Traffic-Sign Recognition Using Deep Learning

Zhongbing Qin

A project report submitted to the Auckland University of Technology
in partial fulfillment of the requirements for the degree of
Master of Computer and Information Sciences (MCIS)

2020

School of Engineering, Computer & Mathematical Sciences

Abstract

Traffic-sign recognition (TSR) has been an essential part of driver-assistance systems, which is able to assist drivers in avoiding a vast number of potential hazards and improve the experience of driving. However, the TSR is a realistic task that is full of constraints, such as visual environment, physical damages, and partial occasions, etc. In order to deal with such constrains, convolutional neural networks (CNN) are widely used to extract the features of traffic signs and classify them into corresponding classes.

In this project, we initially created a benchmark (NZ Traffic Signs 3K) for the traffic-sign recognition in New Zealand. In order to determine which deep learning models are the most suitable one for the TSR task, we chose two kinds of models to perform with deep learning models: Faster R-CNN and YOLOv5. According to the scores of various metrics, we summarized the pros and cons of the picked models for the TSR task.

Keywords: Traffic signs, Faster R-CNN, YOLOv5, CNN, NZ Traffic Signs 3K

Table of Contents

List of Figures

List of Tables

Attestation of Authorship

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person (except where explicitly defined in the acknowledgments), nor material which to a substantial extent has been submitted for the award of any other degree or diploma of a university or other institution of higher learning.

Signature: Zhongbing Qin Date: 22 October 2020

Acknowledgment

First of all, I would like to appreciate the guidance of my supervisor Wei Qi Yan. Dr. Yan's guidance played an extremely pivotal role in the process of completing my final project. He not only gave me professional guidance but also provided a lot of advice on learning methods, which will surely benefit my future researches and life path.

Secondly, I am very grateful to my university, the Auckland University of Technology (AUT), for delivering a supportive and positive academic atmosphere while the Covid-19 is prevalent.

Finally, I sincerely thank my parents for their financial support, caring and understanding while finishing my postgraduate degree.

Zhongbing Qin

Auckland, New Zealand

October 2020

Chapter 1

Introduction

This chapter mainly consists of five sections. In the first section, we will introduce background information of traffic sign detection and the inspiration of this project. Next, the research questions will be set forth. Then, we will list our contributions of this project and explicate them on details. In the last two sections, the objectives and the format of this report will be elucidated, respectively.

1.1 Background and Motivation

Traffic scene understanding is an important topic in the field of computer vision and intelligent systems. Traffic signs effectively assist drivers in the process of driving and keep them drive more safely as they are designed to inform drivers of current road situations and potential hazards. These signs are normally rigid and simple shapes, such as circles, triangles and regular polygons, and with eye-catching colors (Zhu et al., 2016). Hence, traffic sign recognition has become more and more valuable for driver-assistance systems, highway maintenance and especially for self-driving vehicles (Mogelmose, Trivedi, & Moeslund, 2012).

Generally, there are two steps in a typical traffic sign recognition. The first one is to locate and get size information of traffic signs in natural scene images, which is known as traffic sign detection. The second step is to categorize detected traffic signs into the corresponding sub-classes, which is known as traffic sign classification, and this step is generally completed manually. Although traffic sign recognition has gained a plethora of popularity in driver assistant system, there are still numerous difficulties for identifying real-world traffic signs by using computer algorithms due to various size of targets (Zhu et al., 2016), color deterioration and partial occlusion (Yang, Luo, Xu, & Wu, 2015).

In order to deal with these obstacles, many approaches and algorithms have been proposed. In the past, traffic sign detection mainly relied on traditional object detection algorithms and the pipeline of traffic sign detection normally utilized hand-crafted features to extract region proposals, and then combined classifiers to filter out the negatives. Recently, deep learning methods are emerging, and various cutting-edge approaches have been widely applied into this area, such as deep convolutional networks (CNNs). CNNs have brought possibility of learning features from giant amount of data without preprocessing, which avoids the process of designing hand-crafted features and absorbs more generalized features (Zhang, Huang, Jin, & Li, 2017). Besides, CNN has been already set forth as an object classifier in machine learning which have been leveraged on traffic sign classification.

Two benchmarks are widely accepted to evaluate object detection performance, including PASCAL VOC (Everingham, Van Gool, Williams, Winn, & Zisserman, 2010) and ImageNet ILSVRC (Russakovsky et al., 2015). In these datasets, objects typically occupy a large proportion of each image, the size of bounding box is over 20% of the images, but visual objects of interest only occupy small fractions of corresponding images. For example, traffic signs are captured while driving and such small size of objects will have a huge impact on the performance yield by choosing algorithms. A typical traffic sign in a real-world image might be around 80×80 pixels, they are just approximately 0.2% of the image. Thus, it is essential to reconsider the evaluation performed benchmark for those tasks requiring classification and detection of small objects of interest.

In the development of traffic sign recognition, German traffic-sign detection and classification benchmarks brought in a vast majority of benefits for evaluation across various algorithms, which were not comparable until the release of the benchmarks. German Traffic Sign Detection Benchmark (GTSDB) (Stallkamp, Schlipsing, Salmen, & Igel, 2012) and German Traffic Sign Recognition Benchmark (GTSRB) (Stallkamp, Schlipsing, Salmen, & Igel, 2011) presented two public extensive and available datasets and nowadays there are several methods have achieved high accuracy rate based on these datasets. Besides, other datasets are also available in public recent years, such as LISA traffic sign dataset (LISATSD) (Mogelmose et al., 2012), Swedish Traffic Signs Dataset (STSD) (Larsson & Felsberg, 2011) and Chinese Traffic Sign Dataset (CTSD) (Yang et al., 2015). The GTSRB and GTSDB datasets are the most popular ones for recognizing traffic signs and many methods have achieved great success on them.

1.2 Research Questions

In order to effectively recognize real-world traffic signs, this report aims to utilize the state-of-the-art deep learning methods to detect and classify signs, and then evaluate their performance based on the comparison results. Based on the above purposes, the research questions are proposed as follows:

(1) *Which deep learning algorithms are suitable for recognizing various sizes of real-world traffic signs?*

(2) *What detection and classification algorithm systems will fit for our project?*

The fundamental idea of this project is to complete traffic sign recognition in New Zealand. However, due to the lack of customized dataset for recognizing New Zealand's traffic signs, it is necessary to create a partial dataset in this case. Several deep learning algorithms will be chosen to deal with this problem. The pros and cons of them will be presented based on the evaluation results.

1.3 Contribution

The contributions of this report are shown as follows:

- In order to effectively recognize New Zealand's traffic signs, we have created a new and realistic traffic-sign benchmark, which contains partial traffic sign classes because of physical and time limitations. The benchmark is composed of seven traffic sign categories and various sizes of real-world signs were captured. The distinction of this benchmark is that it covers number of small-size objects, which cannot be identified in off-the-shelf datasets. We call this benchmark NZ-Traffic-Signs 3K.

- We conducted an experiment for traffic sign recognition on the latest deep learning model (YOLOV5) and accomplish a comparison across the applied algorithms. The evaluation results illustrated the robustness, which will benefit the research of New Zealand traffic scene understanding.

1.4 Objectives of This Report

Overall, the ultimate objective of this report is to complete the customized traffic-sign recognition in New Zealand and figure out which state-of-art networks better fit into this project.

Firstly, so as to recognize traffic signs in NZ, the dataset generation process will be emphasized since the quality of original data will have a huge impact on the training results of networks and models.

Secondly, the tuning process of detectors and classifiers will be introduced, and we will adjust the neural networks from the following perspectives: (1) Experiments performed on different algorithms (Faster R-CNN and YOLOv5), (2) Validation and evaluation based on the chose models, (3) Parameter tuning process.

Finally, this report will summary the pros and cons across the applied algorithm systems and discover the best system for traffic-sign recognition according to the New Zealand traffic scene.

1.5 Structure of This Report

The rest of this report is structured as follows. In Chapter 2, we will conduct a comprehensive literature review to identify the current studies and technologies that are corresponding to object detection and classification. In Chapter 3, the methodology in this project will be introduced from three aspects, including the benchmark generation methods, experiment designs and evaluation metrics. In Chapter 4, the experiment results will be presented and illustrated. In Chapter 5, we will discuss and analysis the experiment results, and then figure out which algorithms achieve the best performance. Finally, we will make a summary of this project and clarify the future study direction in Chapter 6.

Chapter 2

Literature Review

The focus of this report is to leverage the cutting-edge algorithms to complete traffic-sign recognition. Therefore, we comprehensively reviewed the previous studies and identified the corresponding theories and algorithms over the past ten years.

2.1 Introduction

Traffic sign recognition (TSR) has benefited a large number of realistic applications, such as driver assistance system, autonomous vehicles, and intelligent mobile robots since they have delivered the current state of traffic signs into various systems. However, there are a few difficulties for computer to recognize traffic signs on the road, which are mainly from two aspects: One is relevant to the complex traffic scene (Wang, Ren, & Quan, 2013) on the road and another one is about unbalanced class frequencies in the datasets (Stallkamp et al., 2012).

As for the difficulty of real-world traffic scenes, though traffic signs are always well designed for drivers to easily read and recognize the signs during the driving process, including vivid colors, strong and big size words, as well as various specific shapes, it can be still a tricky task to design such features combining with contaminated conditions (Sermanet & LeCun, 2011). For example, the conditions can be bad illumination, small-size signs in scenes, partial occlusions, rotations and physical damages. All of these factors will have a huge impact on the performance of computer algorithms to recognize traffic signs.

In terms of the characteristics of the benchmarks, they usually have an uneven distribution of data categories. As we known, traffic signs have various types. For instance, the GTSRB includes 43 classes with the lowest frequency rate of 0.5% and the highest frequency rate of near 6% across all classes (Mao et al., 2016).

Before the wide acceptation of convolutional neural networks, the dominant approaches for identifying traffic signs focused on several feature extraction methods and machine learning algorithms. The technique, Histogram Oriented Gradients (HOG) (Dalal & Triggs, 2005), was initially used to detect pedestrians in traffic scenes and the gradients in a color image were calculated along with normalized and weighted histograms. The feature transform technique was utilized to classify window sliding. Several machine learning algorithms were leveraged for traffic-sign classification,

including support vector machines, linear discriminant analysis ensemble (Malik, Khurshid, & Ahmad, 2007) and random forest (Zaklouta & Stanciulescu, 2014), etc.

2.2 Traditional Traffic Sign Detection Solutions

2.2.1 Using Feature Extraction Methods

The initial methods for object detection mainly depend on feature extraction methods. People usually took color and shape features into consideration to achieve traffic-sign detection and classification tasks.

In terms of color features, the images were transformed to other color spaces like HSV (Hue, Saturation, Values) instead of using RGB (Red, Green, Blue). Wang *et al.* (2013) pointed out that the computer algorithms based on RGB color spaces could limit the performance of detection traffic signs due to different illuminant conditions. Besides, Li *et al.* (2014) also proposed a color probability model based on Ohta space to compute the maps of probability for each color belonging to traffic signs (Yang & Wu, 2014).

With regard to shape features, traffic signs have various geometries, such as circular, rectangular, triangular or polygonal. People extracted contour lines by Hough transforms and radial symmetry, etc. The circular traffic signs would deform by shooting angle or other external force. In order to tackle this issue, Wang et al. (2014) proposed an ellipse-detection method in their article (G. Wang, Ren, Wu, Zhao, & Jiang, 2014). Moreover, Liang et al. (2013) designed a list of templets for each traffic-sign class to match shape (Liang, Yuan, Hu, Li, & Liu, 2013). The HOG as one of the most widely used features also benefited a lot for the traffic-sign feature extraction. The HOG feature of each cell would be normalized over each of its neighboring blocks to represent more local detail information, but which can lead to redundant dimensions of a feature representation (Yao, Wu, Chen, Hao, & Shen, 2014). Hence, it is challenging to make a trade-off between rich local details and redundancy.

2.2.2 Using Machine Learning

Several machine learning algorithms were used for traffic sign classification like linear discriminant analysis, support vector machines, random forest and *kd*-trees as well as ensemble classifiers.

