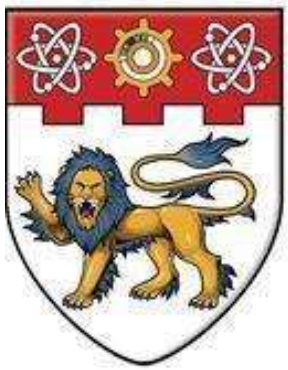
NANYANG TECHNOLOGICAL UNIVERSITY

SCHOOL OF COMPUTER SCIENCE



NANYANG TECHNOLOGICAL UNIVERSITY

Project Title:

Traffic Sign Detection System with Efficient and Low-bit Neural Network Compression In Computer Vision

THIN LAT HAN

(U1822997F)

2022

Table of Content

Abstract

In a traffic setting, traffic sign detection is a critical task. It is becoming even more important as the number of vehicles on the road grows, increasing the accident rate.

Object detection in embedded devices can be used to alert distracted drivers to traffic signs so that the signs can direct them on the road. Object identification on images can have a high level of accuracy and speed, which is required by traffic sign detection system to recognize traffic signs quickly and precisely so that road safety can be ensured. However, to achieve the high accuracy and speed requirements, it usually requires a high-performing Graphics Processing Unit (GPU) and considerable internal storage. Unfortunately, most embedded devices do not have high performing GPU and huge internal storage.

Thus, this project proposes the use of efficient and low-bit neural network compression to make existing object detection model deployable on embedded devices which ensuring that the accuracy and speed is not too compromised.

Using the approach, three models are proposed. Each has a distinct advantage. Version 2 was the best model, with some compromise on the models's accuracy. It has a fast-processing time of 47.39ms and a modest file size of 7.37MB, as well as a high confidence of roughly 97 percent when detecting several traffic signs on an image.

The full project can be found in https://github.com/thinlathan/Traffic-Sign-Detection-System

Acknowledgements

I would like to convey my gratitude to Professor Loke Yuan Ren, who has provided me with constant feedback and support during my FYP project.

Also, my family and friends deserve special recognition for their emotional support throughout this project.

Your assistance will always be treasured.

Thin Lat Han

April 2022

Acronyms

AP	Average Precision
BFLOPS	Billion Floating-Point Operations Per Second
BNN	Binarized Neural Networks
CSPNet	Cross Stage Partial Network
FPS	Frame Per Second
GPU	Graphics Processing Unit
IoU	Intersection over Union
mAP	Mean Average Precision
MS	
COCO	Microsoft Common Objects in Context
PAN	Path Aggregation Network
SPP	Spatial Pyramid Pooling
STE	Straight-Through Estimator
SWAR	Single instruction, multiple data Within A Register
YOLOv4	You Only Look Once version 4

List of Figures

List of Tables

Chapter 1: Introduction

1.1. Background and Motivation

Due to global economic expansion, vehicle ownership per person has expanded dramatically in many countries, leading in an increase in the number of automobiles on the road. The frequency of accidents increases as the route becomes more congested [1]. As a result, reducing car accidents caused by drivers becomes extremely relevant. There are numerous reasons of car accidents, but statistics show that distracted driving is the top cause [2]. Distraction frequently leads to a failure to obey traffic signs, which results in a car collision because the driver failed to notice the information offered by the traffic signs [3].

Traffic signs are used to inform and guide drivers, as well as to control traffic flow. Many people, however, overlook traffic signs due to distractions or physical issues (such as poor eyesight or health conditions). As a result, traffic signs fail to warn and guide cars, leading to accidents. If the car is properly directed by traffic signs, such an issue can be easily rectified. Thus, a traffic sign detection system should be developed to detect traffic signals on behalf of drivers, allowing them to be notified even if they are unaware of the signs.

The traffic sign detection system should include the following features. First , it must be real-time in order to recognize the traffic signs promptly. To be termed real-time, it must process images at a rate of roughly 30 frames per second [citation]. Second, it must be capable of running on embedded devices like cell phones. This necessitates low power consumption, low memory utilization, and low computational power.

Despite their great accuracy, current state-of-the-art real-time detection models are unable to run on embedded devices due to the high computational power required.

Binary-weight and XNOR network models, on the other hand, reveal that they can cut memory use by 32x and increase computing efficiency by 58x [4]. It sounds appealing from a theatrical standpoint. However, because only a few studies have looked at it, it has yet to be confirmed.

1.2. Project Objectives and Scope

The project's goal is to develop an effective real-time traffic sign detecting system that can run on an embedded system. This project will investigate how to adapt an existing object detection algorithm to a traffic sign detection system. The system will then be improved by binarizing model parameters while assuring backpropagation is still feasible. The system should have reduced power consumption, improved memory efficiency, and lower CPU utilization after the enhancement.

1.3. Report Structure

Chapter 1 (Introduction) discusses the necessity for a traffic sign detection system and the system's specifications. The purpose of the project is also stated.

Chapter 2 (Literature Review) examines the literature in relation to the project's several knowledge domains. The reviews' main subjects include object detection and binarized neural networks.

Chapter 3 (Implementation) describes how each project section is implemented.

