the left boundary of the camera's own lane, which is also the right boundary of the paved shoulder. The other lane will be the left boundary of the paved shoulder, further to the edge of the road. To detect this third line, we apply a mask to the Canny edge detection image just to the left of the camera's own lane 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 3.8.

(a) Canny edge detection with previously detected lane

(b) Canny edges with new mask applied

(c) Image with all three detected lines

Figure 3.8: Example sequence of Canny/Hough operations to detect paved shoulder

If the Canny/Hough approach cannot find another line to the left of our the camera vehicle's 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 beside the road.

Please see figure 3.9 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, to highlight where the detected lines intersect the a height just above the bonnet of the car, and a height further into the distance, closer to the "horizon" of the mask.

Figure 3.9a shows a true positive match, where the boundaries of a paved shoulder have been correctly identified.

Figure 3.9b shows a situation where a concrete gutter has resulted in a very narrow area being detected. This potential false positive can be mitigated by requiring a minimum width for any detected area.

(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 3.9: Example paved shoulder detections

Figure 3.9c shows a false positive match, where "noise" from a park 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. However, 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 line to appear consistently 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 the left and right boundaries of a paved shoulder. This criteria was applied to the Frankston/Langwarrin/Baxter training area footage, and the resulting images with detection line overlays were reviewed in chronological order. The results were very inconsistent.

Reviewing the footage, it was found that 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 of gap between the boundaries towards the "horizon" at the top of the frame was relatively wide. There were relatively 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 much less consistent 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. Frames associated with a single stretch of road between two intersections would be assessed as a group, to determine whether a paved shoulder or bicycle lane exists along that stretch.
- Frames from locations within 30 metres of an intersection were excluded from consideration, because it is common for them to "disappear" at the approach to an intersection, and this disappearance should not counted as a missed detection.
- If the model fails to find both boundaries of a paved shoulder in more than 20% of the remaining frames along a stretch of road, then it is assumed that there isn't one.
- A horizontal line is drawn across the frame towards a "horizon" as per figure ***[REF]***. ⇐ 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 75 pixels, then the possible 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 50 pixels, then the boundary lines are considered to be too inconsistent. The stretch of road is assumed not to have a paved shoulder.

The thresholds described above were set based on the following process:

- Update the batch process to output a row for every frame, in chronological order
- Include the name of the road for traceability.
- Include the node IDs of the two nearest intersections, to trace which frames are part of the same group. The first node ID is the one that comes first alphabetically, not the one that is closest to the location associated with the frame. This is to avoid splitting a group of frames for a stretch of road into two groups depending on which intersection is closest.

- Filter out frames where the location is within 30m of either of the two nearest intersections
- Create summary statistic columns for each group for frames, and add them to the output CSV as additional columns:
  - Proportion of frames where boundary lines were not detected
  - Mean width of the potential paved shoulder, in pixels, at the horizontal "horizon" line
  - Standard deviation of the y-coordinate where the boundary lines intersect
  - Standard deviation of the x-coordinate where the boundary lines intersect
- Apply this process to the training area footage from Frankston/Langwarrin/Baxter to produce an output CSV.
- Load the output CSV file into an editor, and add a "truth" column to denote whether a paved shoulder really exists at each location, based on a review of the output frames with the detection line overlay.
- Filter the data to find thresholds in the summary statistics that appear to explain how the "truth" values are partitioned, as closely as possible.
- Create "prediction" column in the output CSV based on the chosen thresholds, and compare it to the "actual" value in the "truth" column.

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, take the distinct combinations of "way" ID and the "node" IDs of the two closest sections, where a paved shoulder was predicted. Find the "way" ID in the OpenStreetMap data, and "draw" a "LineString Feature" in a geojson file for all "nodes" between the two intersection nodes, including the intersection nodes themselves. The geojson file can 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. 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 [37]. It would be helpful to be able to work from a significantly large volume of dash camera images, after the current COVID restrictions have eased.