Linear Discriminate Analysis (LDA) relies on maximum posteriori estimation of the class membership (Wu, Liu, Li, Liu, & Hu, 2013) and the density of classes are presumed to own multiple variate Gaussian and general co-variance matrix. Random Forest is an ensemble method that consists of the set of non-pruned random decision trees, which are all built based on the random training data. The classification output is generated from the majority of voting over all decision trees (Stallkamp et al., 2011). Support Vector Machines (SVM) not only classify the data according to *n*-dimensional data plane with a hyper plane, but also separate non-linearly scattered data by converting the classification plane to higher dimensions (Park & Kim, 2013).

Machine learning techniques provided a wealth of benefits for classifying traffic signs, but they were unable to handle the features, such as various sizes of images and aspect ratios, which must be completed manually. Thus, the feature generated process was always time-consuming and error-prone (Mogelmose et al., 2012).

2.3 Convolutional Neural Network

In the field of deep learning, a CNN model represents a class of deep neural networks and is offered for visual imagery (Valueva, Nagornov, Lyakhov, Valuev, & Chervyakov, 2020). CNNs have multilayer perceptron, which is fully connected. The meaning of fully connection in multilayer perceptron is that each neural in one layer is connected to all neurons of next layer. This architecture can effectively prevent data from overfitting. Their specific applications have image and video recognition, classification, medical image analysis and natural language processing, etc.

A convolutional neural network is composed of an input and an output layer as well

as several hidden layers (Xu, Ren, Liu, & Jia, 2014). The hidden layers also consist of a series of convolutional layers that are based on a multiplication and other dot product. ReLU layer is regarded as the activation function and is generally followed by other convolutions like pooling layers, fully connected layers and normalization layers. They refer to as hidden layers as their inputs and outputs are hidden by the activation function and final convolution. The architecture of a typical CNN is introduced as follows:

2.3.1 Convolutional Layer

The core block of a CNN architecture is convolutional layer. It involves a set of kernels (or filters) that can only receive a small fraction but extend through the full depth of input volume. Each filter is learnable and will be involved across the height and width of input volume along with the computation of dot product between the entries of filters and the input. During the initial process, a 2D feature map will be produced (Liang & Hu, 2015). Finally, the network will learn from activate filters while detecting some specific features at some spatial position in the input (Géron, 2019). The full output of the convolutional layer is formed by storing the activation maps for all filters along the depth dimension. The output of a neural is a small region in the input and the parameters will be shared between neurons in the same activation map.

There are three hyperparameters controlled the size of the output volume of convolutional layer: Depth, stride and zero-padding (Bochinski, Senst, & Sikora, 2017). The depth of the output volume controls how many neurons in a layer that connect to the same region of the input volume. Stride controls the allocation of the depth of columns around the spatial dimensions referring to the width and height of the input. The value of stride (S) should be greater than zero and any integer. In practice, the lengths of S are usually less than three. Less receptive fields overlapping will lead the output volume to have smaller spatial dimensions while stride length is increasing. Zero-padding controls the spatial size of the output volume. Equation (2.1) is to calculate the number of neurons that can fit in a given volume:

$$n = \frac{W-K+2P}{S} + 1 \quad (1)$$

where W is the size of input volume. The kernel size of the convolutional layer neurons is denoted by K. The length of stride and the amount of zero-padding are represented by S and P, respectively. Generally, we set the zero-padding as $P = \frac{K-1}{2}$ while $S = 1$, which make the input volume and output volume have the same spatial size.

2.3.2 Pooling Layer

Similarly, the pooling layer is responsible for reducing the spatial size of the convolved feature. It not only decreases the computational consumption through reducing dimensionality, but also is used to extract dominant features (Giusti, Cireşan, Masci, Gambardella, & Schmidhuber, 2013). Besides, the pooling layer serves to simplify the configuration of parameters and memory footprint, and thus controls overfitting. It is meaningful to insert pooling layers across series of convolutional layers (normally followed by ReLU layers) in a CNN architecture.

There are two dominant types of pooling, including max pooling (Nagi et al., 2011) and average pooling (Sun, Song, Jiang, Pan, & Pang, 2017). As the name implies, the max pooling returns the maximum value from the portion of the image covered by the kernel. The average pooling returns the average across all the values from the portion of the image covered by the Kernel. The most commonly used pooling form has the filter size of 2×2 and a stride of 2 down samples. The depth of volume is not changed.

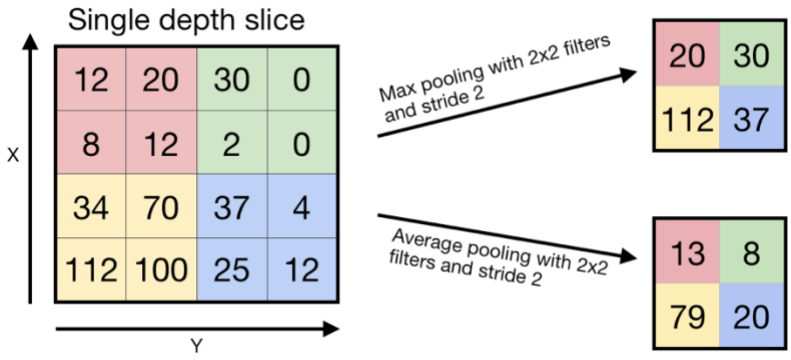

Figure 2.1 Max and average pooling with a filter of size 2×2 and stride 2

In practice, the max pooling is performed better than average pooling (Scherer,

Müller, & Behnke, 2010). Max pooling was used to discard the noisy activations and denoising while reducing dimensionalities. In terms of average pooling, it just simply performs the dimensionality reduction to suppress noises. Hence, we can say that the max pooling is a better option comparing to average pooling.

Due to the aggressive reduction in the size of representation (Suárez-Paniagua & Segura-Bedmar, 2018), they get rid of pooling layer in favor of the architecture of the pure repeated convolutional layers, which is known as the all convolutional net. Discarding pooling layers can benefit the generative model training, such as generative adversarial networks (GANs) (Zhang, Goodfellow, Metaxas, & Odena, 2019). This is the recent trending in this field.

2.3.3 ReLU Layer

Rectified linear unit (ReLU) utilized the non-saturating activation function $f(x) = \max(0, x)$ (Krizhevsky, Sutskever, & Hinton, 2012) to remove negative values from an activation map and replace them with number 0. This operation enhances the nonlinearity of the decision function and the whole network without influencing the reception domain of the CONV layer. There are other options available for increasing the nonlinear properties, such as the saturating hyperbolic tangent $f(x) = \tanh(x)$, $f(x) = |\tanh(x)|$ and the sigmoid function $\sigma(x) = (1 + e^{-x})^{-1}$. ReLU has gain more popularity comparing to other functions due to the faster speed of training the neural network (Krizhevsky et al., 2012).

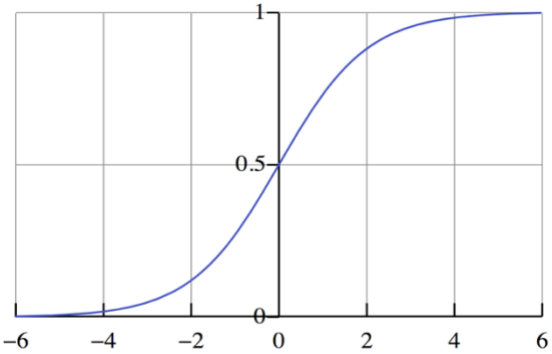

Figure 2.2 The sigmoid function

2.3.4 Fully connected layer

In the fully connected layer, the neurons connect to all activations as seen in the regular neural networks. Generally, inserting a fully connected layer is a cheap way to capture the nonlinear combination of high-dimensional features as represented by the yield of CONV layer (Albawi, Mohammed, & Al-Zawi, 2017).

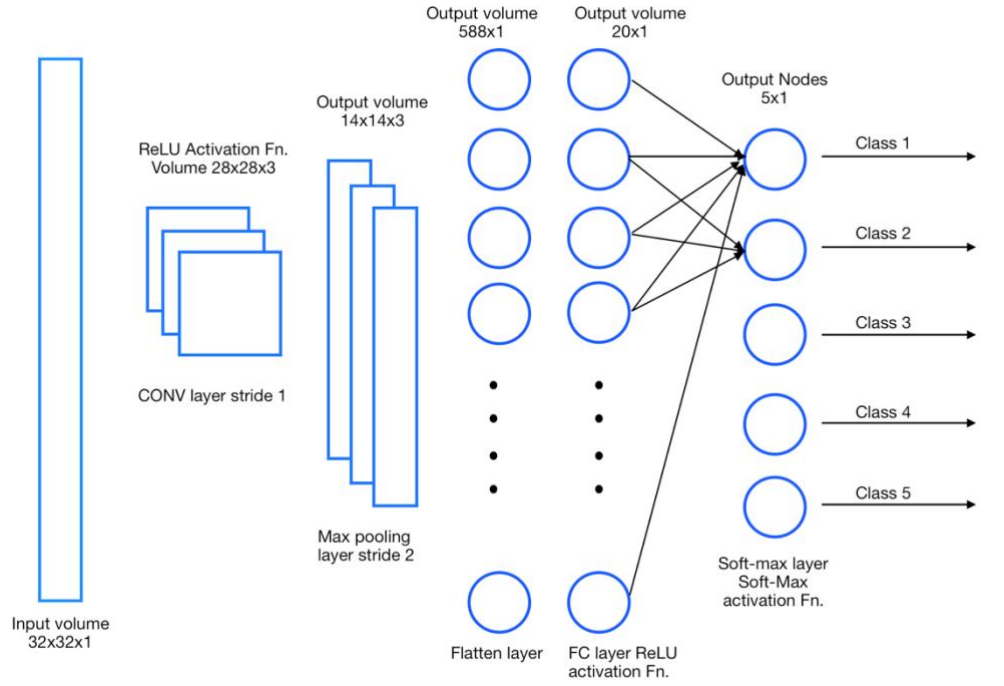

Figure 2.3 Fully connected layer (FC layer)

2.3.5 Loss Layer

The loss layer is normally the last layer in a neural network, which specifies the penalization process of training to the deviation between true labels and predicted results (Xie, Wang, Wei, Wang, & Tian, 2016). Different tasks should apply with different loss functions. For instance, the Softmax loss is to predict a single class between dominant and certain low-level features in images. Sigmoid cross-entropy is to predict independent values of probability in [0, 1].

2.4 Typical Convolutional Neural Networks

The typical CNN architecture is composed of blocks of convolutional layers and pooling

layers followed by a fully connected layer and SoftMax layer at the end. Several such CNN models are AlexNet, VGGNet, LeNet, NiN and all convolutional (All CONV). Besides, some state-of-the-art architectures have been proposed, such as the GoogleNet (Al-Qizwini, Barjasteh, Al-Qassab, & Radha, 2017) with ResNet (Z. Wu, Shen, & Van Den Hengel, 2019) and DenseNet (Jégou, Drozdzal, Vazquez, Romero, & Bengio, 2017).

Actually, all of these architectures have the similar fundamental components (convolution and pooling). Different architectures may have their own topological distinction. For instance, in terms of DCNN (Jin, McCann, Froustey, & Unser, 2017), the AlexNet, VGGNet, GoogleNet could be the most appropriate architectures to employ since they have shown the distinct performance on the task of object recognition. Some of architectures have shown their advantages in dealing with large volume of data, including GoogleNet and ResNet. However, the VGG networks is regarded as a common architecture in this field.

2.4.1 AlexNet

AlexNet was the champion CNN model in the most difficult ImageNet challenge named the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012 (Krizhevsky et al., 2012). This model proposed by Alex and others were deeper and wider than the previous neural network (LeNet), and it achieved the astonishing recognition accuracy against all the traditional approaches. The appearance of AlexNet could be seen as the turning point of the development of using machine learning and computer vision for object detection and classification tasks.