Chapter 4 (Results and Discussion) show the outcomes of the system implementation, along with a discussion of the results based on a comparison to the baseline system.

Chapter 5 (Conclusion) summarizes the project and identifies the knowledge gained. It also looks at different ways that could be used in the future.

Chapter 2: Literature Review

This chapter will be split into two parts - object detection and binarized neural network.

2.1. Object Detection

When we look at objects in images or videos, we can recognize and locate them almost instantaneously. Object detection is the computer vision approach to replicate this intelligence.

2.1.1. Different Approaches of Object Detection

Object detection can be accomplished using two different approaches. The first method is known as "traditional computer vision," and it is a two-stage non-neural network method. To execute object detection, it must first identify an arbitrary number of zones matching to the number of objects using well-known feature descriptors (e.g., SIFT, SURF). The defined region is then classified using techniques such as support vector machine (SVM). The second method is a one-stage neural network-based approach. Instead of dividing zoning and classification into two separate processes while doing object detection, it could execute a one-stage end-to-end object detection.

Studies reveal that the one-stage neural network-based approach outperforms the two-stage non-neural network-based approach for object detection. While it takes more time and images to train the model, it can recognize items more correctly and with a broader range of objects [5].Thus, this project’s approach was focused on one-stage neural network-based approach.

2.1.2. Different types of neural network-based object detection

There are a variety of neural network-based object detection algorithm. Among the state-of-the-art object detectors, 'You Only Look Once version 4' (YOLOv4) was the most outstanding one (Figure 1) with high average precision (AP) and high frame per second (FPS) on Nvidia V100 dataset.

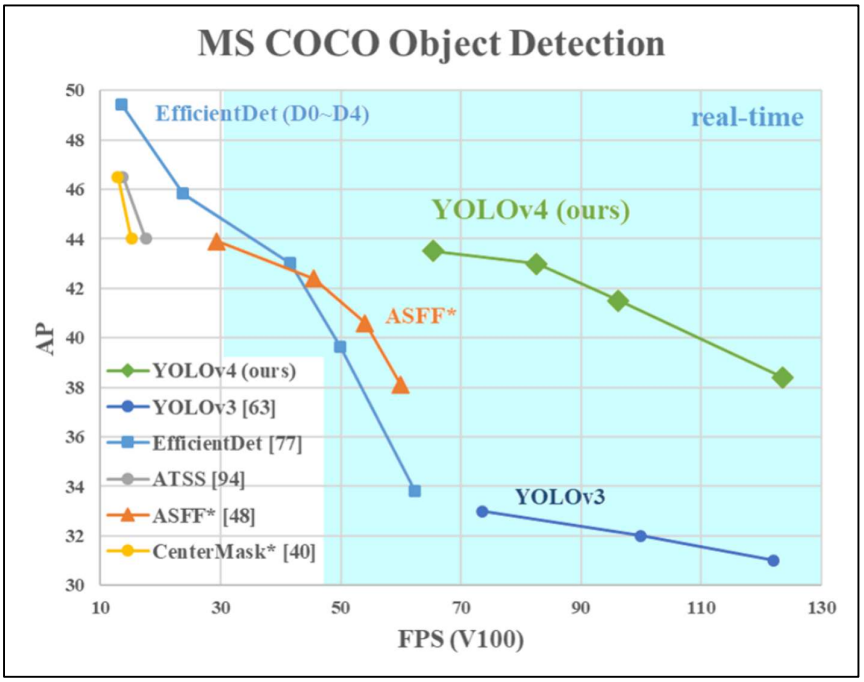


Figure 1: AP against FPS(V100) for a variety of neural network-based object detection algorithm

YOLOv4 also succeeds in all three charts from the renowned Microsoft Common Objects in Context (MS COCO) object detection dataset [6]. YOLOv4 came in fourth on the Mean Average Precision (MAP) chart, with the fourth highest MAP (Figure 2). YOLOv4 is at the top of the FPS table, with the highest FPS (Figure 3). YOLOv4 is at the top of the Interference time ranking, with the shortest interference time (Figure 4)

Rank	Model	MAP	FPS	inference time, ms	Extra Training Data	Paper	Code	Result	Year	Tags
1	YOLOR-D6	57.3	34		×	You Only Learn One Representation: Unified Network for Multiple Tasks			2021	YOLO
2	YOLOR-E6	56.4	45		×	You Only Learn One Representation: Unified Network for Multiple Tasks			2021	single scale YOLO
3	YOLOR-W6	55.5	66		×	You Only Learn One Representation: Unified Network for Multiple Tasks			2021	single scale YOLO
4	YOLOv4-CSP-P7	55.4	16	63	×	Scaled-YOLOv4: Scaling Cross Stage Partial Network			2020	single scale YOLO
5	EfficientDet-D7x (single-scale)	55.1		153	×	EfficientDet: Scalable and Efficient Object Detection			2019	single scale

Figure 2: YOLOv4 came fourth on MAP chart on MS COCO.

