

TRAFFIC SIGN CLASSIFICATION

By

ANG QIAO YI
CHEW LI YANG
WONG SUM HUI
TEOH MING XUE

A PROPOSAL

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

MAY 2024

ABSTRACT

Done by: Chew Li Yang

In this project, we developed a real-time traffic sign classification system that uses modern deep learning techniques, including transfer learning and ensemble learning with pre-trained models. Traffic sign recognition (TSR) plays a vital role in autonomous driving and driver assistance systems, which need to accurately identify signs even in challenging conditions like poor lighting, adverse weather, or partial obstructions. Many powerful models exist, but their large size and high computational demand make them impractical for real-time applications, especially in environments with limited resources. To address this, we created an ensemble model that combines the strengths of **EfficientNetB0**, **MobileNetV3Small**, and **Xception** while each being lightweight but highly effective CNN architectures. By leveraging pre-trained networks, we can shorten training time and improve the model's ability to generalize to unseen traffic signs from the Chinese Traffic Sign Database (TSRD) [29]. Our approach successfully reduces the overall model size while still maintaining high accuracy of 94.60 %. This allows the system to run efficiently in real-time, making it suitable for deployment in vehicles where quick and accurate decisions are critical. The results show a balance between high classification accuracy and computational efficiency, ensuring our system meets the demands of modern road safety and autonomous driving technologies.

Table of Content

TRAFFIC SIGN CLASSIFICATION	i
ABSTRACT.....	ii
LIST OF FIGURES	5
LIST OF TABLES	5
1.1 Problem Statement & Motivation.....	8
1.2 Project Scope & Objectives	9
1.2.1 Project Scope.....	9
1.2.2 Project Objectives.....	10
1.3 Impact, Significance & Contributions.....	11
1.4 Background Information.....	13
CHAPTER 2 – LITERATURE REVIEWS.....	15
2.1 Literature Reviews	15
2.1.1 Towards Real-Time Traffic Sign Detection and Classification.....	15
2.1.2 Traffic Sign Detection and Classification in the Wild.....	18
2.1.3 Lightweight deep network for traffic sign classification[11]	23
2.1.4 Traffic Sign Recognition – How far are we from the solutions?	27
2.2 Comparison of the 4 Techniques	31
CHAPTER 3 – PROPOSED METHOD / APPROACH	33
3.1 Design Specifications.....	33
3.2 System Design	34
3.3 Implementation Challenges.....	41

CHAPTER 4 – SYSTEM IMPLEMENTATION	45
4.1 Hardware Setup.....	45
4.2 Software Setup.....	45
CHAPTER 5 – SYSTEM EVALUATION AND DISCUSSION	48
5.1 Testing Setup and Results	48
5.2 Comparison with previous works	57
5.3 Results Analysis (Case Study)	58
5.3.1 Strength and Weaknesses of Pipelines.....	59
5.3.2 Empirical Comparison.....	66
CHAPTER 6 – CONCLUSION AND RECOMMENDATION.....	70
6.1 Conclusion.....	70
6.2 Recommendation (Future research directions).....	71
REFERENCES.....	72
APENDICES	76

LIST OF FIGURES

Figure 2.1.1. 1: The structure of CNN.	15
Figure 2.1.1. 2: Block Diagram for Traffic Signs Classification and Detection Process	17
Figure 2.1.2. 1: The Flowchart of the Traffic Sign Detection and Classification System [1]19	
Figure 2.1.2. 2: Annotation Pipelines	20
Figure 2.1.2. 3: The traffic sign instances that are annotated with a class label, bounding box and pixel mask.....	20
Figure 2.1.2. 4: The architecture of multi-class network.[1].....	21
Figure 2.1.2. 5: Results for object localization methods for traffic sign across small, medium and large signs	22
Figure 2.1.2. 6: Results for both Fast R-CNN and our method show simultaneous detection and classification of traffic sign across small, medium, and large signs.....	23
Figure 2.1.3.1 The architecture of our teacher network	24
Figure 2.1.3. 2 Table of Performance comparison of the original and pruned student models on the GTSRB dataset	25
Figure 2.1.3. 3 Table of Performance comparison of the original and pruned student models on the GTSRB dataset	25
Figure 2.1.3. 4 Table of Performance comparison on BTSC dataset	26
Figure 2.1.3. 5 Block Diagram of Traffic Sign Classification Process	26
Figure 2.1.4. 1: Block diagram for traffic sign classification process	27
Figure 2.1.4. 2: Table Views of Performance and Running Times of different classifiers reported on GTSC (Times for Classifiers only, disregarding features computation and projection).....	29
Figure 2.1.4. 5: Best Classification Results on GTSC.....	30
Figure 3.1.1: General Block Diagram of Proposed System.....	33
Figure 3.1. 2: Block Diagram with break down workflow	34