There are two innovative concepts introduced in the architecture of AlexNet. Firstly, the first convolutional layer of AlexNet applied Local Response Normalization (LRN) while performing the convolution and max pooling. LRP can be either applied on single channel and feature maps, or applied across single channel and feature maps (Hong-meng, Di, & Xue-bin, 2017). The formular for LRN is:

$$b_{x,y}^{i} = a_{x,y}^{i} / \left(k + \alpha \sum_{j=\max\left(0, i-\frac{n}{2}\right)}^{\min\left(N-1, i+\frac{n}{2}\right)} \left(a_{x,y}^{i}\right)^{2} \right)^{\beta} \tag{2.2}$$

where $a_{x,y}^{i}$ denotes the value yields by the number of i convolution at the position (x, y) and the result of outputting by the ReLU activation function. n is the number of neighboring convolution kernels, and N is the total number of convolution kernels in this layer. The rest of variants are parameters, which are obtained in the experimental validation set.

2.4.2 VGGNet

The Visual Geometry Group neural network (VGGNet) was proposed by the Visual Geometry Group, which is the runner of the 2014 ILSVRC. This network initially showed that the importance of the depth of a network was the crucial factor to achieve higher accuracy of recognition and classification. Two convolutional layers were contained in VGG architecture and both of them utilized the ReLU activation function, which was also used in the following activation function with a single max pooling and several fully connected layers. Three VGG models with different number of layers were proposed, including VGG-11, VGG-16 and VGG-19.

All versions of VGG models ended with three fully connected layers. However, they had different number of convolutional layers. VGG-11, VGG-16 and VGG-19 contained 8, 13 and 16 convolutional layers, respectively. Among them, the VGG-19 was the most computational consumption model, required 138M weights and 15.5M MACs.

2.4.3 ResNet

The Residual Network (ResNet) was developed by Kaiming He *et al.* with the intention of creating a deeper network that can avoid the effect of the vanishing gradient issue (He, Zhang, Ren, & Sun, 2016). ResNet architectures can be performed with different number of layers, such as 34, 50, 101, 152 and even 1202. Among them, the most popular ResNet

architecture had 50 layers, including 49 convolutional layers and 1 fully connected layer. It is worthy to mention that even though the ResNet had 152 layers the complexity of it was still lower than VGGNet.

ResNet is a typical network with residual connection. The final ouput of a residual layer can be defined by the following equation:

$$x_l = F(x_{l-1}) + x_{l-1} \tag{2.3}$$

where x_l is defined as the output of a residual layer. Thus, x_{l-1} is generated based on the output of previous layer. $F(x_{l-1})$ represents the output after performing other operations, such as convolution with various size of filters and Batch Normalization (BN) followed by an activation function like ReLU. The residual networks generally are composed of several fundamental residual blocks, but the operations within the blocks are varied corresponding to different residual architectures (He *et al.*, 2016).

Recently, several improved residual networks have been proposed. For example, a residual network was known as aggregated residual transformation (S. Xie, Girshick, Dollár, Tu, & He, 2017). Moreover, several researchers have combined residual units with Inception, and mathematically it can be expressed by the following eq. (2.4).

$$x_l = F\left(x_{l-1}^{3\times3} \odot x_{l-1}^{5\times5}\right) + x_{l-1} \tag{2.4}$$

where $\odot$ is used to express the concentration operations between two outputs produced by the 3×3 and 5×5 filters. Following the convolutional operation is performed and the outputs of the operation are attached with the inputs of block x_{l-1} (Szegedy, Vanhoucke, Ioffe, Shlens, & Wojna, 2016).

2.5 Object Detection Models

2.5.1 Faster R-CNN

Faster R-CNN is an improved network based on the design of Fast R-CNN and R-CNN,

which use region proposal algorithm like selective search with CNN to generate regional objects in an image. However, the method to propose region proposal requires a lot of time consumption even though it is performed directly to the CNN.

In the architecture of Faster R-CNN, the Region Proposal method (selective search) is replaced by a more advanced approach named Region Proposal Network (Ren, He, Girshick, & Sun, 2015). This network leverages the extracted features of CNN to generate region proposal. We can say that Faster R-CNN is a combination of RPN and Fast R-CNN detector.

A RPN takes an image as input and output a set of rectangular object proposals and each of them is attached with an objectiveness score (Ren *et al.*, 2015). In this network, a concept of anchor boxes is introduced. In an image, some referencing boxes are placed at different positions. The number of k anchor boxes are hyperparameter in the network and generated for each pixel based on the feature map which outputs of CNN. The total number of anchor boxes can be calculated by $h * w * k$ ($h * w$ is the size of feature map). A targeted object will be covered by multiple anchor boxes, and then these redundant predicted results are removed by non-max suppression (Liu, Zhao, & Sun, 2017). Different sizes of anchor boxes replace the operation of using multiple scales at test time. Comparing to Fast R-CNN and R-CNN, the speed of region proposal with selective search is 2 seconds per image but it is just 10 milliseconds.

The loss function applied in Faster R-CNN is similar to the previous networks (e.g. multitask loss). The mathematical expression for multitask loss function is shown below:

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*) \qquad (2.5)$$

Where p_i denotes the predicted probability which yields by classification, p_i^* denotes the similarity of ground truth. t_i and t_i^* respectively represent the predicted box and ground truth box.

2.5.2 You Only Look Once (YOLO)

The release of You Only Look Once (YOLO) was a milestone in the field of object detection. It was presented by Joseph Redmon *et al.* (2016) and immediately gained a lot of attention by fellow workers in computer vision. YOLO is a single network and can be improved end-to-end directly on the performance of detection (Redmon, Divvala, Girshick, & Farhadi, 2016). Instead of repurposing classifiers to complete detection task, the proposers framed object detection as a regression problem to separate bounding boxes and related class probabilities. In other words, the YOLO simplified the process of generating bounding boxes and class probabilities. Compared to cutting-edge detection architecture, it not only speeds up the training process but also doubles the accuracy of real-time detection, even if it has more localization errors.

The workflow of YOLO is briefly introduced as follows: YOLO as a single neural network extracts features from the entire image and uses them to predict each bounding box while predicting all bounding boxes for an image. It divides an input image into an $S \times S$ grid. When the center of an object overlaps with a grid cell, this cell will be used to detect this object. Each gride cell should predict number of bounding boxes and the confidence stores corresponding to these boxes. If there is no object falling into a gride cell, the confidence store is defined as 0.

The confidence store should be the intersection over union (IOU) between the predicted box and the ground truth and the IOU value can be calculated by $\Pr(Object) * IOU_{pred}^{truth}$. Each bounding box has five variables, including x, y, w, h and confidence store. The x, y will be used to produce the coordinate of the center of the bounding box in relation to the bounds of the gride cell. Besides, each cell also predicts the probability of conditional class $\Pr(Class_i|Object)$. Finally, the confidence stores of a specific class for each box is formally defined as:

$$\Pr(Class_i|Object) * \Pr(Object) * IOU_{pred}^{truth} = \Pr(Class_i) * IOU_{pred}^{truth} \quad (2.6)$$

which shows how well the predicted box fits the object.

Different versions of YOLO network have been released in 2016, 2017 and 2018. Recently, YOLOv4, YOLOv5 and PP-YOLO successively proposed within just few months after the release of those three main versions of YOLO. Although the appearing of YOLOv5 has made a fierce discussion in the community, the easier implementation and several improvements (e.g. mosaic data augmentation and auto-learning anchor boxes) cannot be ignored among the YOLO network family.

Chapter 3
Methodology

This chapter mainly expounds the details of the implementation of traffic sign recognition which performed on two trending CNN models, including YOLOv5 and Faster R-CNN. The dataset preparation, training process and evaluation methods will be also introduced in the following content.

3.1 Traffic-Sign Recognition (TSR) in NZ

Traffic sign recognition is a task which is considering with both object detection and classification. It is a real-world application that computer vision techniques are aligned with to develop driver assistant system. In practice, the implementation of this task usually confronts with a lot of uncertain issues, such as color fading, disorientation and variations in size and shape (Fleyeh, 2008). Recently, there are a lot of researches available to deal with such problems and provide solutions to boost the performance of traffic sign recognition. The following diagram illustrates the general workflow of TSR task:

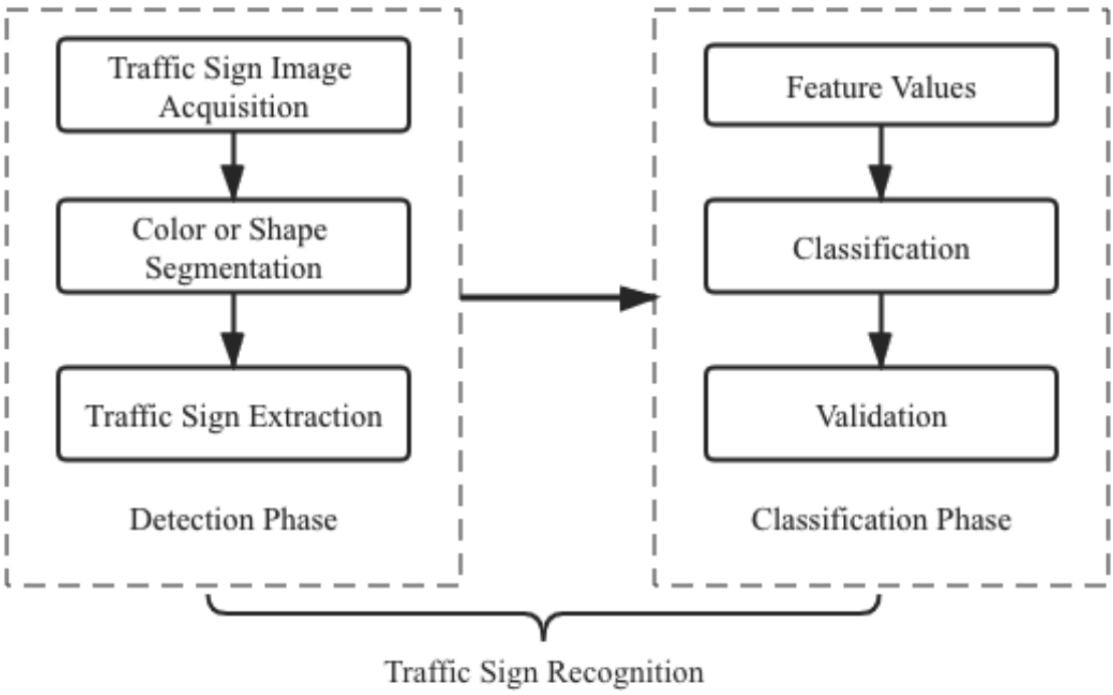

Figure 3.1 The workflow of traffic-sign Recognition

Traffic signs in NZ mainly are categorized into three groups: Regulatory (including general, parking and road user restrictions), Warning (including temporary and permanent), Advisory (including guide and route signs, e.g. street name, community facilities, tourist signs, service signs and general information signs). Although the design of traffic signs in NZ followed the dominant trending and international standards, NZ is not a signatory to the convention on international road signs and signals, and thus some of its traffic signs have different shape and function. According to the above inconsistency,

it is necessary to take the rebuilding of the customized dataset into consideration for effectively recognizing traffic signs in NZ.

3.2 Data Collection

In this project, we used the 12-megapixel wide-angle camera of iPhone 11 to capture the realistic traffic sign images in Auckland. Due to the lower appearing frequency of traffic signs comparing to pedestrians and vehicles, we directly took traffic sign images using camera instead of recording video. The pixels of the images are 1080×1440 and stored in *.JPEG* format. Our dataset (NZ-Traffic-Signs 3K) consist of 3436 images and 3545 instances in total: Stop (236 instances), Keep Left (536 instances), Road Diverges (505 instances), Road Bump (619 instances), Crosswalk Ahead (636 instances), Give Way at Roundabout (533 instances) and Roundabout Ahead (480 instances).

Table 3.1 Examples of seven categories in our benchmark (NZ-Traffic-Signs 3K)

	Stop	Keep Left	Road Diverges	Road Bump	Crosswalk Ahead	Give-away at Roundabout	Roundabout Ahead
Traffic Signs (NZ)							

In order to avoid the overfitting when training the chosen models, we utilized data augmentation to expand our dataset. Several basic manipulations for data augmentation include flipping, rotation, shearing and adding noise as well as blurring images. In this case, we merely applied two augmentation operations, including adding noise and blurring images, based on our original dataset because these methods could to deal with the distorted objects, which could impact the quality of our dataset and even degrade the

accuracy of our training models. The manipulations were implemented by importing a Python library, named Skimage.