Rank	Model	MAP	FPS	inference time, ms	Extra Training Data	Paper	Code	Result	Year	Tags
1	YOLOv4-512	43	83	12	✓	YOLOv4: Optimal Speed and Accuracy of Object Detection			2020	single scale YOLO
2	YOLOv4-CSP CD53s 640	47.5	73		×	Scaled-YOLOv4: Scaling Cross Stage Partial Network			2020	YOLO
3	YOLOR-W6	55.5	66		×	You Only Learn One Representation: Unified Network for Multiple Tasks			2021	single scale YOLO
4	YOLOv4-608	43.5	62	16	×	YOLOv4: Optimal Speed and Accuracy of Object Detection			2020	single scale YOLO
5	CSPResNeXt50-PANet-SPP	33.4	58	17	✓	CSPNet: A New Backbone that can Enhance Learning Capability of CNN			2019	

Figure 3 : YOLOv4 came first on FPS chart on MS COCO.

Rank	Model	MAP	FPS	inference time, ms	Extra Training Data	Paper	Code	Result	Year	Tags
1	YOLOv4-512	43	83	12	✓	YOLOv4: Optimal Speed and Accuracy of Object Detection			2020	single scale YOLO
2	YOLOv4-608	43.5	62	16	×	YOLOv4: Optimal Speed and Accuracy of Object Detection			2020	single scale YOLO
3	ParNet-XL-CSP	48.0		16.4	×	Non-deep Networks			2021	
4	CSPResNeXt50-PANet-SPP	33.4	58	17	✓	CSPNet: A New Backbone that can Enhance Learning Capability of CNN			2019	
5	TTFNet	35.1	54.4	18.4	×	Training-Time-Friendly Network for Real-Time Object Detection			2019	

Figure 4: YOLOv4 came first on FPS chart on MS COCO.

Additionally, YOLOv4 could be trained and used on standard GPU with 8 to 16 GB-VRAM, demonstrating that does not require a lot of memory and GPU [7]. For the traffic sign detection system, this is desirable. Hence, this project's approach was focused on adapting the YOLOv4 algorithm.

2.1.3. YOLOv4

Due to its remarkable speed, YOLOv4 is one of the most advanced real-time object detection algorithms available [7]. Figure 5 depicts the architecture of one stage object detector. YOLOv4 uses CSPDarknet53 as a backbone, Spatial Pyramid Pooling (SPP) and Path Aggregation Network (PAN) as the neck, and YOLOv3 as head.

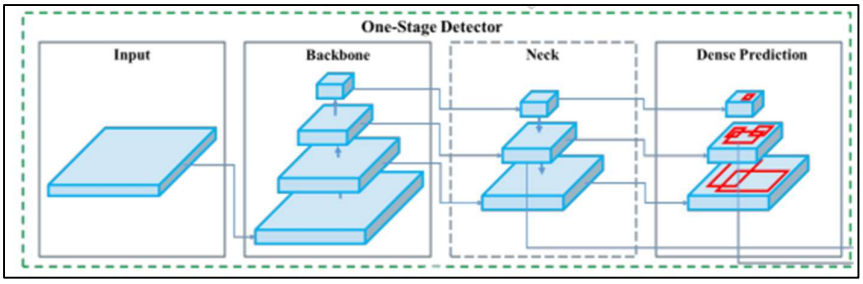


Figure 5: Architecture of one stage object detector.

c is an optimization strategy for achieving a richer gradient combination while decreasing computation [8]. The feature map of the base layer is divided into two components, which are subsequently combined using the cross-stage hierarchy (Figure 6). The split and merge approach allows for more gradient flow through the network, resulting in a richer gradient combination.

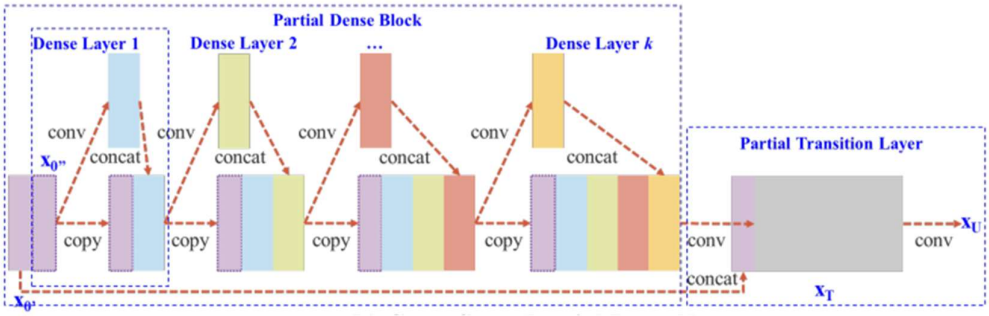


Figure 6: Architecture of CSPNet.

SPP (Spatial Pyramid Pooling) is a method for removing the network's fixed-size limitation, allowing the model to run without fixed-size input images (Figure 7). It is applied on top of the last convolution layer. With SPP, the feature map is computed only once for the entire image. Then, SPP pools features to create a fixed-length representation, which will be supplied into the fully linked layers for training. Thus, with SPP, the convolution features is avoided from being computed multiple times with slow down the speed [9].