Figure 3.2.1 Model Architecture Design for Ensemble Learning.	38
Figure 3.3.1 Class Distribution for Original Training Set.	42
Figure 3.3.2 Class Distribution for Original Test Set.	43
Figure 4.2.1 Example of Libraries imported in Google Colab	46
Figure 5.1.1 Confusion matrix of training sets.....	50
Figure 5.1.2 Confusion matrix of validation sets.	52
Figure 5.1.3 Confusion matrix of testing sets.	54
Figure 5.1.4 Class 43 – Intersection Warning Signs	55
Figure 5.1.5 – Zigzag Road Warning Signs	55
Figure 5.1.6 Learning Curve for the model accuracy and loss.	55
Figure 5.3.1.1 Correctly Classified Images.....	59
Figure 5.3.1.2 Class Distribution for training (after oversampling).	60
Figure 5.3.1.3 Class Distribution for testing (after oversampling).....	60
Figure 5.3.1.4 Class 0 (Speed limits of 5 km/h)	61
Figure 5.3.1.5 Class 3 (Speed limits of 40 km/h)	61
Figure 5.3.1.6 Class 4 (Speed limits of 50 km/h)	61
Figure 5.3.1.7 Class 8 (No left turn and going straight)	61
Figure 5.3.1.8 Class 9 (No going straight or right turn signs).....	62
Figure 5.3.1.9 Class 54 (No Parking).....	62
Figure 5.3.1.10 Class 55 (No Entry).....	62
Figure 5.3.1.11 Misclassified Images.	63
Figure 5.3.1.12 Class 15 (No U-Turn Signs)	63
Figure 5.3.1.12 Class 35 (Pedestrian Crossing Signs)	64
Figure 5.3.1.14 Class 43 (T-junction or side road signs).	64

LIST OF TABLES

Table 3.2.1 Number of samples before and after oversampling.	36
Table 3.2.2 Summary of hyperparameter tuning.	39
Table 4.1 Specifications of laptop	45
Table 4.2.1 Specifications of the software used in the development.....	45
Table 5.1.1 Summary of training results.	48
Table 5.1.2 Summary of validation results	51
Table 5.1.4 Summary of training, validation and testing results.....	57
Table 5.2.1 Comparison of Our Ensemble Model with Previous Work.	57
Table 5.3.2.1 Strengths of the Model.....	66
Table 5.3.2.2 Weaknesses of the model.	67

CHAPTER 1 - INTRODUCTION

1.1 Problem Statement & Motivation

Done by: Teoh Ming Xue

Accurate and timely classification of traffic signs is crucial for the development of autonomous driving systems and advanced driver assistance systems (ADAS). Traffic signs provide essential information that autonomous vehicles rely on to make safe and informed decisions on the road. However, the real-world conditions in which traffic signs appear often complicate their recognition. Traffic signs can be affected by different lighting conditions, adverse weather, and occlusions caused by vehicles, pedestrians, or environmental obstacles like foliage. Additionally, the small size of traffic signs and the complexity of their visual information make them difficult to detect and classify accurately. These challenges reduce the performance of traditional traffic sign recognition (TSR) methods, especially under harsh conditions, and increase the risk of misinterpretation, leading to unsafe vehicle behaviors.

This project is motivated by the need to develop a reliable, efficient, and real-time traffic sign classification system capable of handling these challenges. Autonomous driving technologies and ADAS have the potential to significantly reduce traffic accidents, improve road safety, and offer greater mobility solutions. However, the success of these systems depends on the ability to accurately recognize and interpret traffic signs in real-time. Misclassifications or delays in recognition can lead to incorrect vehicle actions, posing serious safety risks. Therefore, ensuring that TSR systems are not only accurate but also computationally efficient and capable of making decisions in real-time is critical.