Figure 3.2 Augmented images: (a) original images (b) adding noise (c) blurring images

3.3 Research Design for Training Faster R-CNN

In this experiment, we chose Faster R-CNN to conduct recognizing traffic signs with on our dataset. Faster R-CNN needs a traditional CNN as the basic convolutional layers for feature extraction. A pretrained VGG16 model was used to assist us in outputting the feature map.

3.3.1 Dataset Structure for Training Faster R-CNN

In order to successfully implement Faster R-CNN, the data directory should follow the structure of PASCAL VOC dataset. The dataset organization structure is divided into five parts, including Annotations, ImageSets, JPEGImages and SegmentationClass as well as SegmentationObject. The structure is shown in Figure 3.3.

Figure 3.3 Organized dataset directories for Faster R-CNN

The Annotations folder is responsible for storing all the *.xml* label files. The ImageSets stores the segmented datasets for training, validation and testing. Then, the JPEGImages is used to store the *.jpeg* images. Finally, as the name revealed, the SegmentationClass and SegmentationObject are used to the segmented images based on the criteria of different classes and objects.

As for training the Faster R-CNN, the annotation files are stored in *.xml* format and formatted very restrictedly. We use a labelled tool, named *LabelImg,* to label all the images in our dataset.

000002.xml

```
<annotation>
        <folder>final_images</folder>
        <filename>000002.jpg</filename>
        <path>/Users/qin/Desktop/test_dataset/final_images/aug_image_2.jpg</path>
        <source>
                <database>Unknown</database>
        </source>
        <size>
                <width>1080</width>
                <height>1440</height>
                <depth>3</depth>
        </size>
        <segmented>0</segmented>
        <object>
                <name>crosswalk ahead</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>626</xmin>
                        <ymin>716</ymin>
                        <xmax>742</xmax>
                        <ymax>848</ymax>
                </bndbox>
        </object>
</annotation>
```

Figure 3.4 An example of annotation file format for traffic-sign recognition

The LabelImg can export voc format label files while labelling the images. The process of labelling our images is shown down below:

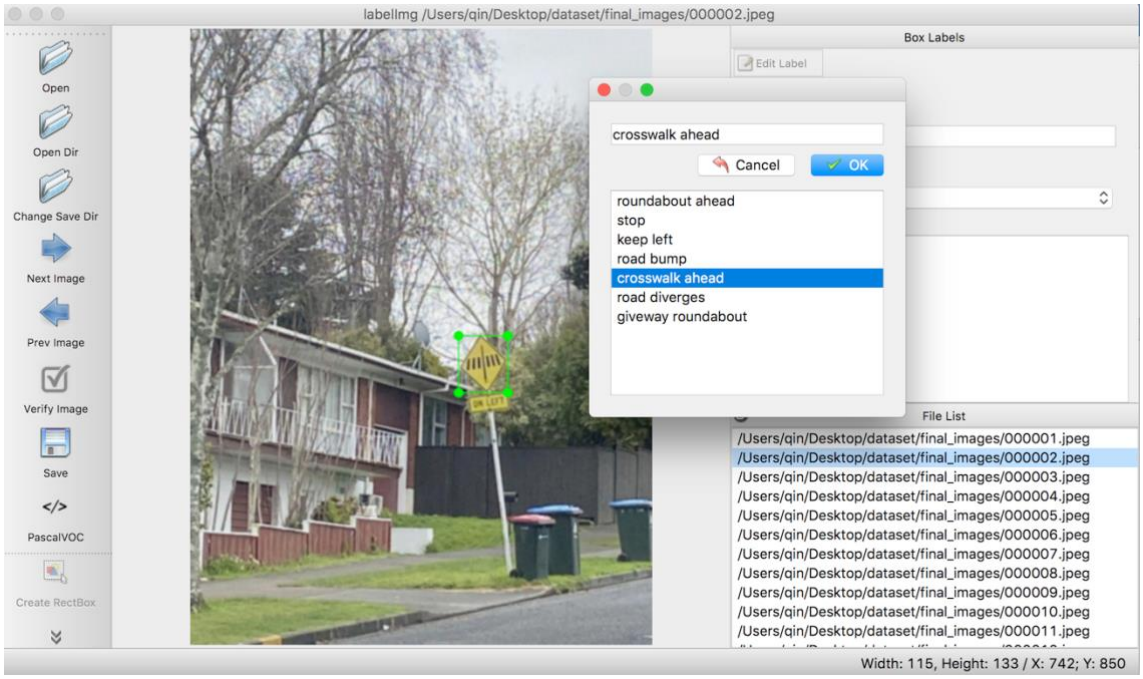

Figure 3.5 Labelling an image in our dataset

3.3.2 Experimental Environment and Parameters for Faster R-CNN

Due to the implementation based on Python, several dependencies should be preinstalled to setup the experimental environment. *Caffe* must be built with support for Python layers. Some Python packages are needed, including Cython, python-opencv and easydict, etc. In order to train Faster R-CNN with VGG16, the CUDA device with Tesla V100-SXM2-16GB are necessary.

During the process of training the Faster R-CNN, critical parameters should be preliminarily set, and the details are shown in Table 3.2. In neural networks, the ideal situation is to make the error function reach a global minimum, but in practice the error may comprise of many local minimums where the optimization can be stuck in, and thus the global optimum cannot be guaranteed. In this case, the algorithm will use the unoptimized results to lead to sub-optimal results. Momentum term can increase the step size to jump from the local minimums. Most importantly, a large value of Momentum can contribute to a faster model convergence. Normally, a large Momentum had better match

with a smaller Learning Rate (LR), otherwise the algorithm might skip the global minimum with a huge step. In this experiment, we set the Momentum to 0.9 and LR to 0.01 to avoid the above issue.

Table 3.2 The parameters for training Faster R-CNN

Parameters	Setting
Momentum	0.9
Learning Rate	0.01
Max Epochs	200
Batch Size	24
Weight Decay	0.0005

The max number of training epoch is also a key factor that influence the performance of a model. An appropriate of max training epochs will contribute to a lower training and validation rate while overfitting is non-existent. As for our project, the most appropriate max epochs number is 200 which were tested after several pilot experiments.

Batch size as another vital parameter in training neural networks is used to estimate error gradient. It is the number of examples from the part of training data to achieve the error gradient estimation. Generally, the more training examples used in the estimation, the higher accuracy will be achieved and the more possibility that the network weights will be adjusted in a way that will boost the performance of the model. The batch size is set to 24 in this experiment.

The data is complicated in the real-world. Having fewer parameters in the process of training a model is a way that can prevent a model from suffering from complex data, but it is an impractical and limited solution. The more parameters are employed with, the more intersections among them exist. These intersections mean more non-linearities, which will help a model solve complex problems. Thus, in order to solve the issue, a parameter called Weight Decay (WD) is used to penalize the complexity. We set the WD to 0.0005 for training Faster R-CNN.

3.4 Research Design for Training YOLOv5

The second performed model in this project is YOLOv5, which was just less than fifty days later than the release of YOLOv4. Although the appearing of it has gained a lot of attentions and debates in the community, it was indeed published with a number of improvements and distinctions. The improvements are mainly reflected in two aspects: improved the accessibility for detecting real-time objects and the performance of prediction either on training speed or accuracy.

Firstly, YOLOv5 is the first release in the YOLO family to be written in PyTorch instead of using PJ Reddie's Darknet (Chen, 2019). The implementation based on PyTorch makes the process of deployment easier and simpler. Secondly, YOLOv5 is extremely fast. In the official YOLOv5 Colab notebook, the inference times up to 0.007 seconds per image, meaning 140 frames per second. Compared to the previous models (e.g. YOLOv4), the processing speed is a milestone in the development of object detectors. Thirdly, it is more accurate. In one of the most popular tests of performing YOLOv5 (Blood Cell Count and Detection Dataset), it achieved approximately 0.895 mean average precision (mAP) after running 100 epochs. Finally, it is much smaller than YOLOv4 and it provides four different size models: YOLOv5s (smallest one with 7.5M params), YOLOv5m (21.8M params), YOLOv5l (47.8M params) and YOLOv5x (largest one with 89.0M params). In this experiment, we chose the smallest model (YOLOv5s) to perform with due to the relatively small scale of our dataset.

3.4.1 Dataset Structure for Training YOLOv5

In order to train the YOLOv5 model, the first step is to label the images in our dataset. A graphical image annotation tool (*LabelImg*) was used to label the images in our dataset. The labelling process is shown in Figure 3.4. Each of the corresponding annotation files was saved as a **.txt* file and the produced labels were exported by YOLO format. The specifications of **.txt* are:

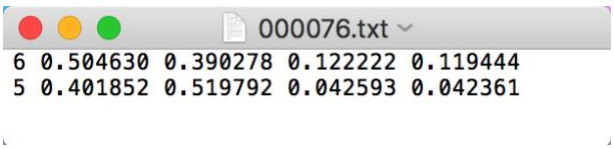

Figure 3.6 An example label file with two traffic signs

- Each object in an image is denoted by one row;
- Each row has the unified format: *class, x_center, y_center, width, height*;
- Box coordinates (*x, y, w, h*) must be normalized in the range of (0, 1). Our initial images are all in pixel sizes. Thus, the *x_center* and *width* are required to be divided by image width. The *y_center* and *height* are required to be divided by image height.
- Class numbers should start from 0. For example, we have seven categories in this case and hence the range of class numbers is (0, 6).

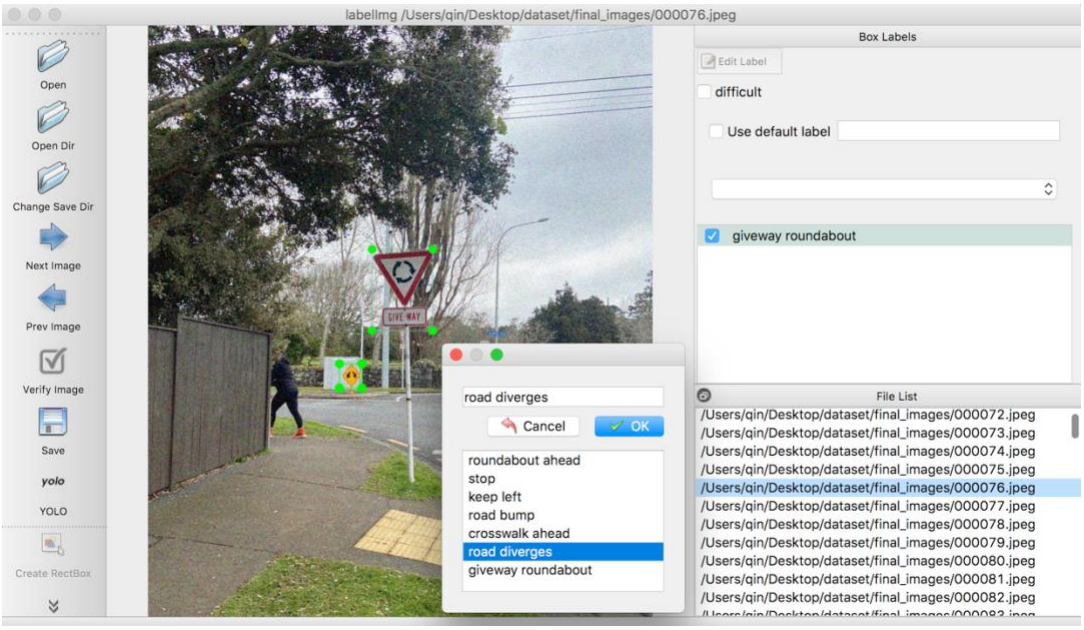

Figure 3.7 Labelling an image by using *LabelImg*

After generating the label files based on our dataset, the next step is to organize directories which save the train and validation images and labels. In the official version of YOLOv5 released by Glenn Jocher, the folder stored data (named $/final\_dataset$ in this case) must be next to the $/yolov5$ directory, and make sure the folder stored all the labels ($final\_datset/labels$) next to the folder stored all the images ($final\_datset/images$).