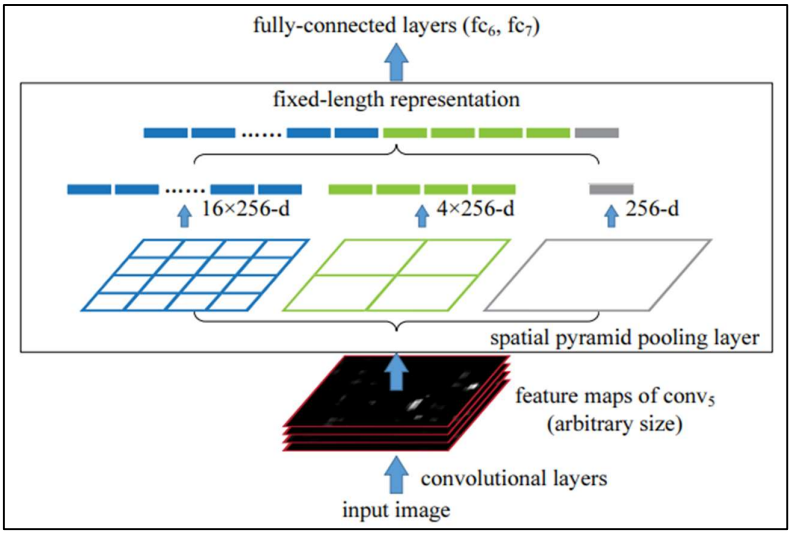


Figure 7: Architecture of SPP method.

PAN (Path Aggregation Network) attempts to improve information flow in proposal-based instance segmentation framework. The PAN framework is shown in Figure 8.

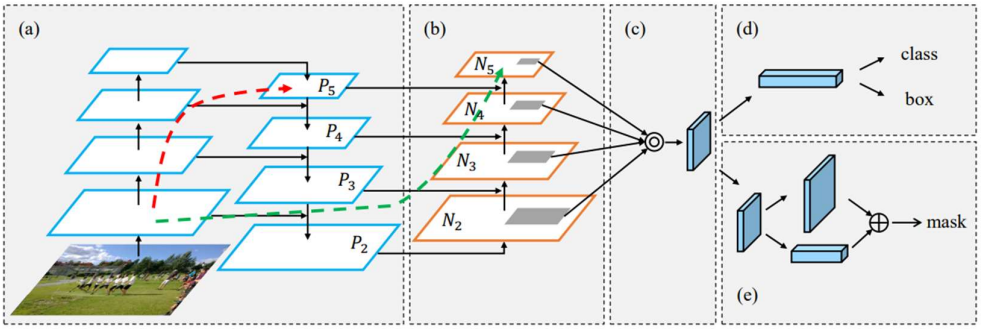


Figure 8: PAN framework. (a) FPN backbone. (b) Bottom-up path augmentation. (c) Adaptive feature pooling. (d) Box branch. (e) Fully connected fusion.

The bottom-up path augmentation improves the overall feature hierarchy with accurate localization signals in lower levels. Adaptive feature pooling connects the feature grid and all feature levels, allowing important information from each feature level to cascade directly to the proposal subnetwork next. Also, to improve mask prediction, a complementary branch recording different views for each proposal is constructed [10].

PAN enhances information flow with minimal additional processing overhead using this approach. It's also a well-recognised framework, having placed first in the COCO 2017 Challenge Instance Segmentation problem and second in the Object Detection task without large-scale training [11].

In a nutshell, YOLOv4 is one of the most powerful real-time object identification algorithms accessible, thanks to these frameworks. Hence, it was used in this project to create a traffic light detection system.

2.2. Binarized Neural Networks

2.2.1. Overview of BNN

Most deep learning applications (such as object detection) are run on the cloud rather than on a device. This is because it necessitates a great amount of computational, memory, and storage resources to suit its massive network parameter. As a result, BNN is presented to fulfil the desire for a hardware-friendly neural network structure with high intensity. Since BNN has fewer size parameters and a faster interference speed, it is suitable for use on resource-constrained embedded systems [12].

BNN is a form of neural network which is similar to CNN but with the single-bit activation and weights in all hidden layers. Binarization techniques (Figure 9) are used to reduce 32-bit numbers to a single-bit value. Binarization uses the XNOR and pop count operations to save model storage and to reduce matrix computing costs [4].

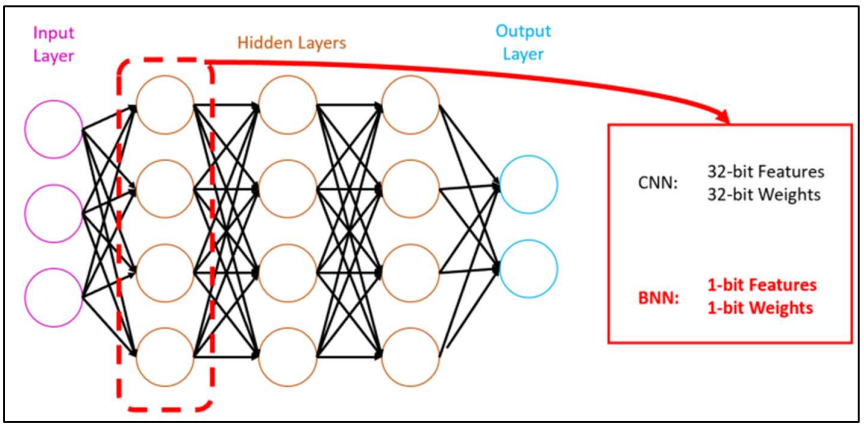


Figure 9: Difference between CNN and BNN neural network.