To address these issues, this project focuses on implementing state-of-the-art deep learning techniques, including transfer learning and ensemble learning using pre-trained convolutional neural networks (CNNs). These methods help improve the system's accuracy and robustness while reducing the computational overhead, allowing for faster and more reliable classification in real-time. Transfer learning allows the model to leverage knowledge from large-scale datasets, improving its generalization to traffic signs from the Chinese Traffic Sign Database (TSRD) [30], while ensemble learning enhances classification performance by combining multiple CNN architectures. The use of multi-scale CNNs improves the detection of small traffic signs in complex environments, making the model more resilient to varying environmental conditions.

A key focus of this project is ensuring model efficiency, which is essential for real-time applications in embedded systems typically used in vehicles. Traditional deep learning models that achieve high accuracy often come with significant computational demands, making them unsuitable for real-time deployment. By optimizing the model's architecture and leveraging techniques like model size reduction and ensemble methods, this project seeks to create a balance between maintaining high classification accuracy and reducing the computational cost. The system will be trained and validated using the TSRD dataset to ensure that it performs well across different real-world scenarios typical of Chinese roadways.

In summary, this project aims to develop a traffic sign classifier that not only achieves high accuracy but also meets the computational and real-time performance requirements of modern autonomous driving systems. By focusing on both model efficiency and real-time TSR, the project contributes to enhancing the safety and reliability of autonomous driving technologies.

References:

1.2 Project Scope & Objectives

1.2.1 Project Scope

Done by: Wong Sum Hui

This project focuses on developing a real-time traffic sign classification system for autonomous vehicles and advanced driver assistance systems (ADAS). The system aims to classify traffic signs accurately in real-world environments, which present challenges such as varying lighting conditions, adverse weather, and occlusions from objects like vehicles or pedestrians. The dataset used for this project is the Chinese Traffic Sign Database (TSRD), which contains a variety of traffic sign images encountered under these real-world conditions. The dataset is split into training, validation, and test sets using an 80:20 ratio via stratified sampling to ensure balanced representation across all classes.

To handle class imbalances in the dataset, the project applies a moderate oversampling technique. This method, using RandomOverSampler, ensures that minority traffic sign classes are represented adequately during training without significantly inflating the dataset size, which is capped at 10,000 samples. Unlike traditional data augmentation methods, which involve generating new variations of existing images through transformations, moderate oversampling

focuses on balancing the dataset by duplicating underrepresented class samples. This technique helps the model learn to classify rare traffic signs more effectively while maintaining computational efficiency.

The model architecture is based on an ensemble of pre-trained convolutional neural networks (CNNs), specifically EfficientNetB0, MobileNetV3Small, and Xception. These models, fine-tuned to classify traffic signs, leverage their strengths in image classification tasks. Global Max Pooling layers are employed to reduce model size and complexity, while dense layers with batch normalization and dropout ensure effective regularization, reducing the risk of overfitting. The goal of this architecture is to maintain a balance between high accuracy and efficient, real-time performance suitable for deployment in embedded systems.

Training the model involves hyperparameter tuning through a grid search, exploring combinations of learning rates, batch sizes, and optimizers such as Adam, RMSprop, and SGD. This ensures that the model is optimized for both accuracy and computational efficiency. Additionally, techniques like early stopping and learning rate reduction are used to enhance the training process by preventing overfitting and ensuring stable convergence.

Upon completion, the model's performance will be evaluated using metrics such as accuracy, precision, recall, F1-score, and computational efficiency. These metrics will be calculated for the test set to ensure the model's ability to generalize beyond the training data. Confusion matrices and learning curves will be used to visualize performance and identify areas for further improvement. The project's ultimate goal is to develop a lightweight, efficient, and highly accurate traffic sign classification system that is suitable for real-time deployment in autonomous driving technologies and enhances road safety.

1.2.2 Project Objectives

Done by: Wong Sum Hui

This project focuses on developing a real-time traffic sign classification system designed for use in autonomous vehicles and advanced driver assistance systems (ADAS). The classification of traffic signs is a critical task, complicated by real-world conditions such as varying lighting, adverse weather, and occlusions. To address these challenges, the project leverages state-of-the-art deep learning techniques and pre-trained models such as EfficientNetB0, MobileNetV3Small,

and Xception. These architectures will be combined through ensemble learning, enhancing the model's robustness and accuracy while maintaining computational efficiency necessary for real-time performance. The specific objectives of the project are:

1. **Development of an Efficient, Real-Time Traffic Sign Classification Model:**

Implement a robust convolutional neural network (CNN) that utilizes pre-trained models (EfficientNetB0, MobileNetV3Small, and Xception) in an ensemble to accurately classify traffic signs under various real-world conditions. In term of model settings, the ensemble model had frozen its base layers, utilization of LeakyReLU and additional dense layers. Also, moderate oversampling on training data after combination of training and testing datasets. This approach will help the model generalize better across challenging environments.