Figure 3.8 Organized dataset folders for YOLOv5

3.4.2 Architecture and Functions of YOLOv5 Model

The model structure of YOLOv5 is the same as the common single-stage object detector (S. Wu, Li, & Wang, 2020). It has three main parts: model backbone, model neck and model head. The overall model architecture is illustrated in Table 3.3.

Table 3.3 The Architecture of YOLOv5s

Prameters	YOLOv5 Head
nc: 7 # number of classes depth_multiple: 0.33 # model depth multiple width_multiple: 0.50 # layer channel multiple	[[-1, 1, Conv, [512, 1, 1]], [-1, 1, nn.Upsample, [None, 2, 'nearest']], [[-1, 6], 1, Concat, [1]], # cat backbone P4 [-1, 3, BottleneckCSP, [512, False]], # 13 [-1, 1, Conv, [256, 1, 1]], [-1, 1, nn.Upsample, [None, 2, 'nearest']], [[-1, 4], 1, Concat, [1]], # cat backbone P3 [-1, 3, BottleneckCSP, [256, False]], # 17 (P3/8-small) [-1, 1, Conv, [256, 3, 2]], [[-1, 14], 1, Concat, [1]], # cat head P4 [-1, 3, BottleneckCSP, [512, False]], # 20 (P4/16-medium) [-1, 1, Conv, [512, 3, 2]], [[-1, 10], 1, Concat, [1]], # cat head P5 [-1, 3, BottleneckCSP, [1024, False]], # 23 (P5/32-large) [[17, 20, 23], 1, Detect, [nc, anchors]], # Detect(P3, P4, P5)]
Anchors	
- [10,13, 16,30, 33,23] # P3/8 - [30,61, 62,45, 59,119] # P4/16 - [116,90, 156,198, 373,326] # P5/32	
YOLOv5 Backbone	
# [from, number, module, args] [[-1, 1, Focus, [64, 3]], # 0-P1/2 [-1, 1, Conv, [128, 3, 2]], # 1-P2/4 [-1, 3, BottleneckCSP, [128]], [-1, 1, Conv, [256, 3, 2]], # 3-P3/8 [-1, 9, BottleneckCSP, [256]], [-1, 1, Conv, [512, 3, 2]], # 5-P4/16 [-1, 9, BottleneckCSP, [512]], [-1, 1, Conv, [1024, 3, 2]], # 7-P5/32 [-1, 1, SPP, [1024, [5, 9, 13]]], [-1, 3, BottleneckCSP, [1024, False]], # 9]	

Model backbone is normally responsible for extracting important features from images. The Cross Stage Partial Networks (CSP) are employed as the backbone of YOLOv5. This state-of-the-art network effectively mitigated the problem of heavy inference computations from the network architecture (Wang et al., 2020). In other words, the CSPNet can significantly reduce the processing time with deeper neural networks. Thus, the employment of it surely contributes to the progress made by YOLOv5.

Model neck is used to generate feature pyramids, which are a fundamental component of recognition systems for detecting multiscale objects and built only based on CNNs (Lin, Dollár, et al., 2017). It not only assists the detector in identifying the same object with various sizes and scales, but also has shown the obvious strength in unseen data. There are off-the-shelf feature pyramids techniques, such as Feature Pyramid Network (FPN) (Lin, Goyal, Girshick, He, & Dollár, 2017), Bi-directional Feature Pyramid Network (BiFPN) (Tan, Pang, & Le, 2020) and Path Aggregation Network (PAN) (S. Liu, Qi, Qin, Shi, & Jia, 2018), etc. In YOLOv5, the model neck is PANet.

Model head is used for performing the final detection. In this part, the anchor boxes are applied on features and output final vectors with class probabilities, objectiveness

scores and bounding boxes. The model head of YOLOv5 follows the previous YOLOv3 and YOLOv4.

The choice of activation functions is vital in deep neural network. Recently, there are a lot of activation functions available like Leaky ReLU (LReLU) (Maas, Hannun, & Ng, 2013), mish, etc. The chosen activation functions in YOLOv5 are LReLU and Sigmoid. Specifically, the LReLU is added in the middle/hidden layers and the Sigmoid is added in the final detection layer.

In terms of ReLU, it was proposed to alleviate potential problems caused by zero gradient, which allows a small and non-zero gradient presented if the unit is not active (Maas et al., 2013),

$$h^{(i)} = \max\left(w^{(i)T}x, 0\right) = \begin{cases} w^{(i)T}x & w^{(i)T}x > 0 \\ 0.01w^{(i)T}x & else \end{cases} \tag{3.1}$$

where $w^{(i)}$ represents the weight vector of the i^{th} middle layer and x is the input. The introduction of Sigmoid activation function can refer to Section 2.3.3 in Chapter 2. As for the optimization function in YOLOv5, we have two options, including Stochastic Gradient Descent (SGD) and Adam. The default optimizer is SGD, but it can be transferred to Adam by the command *"-- adam"*.

The last emphasized function is the loss function or cost function. In YOLOv5, the loss is computed based on three values: objectiveness score, class probabilities and bounding box regression score. YOLOv5 imports the Binary Cross-Entropy with Logits Loss (BCELoss) from PyTorch for calculating the compound loss. This method combines a Sigmoid layer with the BCELoss in one single class, which is more numerically stable than adding the BCELoss after a Sigmoid layer. The unreduced loss can be described as:

$$l(x, y) = L = \{l_1, \dots, l_N\}^T \tag{3.2}$$

$$l_n = -W_n\left[y_n \cdot \log \sigma(x_n) + (1 - y_n) \cdot \log\left(1 - \sigma(x_n)\right)\right] \tag{3.3}$$

Where N is the batch size. When the reduction is not none, the error of a reconstruction can be measured by

$$l(x,y) = \begin{cases} mean(L), & reduction = 'mean' \\ sum(L), & reduction = 'sum' \end{cases} \tag{3.4}$$

While predicting the multilabel classification the loss can be expressed as follows, which achieves by adding weights into positive instances.

$$l_c(x,y) = L_c = \{l_{1,c}, \dots, l_{N,c}\}^T \tag{3.5}$$

$$l_{n,c} = -W_{n,c}\left[p_c y_{n,c} \cdot \log \sigma(x_{n,c}) + (1 - y_{n,c}) \cdot \log\left(1 - \sigma(x_{n,c})\right)\right] \tag{3.6}$$

where *c* is the class number. For example, *c* = 1 means the single label classification and *n* is the number of the instances in the batch as well as p_c is the weight of positive instances for the c class.

3.4.3 Experimental Environment and Parameters for YOLOv5

In this section, we will introduce how we set up the experimental environment and be explicit about the parameters of training YOLOv5. Firstly, YOLOv5 was developed by Python. Different versions of dependencies should be installed to support the implementation of our project. The PyTorch version should be $\geq$ 1.6, Python version $\geq$ 3.8 and CUDA version 10.2. The details of requirements for this project are provided in Table 3.2. Furthermore, this experiment was performed on *Colab* using Tesla *V100-SXM2-16GB.*

Table 3.4 The details of the installed dependencies for YOLOv5

Package Name	Vesion
Cython	——
matplotlib	≥ 3.2.2
numpy	≥ 1.18.5
opencv-python	≥ 4.1.2
pillow	——
PyYAML	≥ 5.3
scipy	≥ 1.4.1
tensorboard	≥ 2.2
torch	≥ 1.6.0
torchvision	≥ 0.7.0
tqdm	≥ 4.41.0

Secondly, a customized $.yaml$ file should be created to describe our dataset and this file is saved under $.data$ directory which is responsible for storing the dataset description file.

The parameters for training YOLOv5 are shown in Table 3.5. In addition to the same parameters as the Faster R-CNN, several other parameters are used in this experiment, such as *giou* which is the GIoU loss gain, cls which is the classification loss gain, *cls_pw* which is the classification of BCELoss positive weight and *obj_pw* which is objectness BCELoss positive weight as well as the loss gain of objectness *obj*.

Table 3.5 The parameters for training YOLOv5

Parameters	Setting
Momentum	0.95
Learning Rate	0.00128
Max Epochs	200
Batch Size	16
Weight Decay	0.000201
giou	1.2
cls	15.7
cls_pw	3.67
obj	20.0
obj_pw	1.36

3.5 Evaluation Methods

To comprehensively evaluate the performance of YOLOv5 and Faster R-CNN, six metrics are considered in this traffic-sign recognition task. They are Generalized Intersection over Union (GIoU), the predicted probability of Objectness, Classification, Precision and Recall as well as mean Average Precisions with multiple IoU.

GIoU is the optimized version of Intersection over Union (IoU), which is the most commonly used metric used to justify the performance of detectors (Rezatofighi et al., 2019). IoU is used to determine true positives and false positives among predicted results. In our experiment, we chose the optimized IoU (GIoU) mainly because it overcomes the weakness of IoU optimization while appearing non-overlapped bounding boxes. Specifically, if $|A \cap B| = 0, IoU(A, B) = 0$. In this situation, the IoU is not able to reflect whether two bounding boxes are far from or vicinal each other. The GIoU mitigates this problem mainly from two perspectives (Rezatofighi et al., 2019):

- IoU is regarded as a distance (e.g. $L_{IoU} = 1 - IoU$), which fulfils all properties of a metric.

- IoU is constant confronting with the problem scale. This can keep two arbitrary shapes A and B separated from the scale of their space.

The GIoU is described by the following eq. (3.7).

$$GIoU = IoU - \frac{|C \backslash (A \cup B)|}{|C|} \tag{3.7}$$

Objectness metric reflects how likely a bounding box contains a targeted object (traffic sign in this case) (Kuo, Hariharan, & Malik, 2015). As we know, the output of the model in YOLO family is a set of boundary boxes and each box contains one box confidence score. The objectness is equivalent to the confidence score, which determines how accurate the boundary box is.

Classification accuracy (CA) is defined as the number of correct predictions divided by the total number of predictions. The following expression is used to calculate this score:

$$CA = \frac{true\_positive + true\_negative}{total\ predictions} \tag{3.8}$$

Precision is a metric that justifies the performance based on the specific class, especially when the distribution of dataset is unbalanced. This metric is regarded as a supplementary of CA due to the weakness of CA performing on uneven data. Precision is defined as

$$ACC = \frac{true\_positive}{true\_positive + false\_positive} \tag{3.9}$$

Recall is another important metric that indicates accuracy of the model. It refers to the percentage of the results correctly categorized by the model. Recall is expressed by

$$recall = \frac{true\_positive}{true\_positives + false\_negatives} \tag{3.10}$$

Mean Average Precision (mAP) is the modified version of Average Precision (AP), which is one of the most popular method for measuring the performance of object detectors such as YOLO, Faster R-CNN and SSD, etc. It is computed by the mentioned

metrics Precision and Recall (Henderson & Ferrari, 2016). The general definition of AP is searching for the area under the precision-recall curve. The range of recall and precision is always between 0 and 1, which makes the value of AP is within (0, 1).

$$AP = \int p(r)dr \tag{3.11}$$

The mAP just simply takes the mean of average precision, which is finally expressed by

$$mAP = \frac{1}{c} \sum_{c \in classes} \frac{true\_positive}{true\_positive + false\_positive} \tag{3.12}$$

where *c* is the class number that is 7 in this case. Generally, IoU is set to larger than 0.5. The mAP with multiple IoU in this experiment is employed. The IoU thresholds increase from 0.5 to 0.95.

Chapter 4

Results

In this chapter, our experimental results will be demonstrated. In addition, a comparison between two chosen model will be conducted based on the accuracy of them for recognizing traffic signs in NZ.

4.1 Data Description

After initially removed the redundant and poor-quality images, the total number of our dataset (NZ Traffic Signs 3K) is 3,439 of the 7 classes. All the images in the dataset are in pixel size 1080×1440. The distribution of all the classes are illustrated in Figure 4.1.