A neural network (NN) is composed of 2 processes: forward and backward propagation.

2.2.2. BNN forward propagation

Forward propagation, also known as model interference, is the process of moving from input layer to output layer. A neural cell is a basic computation structure in a neural network's forward path. BinaryNet [13] proposed that in BNN, both activation and weight of neural cell are binarized to +1 or -1 (Figure 10). This make is possible to store 1 pixel into single bit register, hence, increasing storage efficiency.

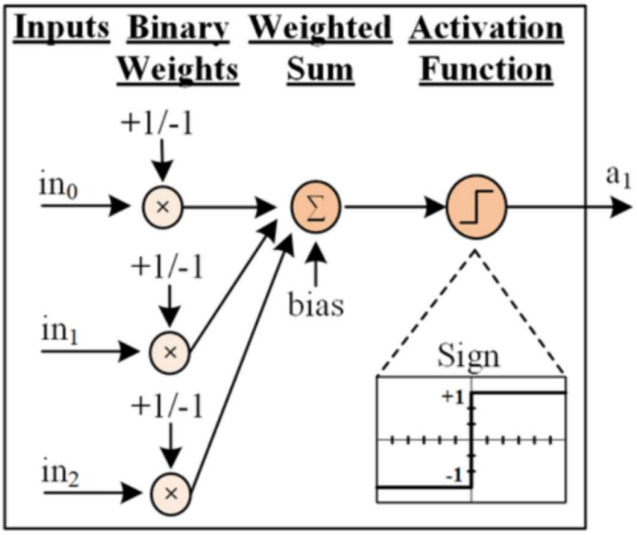


Figure 10: BNN constrain both weight and activation to +1 or -1.

It also proposed the method SWAR (Single instruction, multiple data Within A Register) [13]. SWAR concatenates 32-bit binary variables into 32-bit registers, resulting in a bitwise XNOR speedup of x32. With three SWAR instructions, 32 connections can be assessed as this equation: $result \mathrel{+}= popcount(XNOR(a_0^{32}, w_0^{32}))$

In the equation, result is the resulting weighted sum, a_0^{32} is the concatenated input and w_0^{32} is the concatenated output. As a result, there are only 3 instructions involved (accumulation, popcount and xnor) which require 6 (1+4+1) instructions. This results in a speed increase of 32/6 = 5.3. As a result, both the power efficiency and the speed of operation are enhanced.

2.2.3. BNN backward propagation

Backward propagation is the process of moving from output layer to input layer. Its objective is to fine-tune the weights of the model. Because the derivation result of the binarization function is 0, binary weights cannot be learned using the usual gradient descent method.

In BNN, straight-through estimator (STE) is used to learn binary weights in backward propagation [13] as depicted in Figure 11. During BNN training steps, each layer's actual weights are kept and updated using STE. After training, binarized weights are saved and actual weights are discarded.

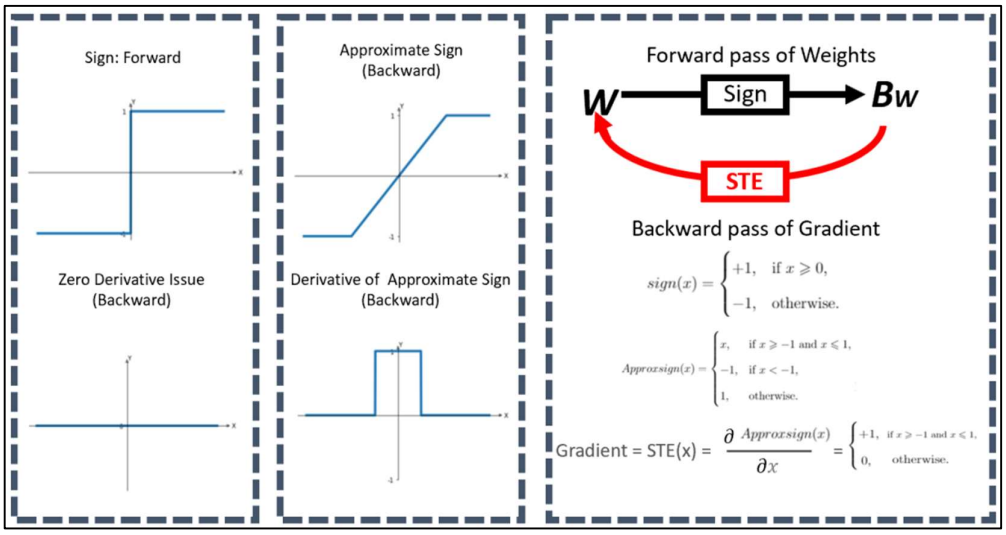


Figure 11: BNN's Backward propagation

2.2.4. Advantage of BNN

First, BNN is memory efficient. As mentioned in 2.2.2, BNN uses binarization operation to convert 32 bits floating data into single bit binary data. Hence, memory size is reduced by x32.