2. **Optimization of Model Efficiency and Size:**

Ensure the model achieves high classification accuracy while minimizing computational overhead. The use of multi-scale CNNs combined in an ensemble will reduce model size and resource usage, enabling real-time classification without sacrificing performance. Our model able to reduce up to 80% of model size and parameter size in comparison of another similar approach.

3. **Comprehensive Evaluation of Model Performance:**

Assess the model using metrics such as accuracy, precision, recall, F1-score, computational efficiency and utilization of grid search. The model will be validated using the Chinese Traffic Sign Database (TSRD), ensuring that it performs well across diverse conditions typical of real-world driving scenarios. The constructed ensemble model is able to reach accuracy of 94.60% that reflects its robustness and capability in classify traffic signs.

1.3 Impact, Significance & Contributions

Done by: Ang Qiao Yi

This project aims to significantly improve traffic sign recognition (TSR) for autonomous vehicles and advanced driver assistance systems (ADAS) by developing a real-time, efficient traffic sign classification model. Accurate and timely identification of traffic signs is crucial for

safe driving, enabling vehicles to make quick decisions such as adjusting speed, changing lanes, and responding to road conditions. A primary focus of this project is to balance high classification accuracy with model efficiency, ensuring that the system can operate in real-time while being suitable for embedded systems, where both memory and processing power are limited.

To achieve this balance, the model has been carefully optimized to reduce its size while still maintaining high performance. This ensures that it can be deployed in real-time traffic sign recognition systems, which require quick and accurate processing of visual data, even in resource-constrained environments like vehicles' embedded systems. By leveraging transfer learning and ensemble learning techniques, the project demonstrates how modern deep learning models can deliver high classification accuracy without excessive computational demands, making them practical for real-time use.

The project also addresses the key challenges faced in real-world traffic sign recognition, such as lighting variations, adverse weather, and occlusions. These factors often degrade the performance of traditional methods, but by employing advanced convolutional neural networks (CNNs)—such as EfficientNetB0, MobileNetV3Small, and Xception—this project provides a robust solution capable of handling diverse real-world conditions. The goal is not only to create a model that performs well under these conditions but also to ensure that it runs efficiently enough for real-time applications.

The contributions of this project are twofold. First, it develops a lightweight, yet highly accurate CNN-based model for traffic sign classification, optimized for real-time deployment in embedded systems. Techniques such as multi-scale processing and model reduction have been employed to ensure that the model is both fast and resource-efficient, making it ideal for use in autonomous driving systems. Secondly, the project provides a comprehensive evaluation of both traditional classification methods and modern deep learning approaches, demonstrating how ensemble models and transfer learning significantly improve performance, both in terms of accuracy and efficiency.

By focusing on model efficiency and real-time performance, this project makes significant contributions to the development of traffic sign recognition systems that are more reliable, faster, and better suited for autonomous vehicles and ADAS. The findings from this research lay the groundwork for creating scalable, efficient models that can be deployed in real-time scenarios, ensuring that autonomous systems can navigate complex driving environments with greater safety and precision.

1.4 Background Information

Done by: Chew Li Yang

For traffic sign classification within autonomous driving systems and advanced driver assistance systems (ADAS), the tasks of detecting and classifying signs are crucial for decision-making processes. The ability to recognize traffic signs accurately under diverse conditions—such as changes in lighting, weather, and occlusions—determines the safety and efficiency of these systems. This project focuses on the classification phase, which involves determining the type of sign from identified regions in an image. Accurate classification is essential as it provides the necessary information for real-time decisions, such as speed adjustments, lane changes, and responses to road conditions.