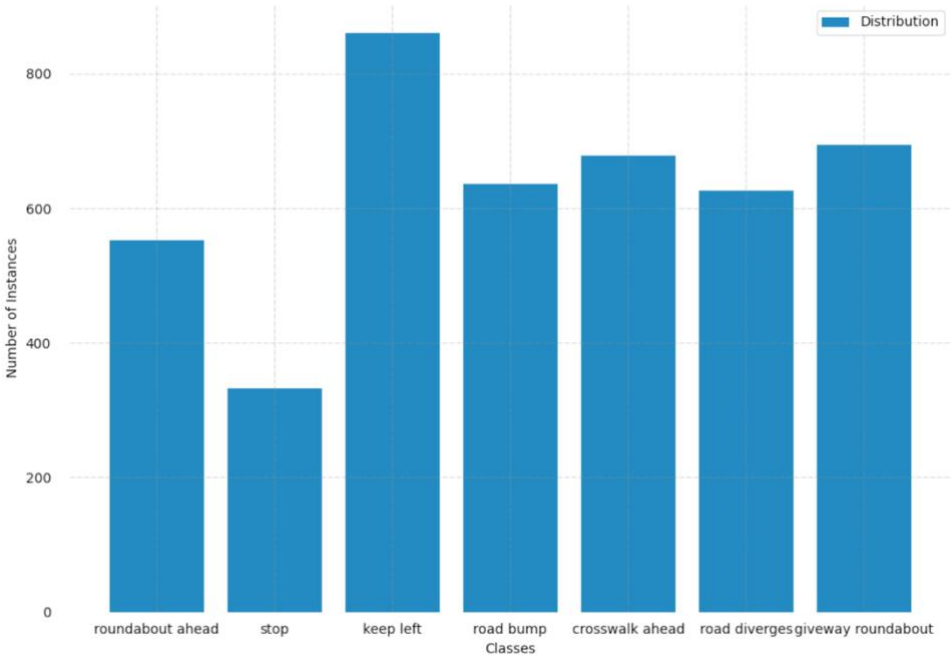

Figure 4.1 The distribution of seven classes in our dataset

In Figure 4.1, we see that the distribution of the data is relatively satisfactory except the class “Stop” that has the minimum number of instances comparing to other classes in the dataset. A balanced distribution across all the classes could not only benefit the performance of two models over all the classes but also the effectiveness of the comparison results between two models.

In Chapter 1, one of our main objectives in this project is to evaluate the performance of neural networks on recognizing the traffic signs in small sizes. Hence, we concerned more about the various sizes recognition of traffic signs in NZ. More clearly, we used a scatter plot to show the density of different sizes of traffic signs in our dataset.

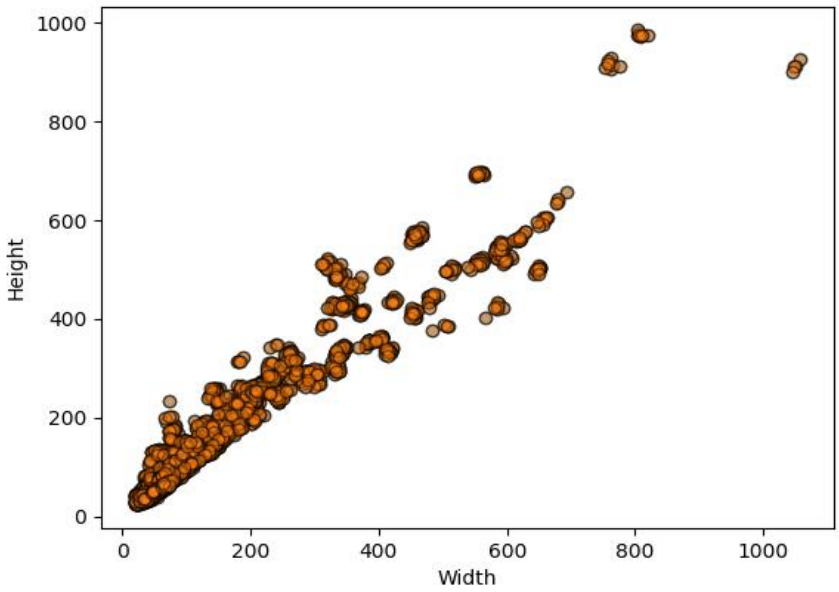

Figure 4.2 The density of different sizes of traffic signs in our dataset

Each dot in the scatter represents a traffic-sign sample in the dataset and (x, y) is practical pixel location in the corresponding image. As we see, most of pixel sizes are gathering in the pixel size of $(200,200)$, which is expected for training the chosen models to recognize smaller size objects. Besides, the positions of all the samples in the images are shown in Figure 4.3.

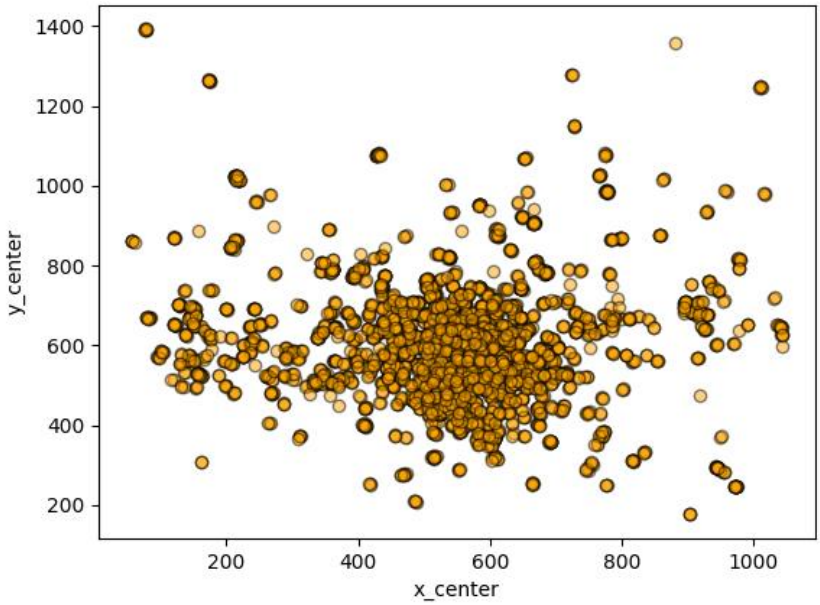

Figure 4.3 The positions of all samples in the images

In the training phase, we split our data into two parts 80% for training and 20% for validation.

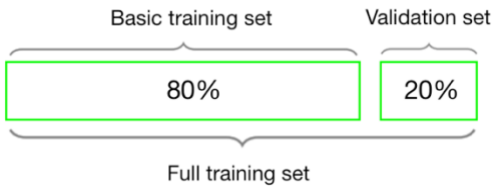

Figure 4.4 The distribution of our data for training and validation

4.2 Experiment Results of Faster R-CNN

In this experiment, we used the model Faster R-CNN as the detector and VGG16 as the classifier to perform the traffic-sign recognition task. The experimental results are provided in Table 4.1. We evaluate the performance of the Faster R-CNN with VGG16 mainly using three measures, including Precision, Recall and Mean Average Precision with IoU 0.5 ($mAP@0.5$). Fortunately, the accuracy of predictions is relatively good across seven classes.

Table 4.1 Experimental results for Faster R-CNN across seven classes

Index	Classes	Precision	Recall	mAP@0.5
0	Roundabout ahead	0.957	0.952	0.961
1	Stop	0.970	0.959	0.972
2	Keep left	0.899	0.903	0.900
3	Road bump	0.925	0.930	0.933
4	Crosswalk ahead	0.937	0.939	0.943
5	Road diverges	0.929	0.930	0.932
6	Give way at roundabout	0.964	0.958	0.962

The Faster R-CNN has shown an obvious strength in predicting traffic signs. It is worthy to mention that almost all the evaluation measures are over 0.9 except the prediction of Keep Left with the Precision score 0.899. The two classes with the highest precision score are Stop and Road Diverges, which is mainly due to the extremely distinct features exist in the design of these types of signs.

In order to evaluate the performance of the model on smaller traffic signs, we also performed another experiment and justify the results from this perspective. The same measures are applied to estimate the prediction results.

Table 4.2 Prediction results of various sizes of the traffic signs based on Faster R-CNN

Pixel size	Precision	Recall	mAP@0.5
<=200	0.907	0.914	0.915
[200, 400]	0.977	0.973	0.979
>=400	0.945	0.947	0.950

In Figure 4.2, we collected a large percentage of small size traffic signs in our dataset. Such amount of data guarantees the performance of the Faster R-CNN on recognizing small size objects in this experiment. The model achieves the highest precision for recognizing the traffic signs in pixel sizes [200, 400]. Although the lowest accuracy score appearing while recognition the traffic signs in smaller sizes (<=200 pixels), the score of this size category is still over 0.9, which is an impressive result of a CNN for recognizing objects.

After illustrating the specific results across different categories from the two perspectives, we also performed the Faster R-CNN on the whole dataset rather than solely training it according to different categories. The five measures mentioned in Chapter 3 (GIoU, Objectness, Classification, Precision, Recall and mAP with multiple IoU) are used to estimate the overall performance of the model. Those metric scores both in the process of training and validation are shown in Figure 4.5.

After trained 200 epochs, an obvious trend of convergence both shown in the process of training and validation for the losses of GIoU, Objectness and Classification. In terms of GIoU loss, the final score of it converges to less than 0.02. Incorporating the GIoU loss can improve the model performance on datasets (Rezatofighi et al., 2019). The objectness loss is 0.005 during the training, and even reaches to zero while validating the model.

Impressively, the Classification loss almost reaches zero both in the processes of training and validation.

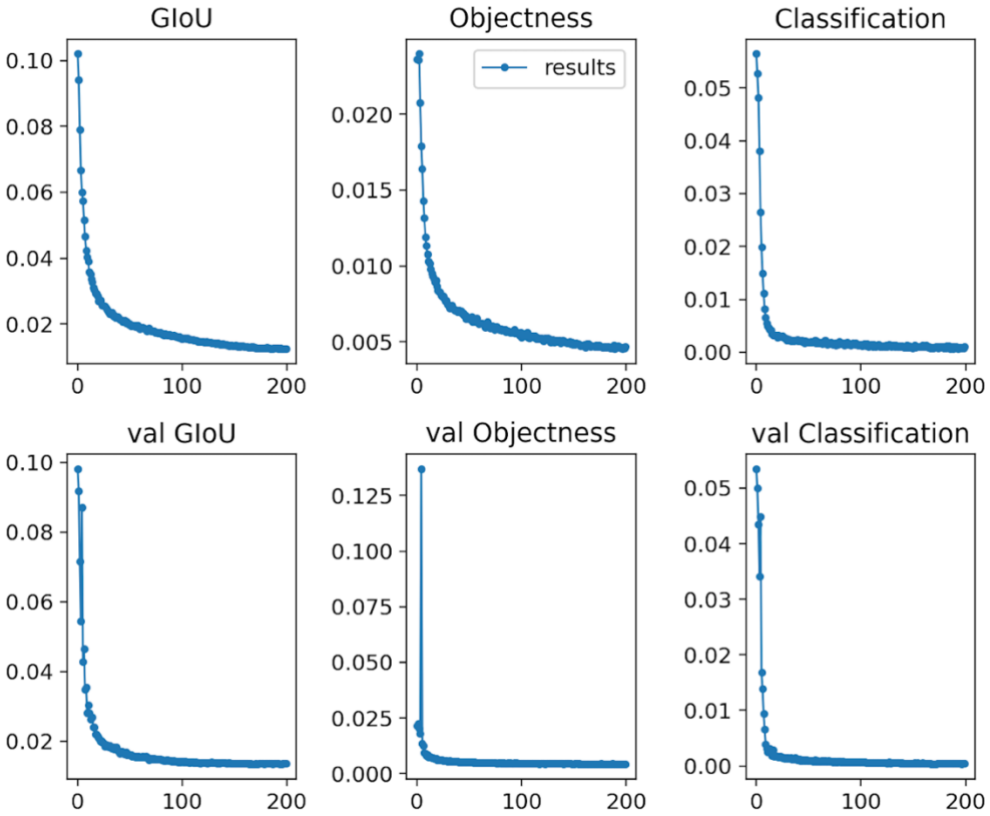

Figure 4.5 Three types of losses for Faster R-CNN

In Figure 4.6, the Precision-Recall curves are consistent, which means the scores of Precision increase along with the increase of Recall scores. The consistence between two curves affirms the good performance of the Faster R-CNN in the traffic-sign recognition. Moreover, the scores of mAP with 0.5 threshold of IoU are significantly converging to 1.