Second, BNN is incredibly fast. Since both activation and weight are binarized, BNN uses bitwise XNOR operation, as indicated in 2.2.2. Because the main operation of a neural network is multiply-accumulation, a speed gain of 5.3 can be achieved for every multiply-accumulation operation by using bitwise XNOR. Additionally, floating point multiply-accumulation needs 200 FPGA slices in FPGA implementation, but bitwise XNOR and population count only take 1 slice [13]. This demonstrates that BNN is suitable for use on embedded devices.

Third, BNN is power efficient. According to Horowitz [14], different type of data's operation consumes different amounts of energy (table 1), and different memory sizes consume variable amounts of energy when accessed (table 2). As BNN reduce memory size of activation and weight through binarization, power usage in operation and memory access decreases.

Operation	Multiply/pJ	Addition/pJ
8-bit integer	0.22	0.03
32-bit integer	3.1	0.1
16-bit floating point	1.1	0.4
32-bit floating point	3.7	0.9

Table 1: Energy consumption of multiply-accumulations.

Memory size	64-bit memory access
8K	10pJ
32K	20pJ
1M	100pJ
DRAM	1.3-2.6pJ

Table 2: Energy consumption of memory accesses.

Chapter 3: Implementation

3.1. Dataset for YOLO

To train the project's desired dataset (traffic sign) with YOLOv4, the dataset must be modified to conform to the YOLO format [15].

Each image in the dataset must be accompanied by a label file(txt format) with the same name. A single bounding box of an object in the image is represented by each row entry in the label file. The box's information is contained in the row entry: <object-class-id> <center-x> <center-y> <width><height>.

The first field <object-class-id> is an integer that represents the object's class. The second <center-x> and third <center-y> fields are the x and y coordinates of the bounding box's center, normalized by the image width and height. The fourth <width> and fifth <height> fields represent the bounding box's width and height, normalized by the image width and height.

Annotation can be done using python scripts that generate label text files for each image based on known information about position of the bounding boxes. Alternatively, to create a label for each image, open-source GUI applications like Yolo_Label [16] can be used.

Then, 3 files need to be created: classes.name, train.txt and test.txt. Each row in classes.name should correspond to an object class name, and the index should correspond to the field <object-class-id> in the label text file. The train.txt and test.txt consist of the file paths for the training and testing images. Each row in the files is formatted as “image path>”.

Initially, I attempted to use YOLO Label to annotate each image to adjust the dataset to comply to the YOLO format. However, considering Dr Loke's advice, I concluded that it

was impossible given the project's time limitations and the massive quantity of picture annotation required due to the sheer volume of photos required for training. Hence, I opted to use the Traffic Signs Dataset in Kaggle [17], which had already been converted to YOLO format.

This dataset contains 741 YOLO-formatted photos with associated label text files. Its images are divided into four classes: prohibitory, danger, mandatory and other.

3.2. Data pre-processing

First, data augmentation was implemented to the dataset using Clodsa [18]. It is an open-source image augmentation toolkit that supports a wide range of augmentation techniques and allows users to mix and match them effortlessly. Data augmentation techniques used are horizontal and vertical flip, rotation, average blurring and raise hue. With data augmentation, the number of images available in the dataset increases by 5 times from 741 to 3705. Next, images in the dataset are binarized and the pixel domain is normalized from 0 to 1. Finally, the test.txt and train.txt files are generated. The ratio of the test image to the training image is 1:9. The implementation can found in the notebook named “Image_preprocessing.ipynb”.

3.3. Experimental Environment

The Google Colaboratory [19] was chosen as the test environment. It's a free, cloud based Jupyter notebook environment that supports machine learning on CPUs, GPUs, and TPUs. Most crucially, Google Collaboratory places no restrictions on the user's PC while also providing GPUs and TPUs to speed up the training process, which can easily take over 5 hours even with GPUs.

3.4. Model Implementation

The original YOLOv4 model is being trained using the MS COCO dataset, which contains 80 classes. The classes, however, do not contain traffic signs. As a result, YOLOv4 must be trained using a customized dataset to recognise traffic signs. "Yolov4.conv.137" is used instead of training the model from scratch. It's a pre-trained YOLOv4 weight with 137 convolution layers of training.

Training the model from scratch without pre-trained weight will necessitate a large dataset and take a long period. However, because extracted features from images (e.g., circles, lines) are similar, the feature extraction part of the pre-trained weight can be used. As a result, because the pre-trained weight has already been optimized, models can converge faster with optimised feature extraction in place.

3.5. Different trained models

Different models are trained using different parameters to compare what is the best parameter for traffic sign detection system.

The model “version 1” is trained using YOLOv4 without image augmentation and Yolov4 configuration of width and height = 416.

The model “version 2” is trained using YOLOv4 with image augmentation and Yolov4 configuration of width and height = 320.