Chapter 4

Results and Discussion

⇐ *[Criteria 40: clear and complete presentation of results]*

⇐ *[Criteria 40: sufficient quantity of work]*

⇐ *[Criteria 40: appropriate intellectual level]*

⇐ *[Criteria 40: appropriate consideration of evidence in discussion]*

⇐ *[Criteria 40: uncertainty/error analysis]*

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

⇐ *[Table showing training loss at each 100 epochs for multiple models]*

⇐ *[Table showing evaluation mAP at 5k 10k 15k 20k 25k 30k for multiple models]*

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

4.2.1 Mount Eliza

⇐ *[Detected Map]*

⇐ *[OpenStreetMap Map]*

[PBN Map] ⇐

[Comparison of Detected vs. OpenStreetMap] ⇐

4.2.2 Another suburb

[Pick a suburb further in, residential like Carnegie? Check OSM first.] ⇐

4.3 RQ3: Applying the process to dash camera footage

[Comparison of Detected vs. OpenStreetMap] ⇐

4.4 RQ4: Surveying other infrastructure details using dash camera footage

[Produce map of paved shoulders, and discuss. What is true?] ⇐

4.5 Limitations

Discussion of limitations of the work

4.6 Opportunities for future research

Chapter 5

Conclusion

⇐ *[Criteria 15: conclusions are supported by the observations/results/calculations]*

⇐ *[Criteria 15: conclusions relate to the original research questions/aims/hypotheses]*

Appendix A

OpenStreetMap XML Concepts

Explain ways, nodes, tags, natural features, intersections...

Appendix B

Computer Environment

Two computers were used

B.1 Machine Learning workstation

Hyper-V

B.2 Nominatim server

Install instructions?

Appendix C

Process documentation

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

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

And the both the code and any key data collected are available on FigShare at:

`TBA`

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.

C.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 step through one cell at a time if preferred)

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

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

C.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 B.2.

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

C.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 a copy of the output file from "FigShare", or by using OpenStreetMap as the driver for sampling intersections along known bicycle lane routes, via notebook 03.

C.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 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.

C.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 C.1 for a screenshot of the GUI.

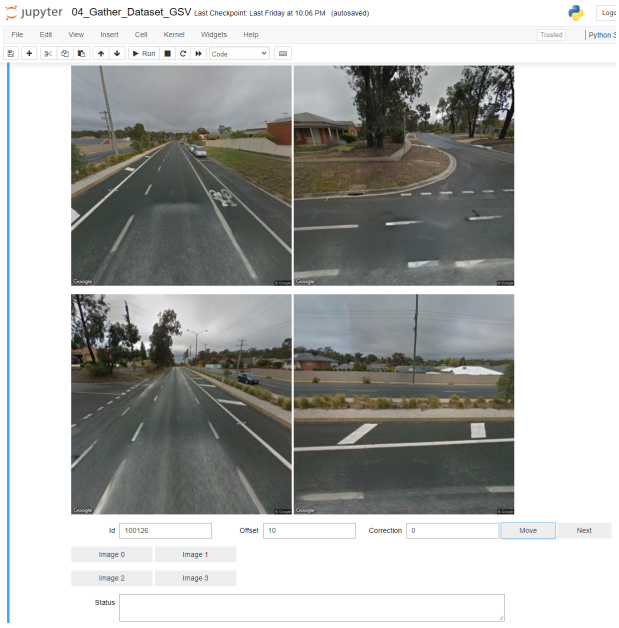


Figure C.1: GUI for gathering Google Street View image dataset

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 C.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.

C.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.

C.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.

C.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.

C.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?

C.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.

C.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.

C.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.

C.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.

C.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.

C.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.

C.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.

C.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.

C.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.

C.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.

C.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, or detecting hazardous road surface defects, etc.

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.

C.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.

C.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.

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

[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

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

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

[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

[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

[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

[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

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

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

[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

[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

[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

[14] 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

[15] 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. Wat-

tenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," 2016.

[16] 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.

[17] 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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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

[28] 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.

[29] 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

[30] 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

[31] "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

[32] 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.

[33] 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

[34] 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.

[35] 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

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

[37] 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.

[38] 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

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

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

[41] [Online]. Available: https://developers.google.com/maps/documentation/streetview/usage-and-billing

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

[43] 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

[44] 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

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

[46] 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.

[47] 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.