The mAP values with multiple IoU also converge to an impressive value 0.8. Finally, we tested the performance of the Faster R-CNN on several images. The predicted results are shown in Figure 4.7 and Figure 4.8.

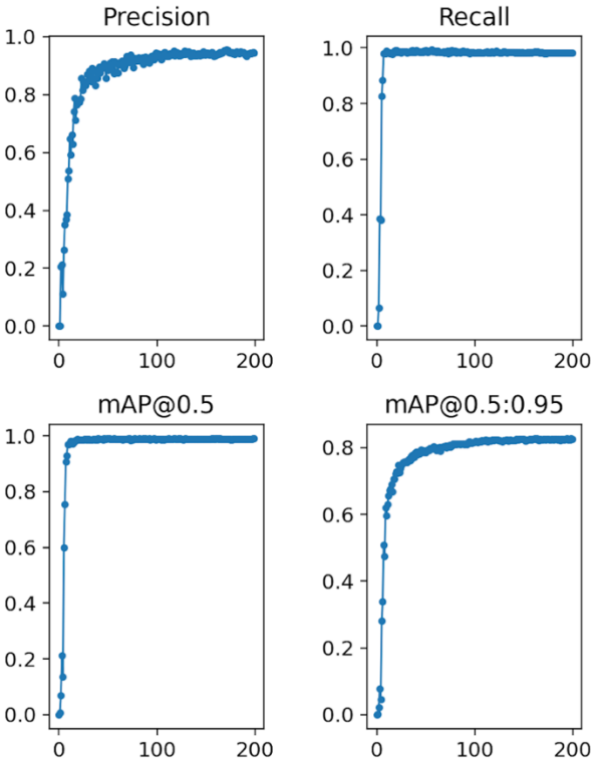

Figure 4.7 Several tested images with class index

Figure 4.8 Several tested images with confidence scores

4.3 Experimental Results of YOLOv5

In this experiment, we chose YOLOv5 to perform with and it is a newly released end-to-end network that is different from the Faster R-CNN. Similarity, we conducted the model training based on the categized of all seven classes in our dataset (NZ Traffic Signs 3K). The experimental results are provided in Table 4.3.

Table 4.3 Experimental results for YOLOv5 across seven classes

Index	Classes	Precision	Recall	mAP@0.5
0	Roundabout ahead	0.949	0.951	0.954
1	Stop	0.952	0.956	0.959
2	Keep left	0.901	0.912	0.923
3	Road bump	0.922	0.927	0.929
4	Crosswalk ahead	0.933	0.938	0.941
5	Road diverges	0.934	0.930	0.936
6	Give way at roundabout	0.955	0.957	0.960

In Table 4.3, the recognition result of Stop signs remains a high accuracy (0.952), even though it has a minimum number of samples in the dataset. Surprisingly, the predicted accuracy of Keep Left has a slightly increase (0.02) comparing to its accuracy rate reached by the Faster R-CNN. As for the rest of traffic-sign recognition, the output of the YOLOv5 maintains a high level of accuracy rate and impressively all the results are over 0.9.

Table 4.4 Prediction results of various sizes of the traffic signs based on YOLOv5

Pixel size	Precision	Recall	mAP@0.5
<=200	0.883	0.892	0.890
[200, 400]	0.976	0.971	0.974
>=400	0.931	0.939	0.939

In the interest of exploring the performance of networks on recognizing small size of traffic signs, we similarly performed an experiment according to various sizes of traffic signs. While predicting the traffic signs with less than 200 pixels, the YOLOv5 still achieved a fairly good performance. The YOLOv5 outputs the best results (0.976) while recognizing the traffic signs with medium sizes [200, 400] in this case. The accuracy rate has a slightly decrease comparing to the Faster R-CNN, but it is still over 0.88.

An overall performance of the YOLOv5 also was justified in this case. The distribution of training and validation sets are invariant, 80% for training and 20% for validation. The evaluation is conducted according to the same measures as the Faster R-CNN, including different losses, Precision and Recall as well as the mAP with multiple IoU.

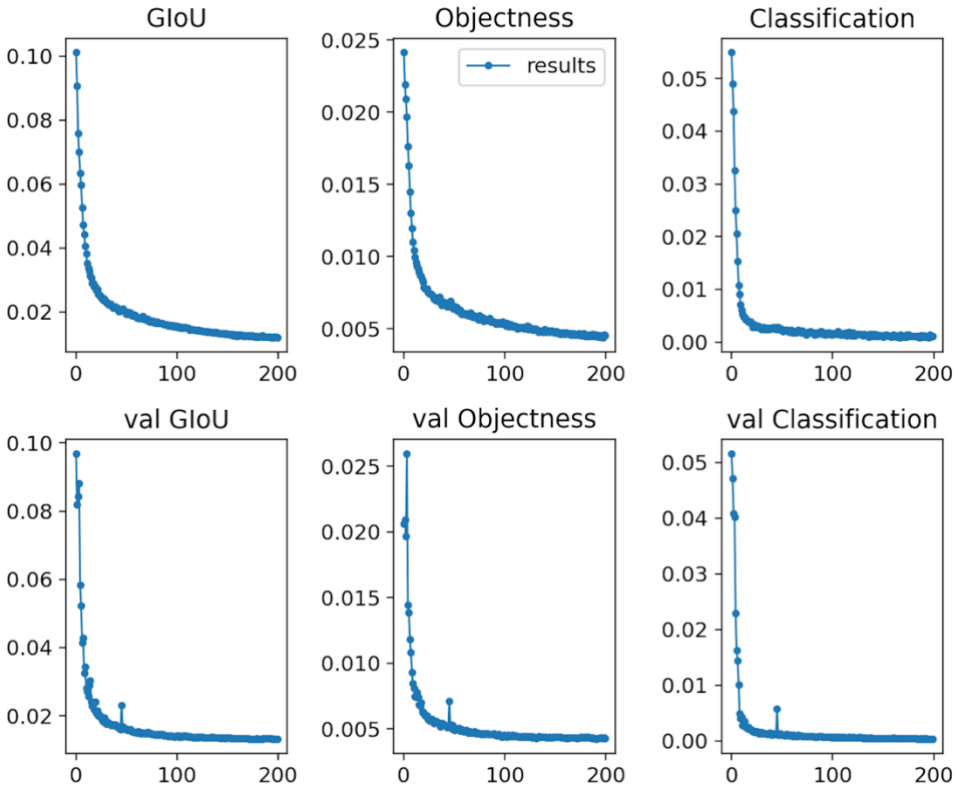

Figure 4.9 Three types of losses for YOLOv5

On the one hand, comparing to the results of the Faster R-CNN, there is no big difference between the losses of the Faster R-CNN and YOLOv5 except the loss score of Objectness. The loss of Objectness is around 0.005 which is much lower than the loss (0.025) achieved by the Faster R-CNN. The objectness loss is defined to evaluate how bad our model identifies the positions and object class during the processes of training and validation (Kong et al., 2017). In this case, the YOLOv5 gains a better result than the Faster R-CNN.

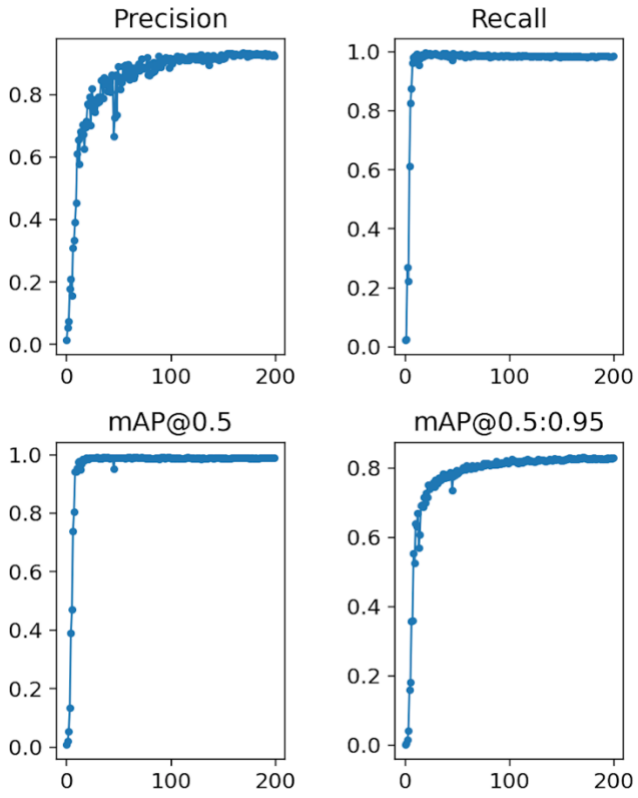

Figure 4.10 The metrics for evaluating the overall performance of YOLOv5

Figure 4.11 Several tested images with class index

On the other hand, Precision score is slightly lower that the Faster R-CNN but the curve of it has more fluctuation, which means the trend of convergence is not stable. In other words, we can say that the Faster R-CNN achieved a better prediction on recognizing the traffic signs in our dataset. Finally, there is no big difference in mAP scores between the two models. At the end of the experiment, we use some of images to test the overall performance of the YOLOv5. The details are shown in Figure 4.11 and 4.12.

Figure 4.12 Several tested images with confidence scores

Chapter 5
Analysis and Discussions

In this chapter, a comprehensive analysis and discussion will be conducted according to the experimental results for the two models. The pros and cons of the two models for traffic-sign recognition will be identified. Besides, the limitations of this project will be also clarified at the end.

5.1 Analysis and Discussion

After comparing the results of two models, we concluded that the Faster R-CNN has achieved a higher accuracy rate than the YOLOv5 for recognizing the traffic signs in NZ. The conclusion was summarized according to the loss scores and precision related metrics. The Faster R-CNN has achieved lower loss scores while gaining higher precision related scores, such as the consistent changing trend of Precision and Recall scores and mAP values.

However, in the testing phase, we noticed that the end-to-end model YOLOv5 is more efficient when it was processing the data of inference. The test video in the inference is composed of 2,074 frames. The processing time for per frame of the YOLOv5 is only around 0.011 seconds but the time consumption for the Faster R-CNN (37 seconds) is so much longer than the YOLOv5. From the perspective of time consumption, the YOLOv5 is a more reasonable choice for performing recognition tasks.

In summary, the Faster R-CNN is a much accurate model for recognizing traffic signs without considering the time consumption. The YOLOv5 is a better one when the tasks concern more about the data processing time.

5.2 Limitations of This Project

The limitations of this project are mainly reflected in four aspects. Firstly, our dataset does not cover all the classes of traffic signs in New Zealand, which is mainly due to the limited time and physical resources. Thus, only seven the most common used traffic signs are contained in our dataset for training networks for recognition tasks. The practicability of this project will suffer from the partial types of traffic-sign data and the project temporally stuck in the experimental phase.

Secondly, though the experiments in this project were performed both on the two-stage and on-stage neural networks (Faster R-CNN with VGG16 and YOLOv5), there are

still a lot of models that are worthy to be estimated so that a more comprehensive results will be summarized and the conclusions will be more instructional in this field.

Thirdly, there are only several metrics used to evaluate the performance of the two models. Several meaningful measures, such as F1 score and the areas under Precision-Recall curve, could be considered as the additional measures to help us choose appropriate models for recognizing traffic signs.

Lastly, we only explored this problem from a limited researching angle. There are a lot of real-world difficulties while recognizing traffic signs like we mentioned at the beginning of this report, including illuminant issue, rotations, partial occasions and physical damages, etc. However, in this project, we only focused on tackling with the problem of recognizing smaller size of traffic signs. There are still a lot of researching spaces in this field to be mined.