The model “version 2” is trained using YOLOv4 with image augmentation, changing input image to binary, and Yolov4 configuration of width and height = 416.

3.6. Model Evaluation

After training, the models must be evaluated, and different separate measuring methods must be employed to demonstrate the trained models' performance. The

following methods were utilized to measure the performance: image processing time, workspace size, total BFLOPS, mAP@0.5 on VOC(IOU 0.50) and confidence of model.

Image processing time is a measurement of the speed of the model. It is the amount of time it takes to perform detect objects in an image.

Workspace size is a measurement of memory efficiency. It represents the amount of memory space required for the object detection on an image.

BFLOPS (billion floating-point operations per second) is a measurement of computer performance. It is a unit of measurement for the speed of a computer in operation per second, particularly for floating-point arithmetic operations.

Mean average precision (mAP) is a popular metric to measure the accuracy of object detectors. It is calculated by taking the mean AP (average precision) over all classes and/or overall IoU (intersection over union) thresholds, depending on different detection challenges that exist. IOU is the ratio of the predicted and ground truth bounding boxes' area of intersection and area of union, as shown in Figure 12. In the metric mAP@0.5 (which is utilized as a performance metric in Chapter 4), IOU was set to 0.5.

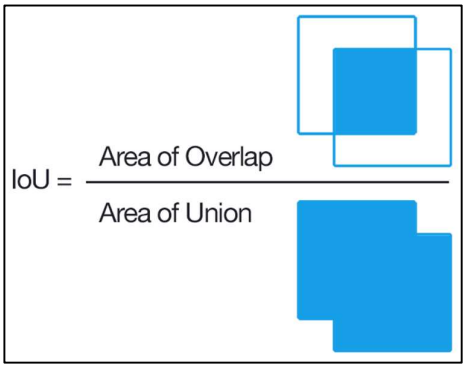


Figure 12: Definition of IOU.

The percentage confidence is used to determine the model's accuracy. It measures the confidence of the model when the model detected an object as a particular class.

Chapter 4: Results and Discussion

4.1. Quantitative Results

Model	mAP@0.5 (IOU 0.50)	Image processing time	Workspace size	Total BFLOPS
Version 1	96.82%	100.281ms	12.46MB	59.585
Version 2	65.53%	47.39ms	7.37MB	35.257
Version 3	44.13%	104.46ms	12.46MB	59.585

Table 3: Accuracy, speed, and size of different version of the models

Model	Confidence when detecting 1st sign	Confidence when detecting 2nd sign
Version 1	95%	99%
Version 2	95%	98%
Version 3	99%	96%

Table 4: Confidence when detecting traffic sign in image of the models

Table 3 shows the comparison in terms of accuracy (mAP0.5), image processing time, workspace size and number of BFLOPS. Table 4 shows the comparison in terms of confidence level when detecting traffic signs using the trained model.

From Tables 3 and 4, it is evident that model "Version 2" provides the greatest results. Despite the fact that the mAP@0.5 is slower than Version 1, it offers significant benefits in terms of image processing time, CPU usage (BFLOPS), and memory (size of model). As a result, the compromise is acceptable.

Version 3 is anticipated to yield the best results initially since its input photos are binarized into black and white images before training. Its performance, however, was the poorest, contrary to expectations.

4.2. Qualitative result

The models can be used to predict traffic signs on images (Figure 13) and a bounding box with label/class name will be displayed above the bounding box.

Figure 13: Traffic sign detection on image

The model can also be used to predict traffic signs in video (Figure 14). Figure 14 shows that, even though the traffic signals are far away and appear too small to be perceived by human eyes, the model was able to detect them. This shows the model's accuracy in detecting small objects, like far away traffic sign.

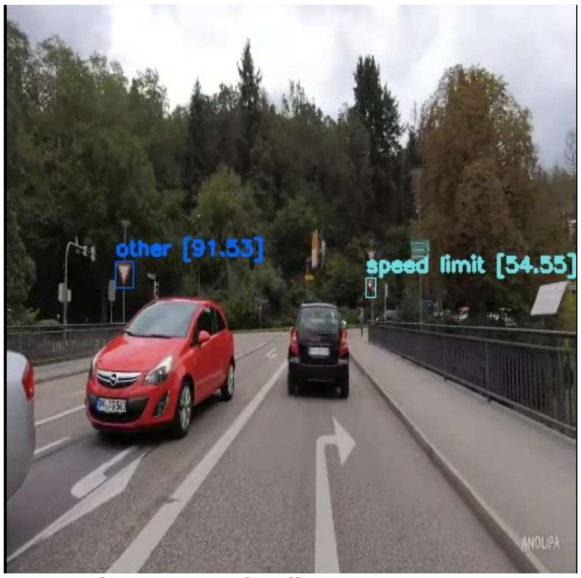


Figure 14: Screenshot of traffic sign detection on a video.

Chapter 5: Conclusion

5.1. Findings

The project demonstrates the feasibility of a traffic detection system that uses efficient and low-bit neural network compression in computer vision. The system can meet the requirements for both rapid and accurate traffic sign identification, as well as limited memory storage and a slower GPU in an embedded device. However, this experiment showed that there are still areas for improvement in both accuracy and embedded system requirements.