The advent of convolutional neural networks (CNNs) has greatly improved traffic sign classification, allowing models to automatically extract hierarchical features from images, which in turn increases accuracy over traditional methods. By leveraging CNNs, the variability in traffic sign appearance due to shape, color, size, and environmental factors can be managed more effectively, making CNN-based approaches highly suitable for traffic sign recognition. Advanced techniques such as multi-scale CNNs and ensemble learning have demonstrated robust performance, achieving near-perfect classification rates on standard benchmarks. These models work by analyzing images at multiple scales, identifying both fine and broad features, which is particularly useful for detecting small and hard-to-recognize traffic signs.

Nevertheless, real-world conditions present significant challenges for traffic sign classification. Traffic signs often occupy only a small portion of the image, making them harder to detect and classify. Factors such as varying lighting conditions, weather disturbances, and occlusions from objects like vehicles and pedestrians further complicate the task. Inaccuracies in

classification, whether due to blurry or obscured signs, can lead to unsafe decisions in autonomous driving systems. To overcome these challenges, the focus is on creating models that not only achieve high accuracy but are also efficient enough for real-time applications.

This project aims to address these challenges by implementing robust CNN-based models that can operate efficiently in real-time. Key techniques used include transfer learning and ensemble learning, which leverage pre-trained models like EfficientNetB0, MobileNetV3Small, and Xception to enhance performance without the need for extensive training. These models are known for their ability to balance accuracy with efficiency, making them well-suited for embedded systems in vehicles where memory and computational resources are limited.

Furthermore, the project introduces a moderate oversampling approach to handle class imbalances in the dataset, ensuring the model can generalize well to underrepresented traffic signs. The overall objective is to create a real-time traffic sign recognition system that is not only accurate but also computationally efficient, thus making it feasible for deployment in real-world autonomous driving scenarios. This work contributes to the growing field of traffic sign recognition by developing models that are both scalable and adaptable, addressing key issues in autonomous driving safety and road efficiency.

CHAPTER 2 – LITERATURE REVIEWS

2.1 Literature Reviews

2.1.1 Towards Real-Time Traffic Sign Detection and Classification

Done By: Ang Qiao Yi

Traffic sign classification is about classifying the detected signs into various related classes in this journal. Convolutional Neural Networks (CNNs) have performed exceptionally well in this area, often doing better than humans. In this case, the journal trained three CNNs for the three-super class, respectively.

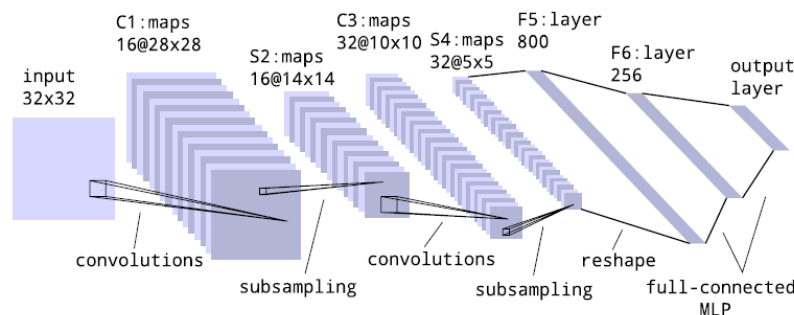


Figure 2.1.1. 1: The structure of CNN.

Various implementation techniques were much different from detection settings, the journal had to supplies back the color and resize images to 32x32 due to the properties of CNN. Another problem for the captured image as the input for the CNN, one specific technique needs to be done to decrease this kind of influence because of the various weather and lighting condition of the images. The journal provides solution to it by using contrast limited adaptive histogram equalization (CLAHE) [4], to perform adjustment on the contrast of the images. As previously mentioned, three CNNs trained were sharing the same simple structure as shown in Figure 2.1.1.1. expect last full connected layer with different number of nodes which is equivalence to the number of sub-classes in each super class to improve computational efficiency. Rather than that, the journal highlighted each CNN contains two convolutional and two subsampling layers, including a full-connected MLP in the final two layer. Furthermore, L2 pooling and more specific settings like image resizing in each subsequent layer were made.