Chapter 6

Conclusion and Future Work

In this chapter, we will draw a conclusion for the project based on our experimental results and analysis. In addition, the future research directions will be pointed out.

6.1 Conclusion

One of the objectives of this report is to propose a customized benchmark for recognizing the traffic signs in NZ since there is no benchmarks that can fit into traffic-sign recognition tasks for all counties. Our dataset consists of 3,436 images in total and contains seven classes of traffic signs of NZ. The distribution of these classes is more even comparing to the most popular benchmark, German Traffic Sign Recognition Benchmark (GTSRB), which is an improvement directly contributed to the distinct performance of the two chosen models. Most importantly, we trained CNN models to recognize small size traffic signs, thus there are sufficient instances in smaller sizes in our dataset. The results of two models which performed based on our dataset are promising and impressive.

Another objective of this report is to evaluate the neural networks for this task. We evaluated the performance of a one-stage model (YOLOv5) and a two-stage model (Faster R-CNN with VGG16). According to the comparison between the two models, we concluded that the Faster R-CNN is a better option for TSR without considering the time consumption as the higher-level accuracy reached by it. YOLOv5 is more sufficient and important there is a slightly degrade of accuracy rate comparing to the Faster R-CNN.

6.2 Future Work

Recently, there is redundant research work emerging for recognizing traffic signs for handling with the real-world problems. On the one hand, in future, we will complete our benchmark by covering more types of the traffic signs in NZ so that we can make this project more instructional in this field. On the other hand, more object recognition techniques will be employed into TSR. For example, recognizing objects utilizes heatmaps methods. Finally, more evaluation measures also should be used to estimate the performance of different models.

References

Al-Qizwini, M., Barjasteh, I., Al-Qassab, H., & Radha, H. (2017). *Deep learning algorithm for autonomous driving using googlenet.* In IEEE Intelligent Vehicles Symposium (IV).

Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2017). *Understanding of a convolutional neural network.* In International Conference on Engineering and Technology (ICET).

Bochinski, E., Senst, T., & Sikora, T. (2017). *Hyper-parameter optimization for convolutional neural network committees based on evolutionary algorithms.* In IEEE International Conference on Image Processing (ICIP).

Chen, R.-C. (2019). Automatic license plate recognition via sliding-window DarkNet-YOLO deep learning. *Image and Vision Computing, 87*, 47-56.

Dalal, N., & Triggs, B. (2005). *Histograms of oriented gradients for human detection.* In IEEE Conference on Computer Vision and Pattern Recognition (CVPR'05).

Everingham, M., Van Gool, L., Williams, C. K., Winn, J., & Zisserman, A. (2010). The pascal visual object classes (VOC) challenge. *International Journal of Computer Vision, 88*(2), 303-338.

Géron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to build intelligent systems*. O'Reilly Media.

Giusti, A., Cireşan, D. C., Masci, J., Gambardella, L. M., & Schmidhuber, J. (2013). *Fast image scanning with deep max-pooling convolutional neural networks.* In IEEE International Conference on Image Processing.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). *Deep residual learning for image recognition.* In IEEE Conference on Computer Vision and Pattern Recognition.

Henderson, P., & Ferrari, V. (2016). *End-to-end training of object class detectors for mean average precision.* In Asian Conference on Computer Vision.

Hong-meng, L., Di, Z., & Xue-bin, C. (2017). Deep learning for early diagnosis of Alzheimer's disease based on intensive AlexNet. *Computer Science, 44*(6), 50-59.

Jégou, S., Drozdzal, M., Vazquez, D., Romero, A., & Bengio, Y. (2017). *The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation.* In IEEE Conference on Computer Vision and Pattern Recognition Workshops.

Jin, K. H., McCann, M. T., Froustey, E., & Unser, M. (2017). Deep convolutional neural network for inverse problems in imaging. *IEEE Transactions on Image Processing, 26*(9), 4509-4522.

Kong, T., Sun, F., Yao, A., Liu, H., Lu, M., & Chen, Y. (2017). *Ron: Reverse connection with objectness prior networks for object detection.* In IEEE Conference on Computer Vision and Pattern Recognition.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *Imagenet classification with deep convolutional neural networks.* In Advances in Neural Information Processing Systems.

Kuo, W., Hariharan, B., & Malik, J. (2015). *Deepbox: Learning objectness with convolutional networks.* In IEEE International Conference on Computer Vision.

Larsson, F., & Felsberg, M. (2011). *Using Fourier descriptors and spatial models for traffic sign recognition.* In Scandinavian Conference on Image Analysis.

Liang, M., & Hu, X. (2015). *Recurrent convolutional neural network for object recognition.* In IEEE Conference on Computer Vision and Pattern Recognition.

Liang, M., Yuan, M., Hu, X., Li, J., & Liu, H. (2013). *Traffic sign detection by ROI extraction and histogram features-based recognition.* In International Joint Conference on Neural Networks (IJCNN).

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017). *Feature pyramid networks for object detection.* In IEEE Conference on Computer Vision and Pattern Recognition.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017). *Focal loss for dense object detection.* In IEEE International Conference on Computer Vision.

Liu, B., Zhao, W., & Sun, Q. (2017). *Study of object detection based on Faster R-CNN.* In Chinese Automation Congress (CAC).

Liu, S., Qi, L., Qin, H., Shi, J., & Jia, J. (2018). *Path aggregation network for instance segmentation.* In IEEE Conference on Computer Vision and Pattern Recognition.

Maas, A. L., Hannun, A. Y., & Ng, A. Y. (2013). *Rectifier nonlinearities improve neural network acoustic models.* In ICML.

Malik, R., Khurshid, J., & Ahmad, S. N. (2007). *Road sign detection and recognition using colour segmentation, shape analysis and template matching.* In International Conference on Machine Learning and Cybernetics.

Mao, X., Hijazi, S., Casas, R., Kaul, P., Kumar, R., & Rowen, C. (2016). *Hierarchical CNN for traffic sign recognition.* In IEEE Intelligent Vehicles Symposium (IV).

Mogelmose, A., Trivedi, M. M., & Moeslund, T. B. (2012). Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey. *IEEE Transactions on Intelligent Transportation Systems, 13*(4), 1484-1497.

Nagi, J., Ducatelle, F., Di Caro, G. A., Cireşan, D., Meier, U., Giusti, A., . . . Gambardella, L. M. (2011). *Max-pooling convolutional neural networks for vision-based hand gesture recognition.* In IEEE International Conference on Signal and Image Processing Applications (ICSIPA).

Park, J.-G., & Kim, K.-J. (2013). Design of a visual perception model with edge-adaptive

Gabor filter and support vector machine for traffic sign detection. *Expert Systems with Applications, 40*(9), 3679-3687.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). *You only look once: Unified, real-time object detection.* In IEEE Conference on Computer Vision and Pattern Recognition.

Ren, S., He, K., Girshick, R., & Sun, J. (2015). *Faster R-CNN: Towards real-time object detection with region proposal networks.* In Advances in Neural Information Processing Systems.

Rezatofighi, H., Tsoi, N., Gwak, J., Sadeghian, A., Reid, I., & Savarese, S. (2019). *Generalized intersection over union: A metric and a loss for bounding box regression.* In IEEE Conference on Computer Vision and Pattern Recognition.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., . . . Bernstein, M. (2015). Imagenet large scale visual recognition challenge. In *International Journal of Computer Vision, 115*(3), 211-252.

Scherer, D., Müller, A., & Behnke, S. (2010). *Evaluation of pooling operations in convolutional architectures for object recognition.* In International Conference on Artificial Neural Networks.

Sermanet, P., & LeCun, Y. (2011). *Traffic sign recognition with multi-scale convolutional networks.* In International Joint Conference on Neural Networks.

Stallkamp, J., Schlipsing, M., Salmen, J., & Igel, C. (2011). *The German traffic sign recognition benchmark: A multi-class classification competition.* In International Joint Conference on Neural Networks.

Stallkamp, J., Schlipsing, M., Salmen, J., & Igel, C. (2012). Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks, 32*, 323-332.

Suárez-Paniagua, V., & Segura-Bedmar, I. (2018). Evaluation of pooling operations in convolutional architectures for drug-drug interaction extraction. *BMC bioinformatics, 19*(8), 39-47.

Sun, M., Song, Z., Jiang, X., Pan, J., & Pang, Y. (2017). Learning pooling for convolutional neural network. *Neurocomputing, 224*, 96-104.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). *Rethinking the inception architecture for computer vision.* In IEEE conference on computer vision and pattern recognition.

Tan, M., Pang, R., & Le, Q. V. (2020). *Efficientdet: Scalable and efficient object detection.* In IEEE/CVF Conference on Computer Vision and Pattern Recognition.

Valueva, M. V., Nagornov, N., Lyakhov, P. A., Valuev, G. V., & Chervyakov, N. I. (2020). Application of the residue number system to reduce hardware costs of the convolutional neural network implementation. *Mathematics and Computers in Simulation, 177*, 232-243.

Wang, C.-Y., Mark Liao, H.-Y., Wu, Y.-H., Chen, P.-Y., Hsieh, J.-W., & Yeh, I.-H. (2020). *CSPNet: A new backbone that can enhance learning capability of CNN.* In IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops.

Wang, G., Ren, G., & Quan, T. (2013). *A traffic sign detection method with high accuracy and efficiency.* In International Conference on Computer Science and Electronics Engineering.

Wang, G., Ren, G., Wu, Z., Zhao, Y., & Jiang, L. (2014). A fast and robust ellipse-detection method based on sorted merging. *The Scientific World Journal*.

Wu, S., Li, X., & Wang, X. (2020). IoU-aware single-stage object detector for accurate localization. *Image and Vision Computing*, 103911.

Wu, Y., Liu, Y., Li, J., Liu, H., & Hu, X. (2013). *Traffic sign detection based on*

convolutional neural networks. In International Joint Conference on Neural Networks (IJCNN).

Wu, Z., Shen, C., & Van Den Hengel, A. (2019). Wider or deeper: Revisiting the resnet model for visual recognition. *Pattern Recognition, 90*, 119-133.

Xie, L., Wang, J., Wei, Z., Wang, M., & Tian, Q. (2016). *Disturblabel: Regularizing CNN on the loss layer.* In IEEE Conference on Computer Vision and Pattern Recognition.

Xie, S., Girshick, R., Dollár, P., Tu, Z., & He, K. (2017). *Aggregated residual transformations for deep neural networks.* In IEEE Conference on Computer Vision and Pattern Recognition.

Xu, L., Ren, J. S., Liu, C., & Jia, J. (2014). *Deep convolutional neural network for image deconvolution.* In Advances in Neural Information Processing Systems.

Yang, Y., Luo, H., Xu, H., & Wu, F. (2015). Towards real-time traffic sign detection and classification. *IEEE Transactions on Intelligent Transportation Systems, 17*(7), 2022-2031.

Yang, Y., & Wu, F. (2014). *Real-time traffic sign detection via color probability model and integral channel features.* In Chinese Conference on Pattern Recognition.

Yao, C., Wu, F., Chen, H.-j., Hao, X.-l., & Shen, Y. (2014). *Traffic sign recognition using HOG-SVM and grid search.* In International Conference on Signal Processing (ICSP).

Zaklouta, F., & Stanciulescu, B. (2014). Real-time traffic sign recognition in three stages. *Robotics and Autonomous Systems, 62*(1), 16-24.

Zhang, H., Goodfellow, I., Metaxas, D., & Odena, A. (2019). *Self-attention generative adversarial networks.* In International Conference on Machine Learning.

Zhang, J., Huang, M., Jin, X., & Li, X. (2017). A real-time Chinese traffic sign detection

algorithm based on modified YOLOv2. *Algorithms, 10*(4), 127.

Zhu, Y., Zhang, C., Zhou, D., Wang, X., Bai, X., & Liu, W. (2016). Traffic sign detection and recognition using fully convolutional network guided proposals. *Neurocomputing, 214*, 758-766.

Zhu, Z., Liang, D., Zhang, S., Huang, X., Li, B., & Hu, S. (2016). *Traffic-sign detection and classification in the wild.* In IEEE Conference on Computer Vision and Pattern Recognition.