5.2. Future work

Thus, more improvement must be made in must be made in training the neural network to fulfil the areas for improvement.

First, more research can be made to YOLOv4-tiny[20]. It is a faster object detection method with fewer parameters (smaller size) and a simpler network, making it ideal for traffic sign identification. Attempts were done with YOLOv4-tiny in this project. However, due to time constraints and a lack of information to resolve the dataset dispute, it was not thoroughly investigated.

Next, XNOR-net model can be considered to be incorporated into the model. It is shown that yolo3-tiny-xnor[21] can enhance detection performance by 2 to 4 times. However, there was no XNOR-net model for YOLOv4 models at the time, therefore the yolo3-tiny-xnor model had to be modified to fit the YOLOv4 model.

Last but not least, research can be made to design software so that the model can be installed and executed on a variety of embedded devices. With it, more exact performance data can be obtained.

Reference

[1] A. E. Retallack and B. Ostendorf, “Relationship between traffic volume and accident frequency at intersections,” International Journal of Environmental Research and Public Health, vol. 17, no. 4, p. 1393, 2020.

[2] D. C. J. D. Harry Brown Jr, “Leading causes of car accidents with statistics,” JD Supra, 19-Dec-2018. [Online]. Available: https://www.jdsupra.com/legalnews/leading-causes-of-car-accidents-with-60370/. [Accessed: 15-Mar-2022].

[3] J. Donatiello, “Causes of major road accidents and the most common types,” Watts Guerra LLP, 12-Jul-2021. [Online]. Available: https://wattsguerra.com/causes-of-major-road-accidents-and-the-most-common-types/. [Accessed: 19-Mar-2022].

[4] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “XNOR-net: ImageNet classification using binary convolutional Neural Networks,” Computer Vision – ECCV 2016, Aug. 2016.

[5] A. Lee, “Comparing Deep Neural Networks and Traditional Vision Algorithms in Mobile Robotics.” [Online]. Available: https://www.cs.swarthmore.edu/~meeden/cs81/f15/papers/Andy.pdf. [Accessed: 19-Mar-2022].

[6] “Papers with code - coco benchmark (real-time object detection),” The latest in Machine Learning. [Online]. Available: https://paperswithcode.com/sota/real-time-object-detection-on-coco. [Accessed: 19-Mar-2022].

[7] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, YOLOv4: Optimal Speed and Accuracy of Object Detection, pp. 1–13, Apr. 2020.

[8] C.-Y. Wang, H.-Y. Mark Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, and I.-H. Yeh, “CSPNet: A new backbone that can enhance learning capability of CNN,” 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Nov. 219AD.

[9] K. He, X. Zhang, S. Ren, and J. Sun, “Spatial pyramid pooling in deep convolutional networks for visual recognition,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 37, no. 9, Apr. 2015.

[10] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, “Path Aggregation Network for instance segmentation,” 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Sep. 2018.

[11] S.-H. Tsang, “Reading: Panet - path aggregation network, 1st place in Coco 2017 Challenge (instance segmentation),” Medium, 20-Jul-2020. [Online]. Available: https://becominghuman.ai/reading-panet-path-aggregation-network-1st-place-in-coco-2017-challenge-instance-segmentation-fe4c985cad1b. [Accessed: 19-Mar-2022].

[12] C. Yuan and S. S. Agaian, A comprehensive review of Binary Neural Network, Feb. 2022.

[13] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, Binarized Neural Networks: Training Neural Networks with Weights and Activations Constrained to +1 or −1, Mar. 2016.

[14] M. Horowitz, “1.1 computing's energy problem (and what we can do about it),” 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), Feb. 2014.

[15] F. S. Uche, Dataset Reusability Technique for Building YOLOModels, Jun. 2020.

[16] developer0hye, “Yolo_Label,” GitHub/Yolo_Label. [Online]. Available: https://github.com/developer0hye/Yolo_Label. [Accessed: 19-Mar-2022].

[17] V. Sichkar, “Traffic signs dataset in Yolo Format,” Kaggle, 03-Apr-2020. [Online]. Available: https://www.kaggle.com/datasets/valentynsichkar/traffic-signs-dataset-in-yolo-format?select=yolov3_ts_train.cfg. [Accessed: 19-Mar-2022].

[18] Joheras, “CLODSA,” GitHub/Joheras/CLODSA, 30-Aug-2021. [Online]. Available: https://github.com/joheras/CLoDSA. [Accessed: 19-Mar-2022].

[19] Google. (n.d.). Welcome To Colaboratory. Google. Retrieved March 20, 2022, from https://research.google.com/colaboratory/

[20] Z. Jiang, L. Zhao, S. Li, and Y. Jia, “Real-time object detection method based on improved YOLOv4-tiny,” Nov. 2022.

[21] AlexeyAB, “Alexeyab/Darknet: Yolov4 / scaled-yolov4 / yolo - neural networks for object detection (windows and linux version of darknet),” GitHub. [Online]. Available: https://github.com/AlexeyAB/darknet. [Accessed: 21-Mar-2022].