Furthermore, their efforts were then experimentally evaluated using both German and Chinese road datasets for traffic sign detection and classification [6]. For the detection process, theirs was an optimized approach to using color probability models for traffic signs, presenting color changes in complex backgrounds, thereby obtaining superior results in recall rates over SVM and hue-thresholding, with a small number of proposals. They optimized the MSER region detector parameters for effective extraction of the traffic sign proposals with high recall, as well as low processing time, and the novel method of color HOG features with an SVM classifier [5] outperformed existing methods in achieving precision-recall metrics on these datasets, which implies considerable computational efficiency superiority over the existing methods. They trained a CNN with augmented data to be able to classify traffic signs into specified sub-classes; therefore, robust performance across traffic sign datasets can be guaranteed. In general, the proposed integrated approach leverages advanced image processing techniques and machine learning algorithms to achieve competitive accuracy and efficiency in application-related traffic sign recognition.

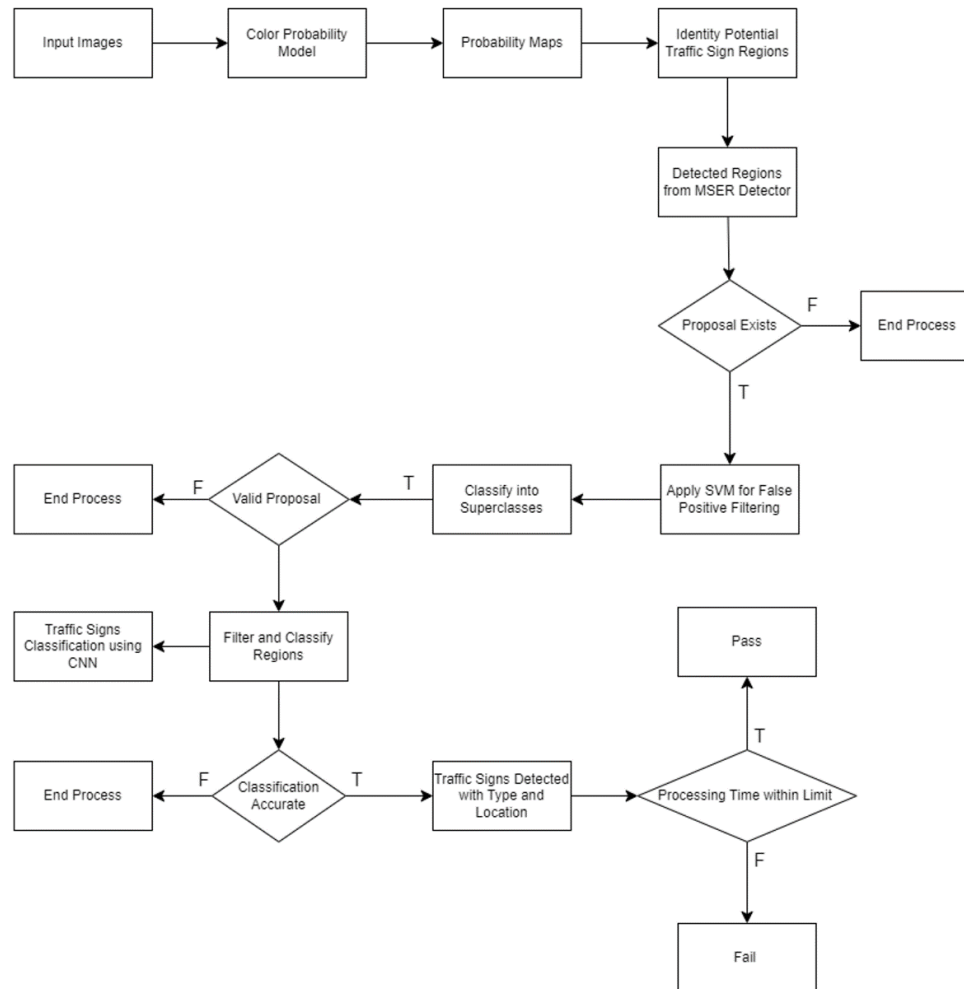


Figure 2.1.1. 2: Block Diagram for Traffic Signs Classification and Detection Process

Based on Figure 2.1.1.2, the block diagram shows a traffic sign detection and classification system. The pipeline starts with input images to which the color probability model is applied to get the probability maps of likely traffic sign regions. These regions are identified at their location with identity potential, then thresholded for isolation and further processing. Stable regions are identified as probable candidates for traffic signs using the MSER detector. A decision point checks for proposed regions, ending the process if none exist. Afterwards, proposed regions are filtered and classified for validity, which includes an SVM classifier for filtering false positives and classifying into super classes of traffic signs. Another decision point checks the validity of the region; in case of an invalid proposal, the process ends. The CNN class, Convolutional Neural Network, further refines the valid proposals to classify the exact kind of traffic sign. In the case of

correct classification, the detected traffic signs specify their type and location. Finally, a decision point checks whether the processing time is within acceptable limits; if yes, then the process ends in success, otherwise failing it. Each step ensures that the traffic signs are identified accurately and quickly from various scenes.

As the conclusion for the literature review, real-time detection and classification of traffic signs have been tremendously improved with advancements in high-level methods. For example, Histograms of Oriented Gradients (HOGs), Support Vector Machines (SVM), and Convolutional Neural Networks (CNNs). These improvements have served, in large measure, to boost accuracy and efficiency to new levels, where challenges emanating from variations in light and complex backgrounds were inhibiting before. These improvements have been validated using benchmark datasets, like GTSDb and GTSRB, that reflect increased detection rates and reductions in false positives. Use of HOG-SVM for initial detection and use of CNNs for detailed classification emphasizes moving toward systems that are more robust and efficient in computational terms. Research on these methodologies will continue, striving to make them better so as to realize intelligent transportation systems that are safer and have more reliability while adapting to an increasingly realistic view of the world with improved precision and speed.

2.1.2 Traffic Sign Detection and Classification in the Wild

Done by: Wong Sum Hui

Up to the present day, convolutional neural networks (CNNs) have shown their strengths for tasks including task-sign detection and classification in the wild. The detection step suggests bounding boxes containing traffic signs within specific categories based on various functions. This is followed by the second classification step, which distinguishes the specific details of each traffic sign. Therefore, there is more than one way to categorize traffic signs depending on the type of functions. Even within these categories they can be grouped by their general shapes and how they look like with some small differences.

Even current benchmarks show perfect or near-perfect performance, either in the detection of traffic signs, such as the 100% recall and precision, or its classification, such as 99.67% precision. However, this performance is based on datasets that are dramatically very different from real-world cases. Some of the real challenging aspects are that usually traffic signs occupy small

portions of an image, often even less than 1%, and numerous cases of negative filtering should take place. Therefore, the authors of the chosen article [1] tried to form a new more genuine benchmark and used it to perform the synergistic CNN-based approaches in traffic sign detection and classification.

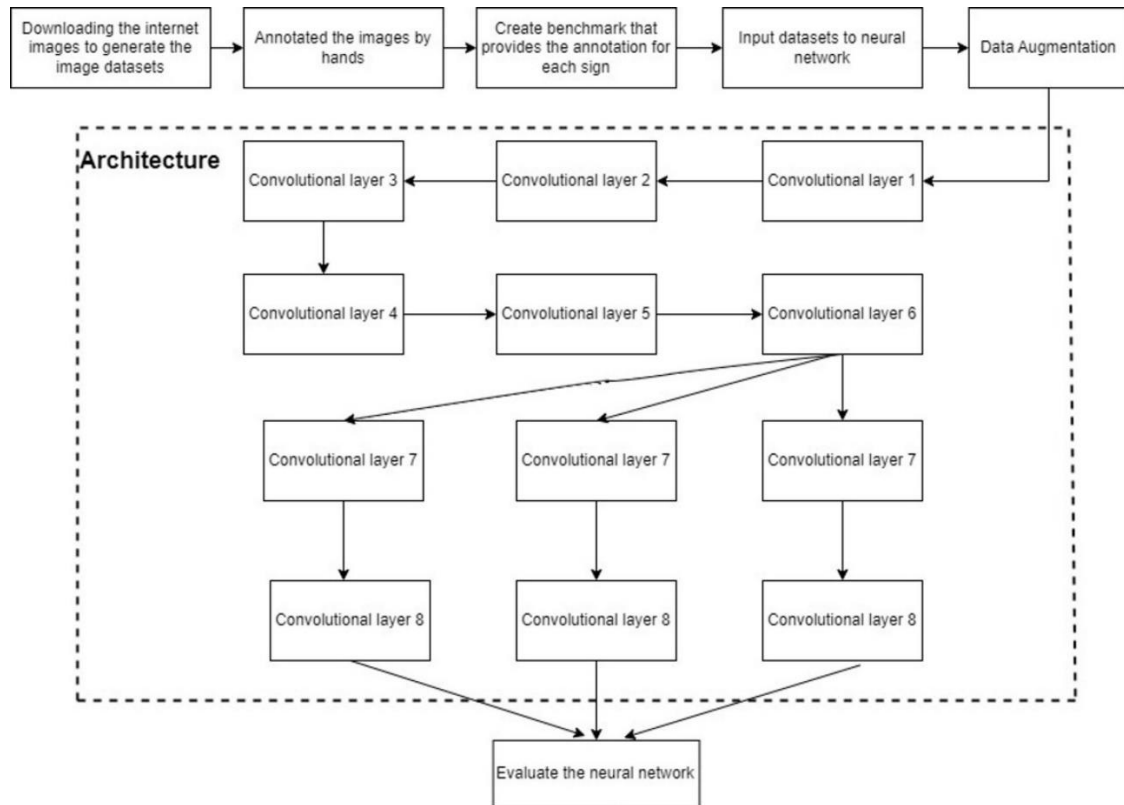


Figure 2.1.2. 1: The Flowchart of the Traffic Sign Detection and Classification System [1]



Figure 2.1.2. 2: Annotation Pipelines

The studies started with creating the image databases by cataloguing images from the internet based on relevant search terms. Additionally, the images that excluded traffic signs were also taken into the benchmark to closely reflect the practical application. This is used to test if it can differentiate real traffic-signs from others. Then, they manually annotated that image as shown as **Figure 2.1.2.2**. When annotating traffic signs, they provided the rectangular box around the sign, marking the vertices of the box and labelling the class of each sign.

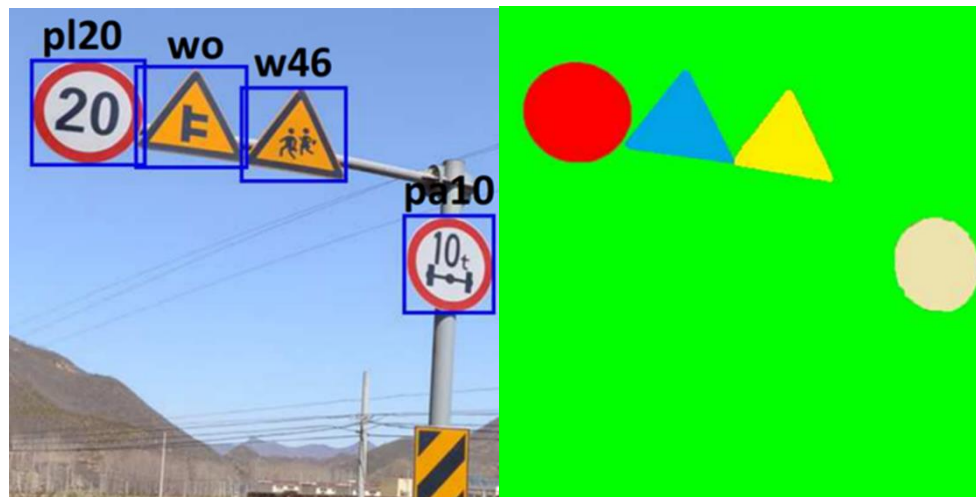


Figure 2.1.2. 3: The traffic sign instances that are annotated with a class label, bounding box and pixel mask

Eventually, the new benchmark Tsinghua-Tencent 100k was created. This new benchmark evidently outperforms the previously widely used benchmark, GTSDb. In this more realistic benchmark, the images are more diverse, and the signs within them are significantly smaller. It includes 111 times more images, each at 32 times the resolution of the images in the previous benchmark. This benchmark comprises 100,000 high-resolution (2048 x 2048) images containing 30,000 traffic sign instances, with each sign annotated with a class label, bounding box, and pixel mask, as illustrated in Figure 3. Moreover, these images encompass a wide range of lighting and weather conditions.

To address the challenges posed by the new benchmark, researchers trained two networks on this dataset. One network focuses solely on detection to categorize all sign classes, while the other handles both detection and classification simultaneously. Some modifications were done on the convolutional neural network (CNN) structure reviewed by previous literature [2]. As shown in the Figure 4, the new network employs full convolutional layers and splits into branches following the 6th layer. This adjustment accelerates the network's convergence compared to the previous structure. Additionally, another improvement is that this network ultimately splits into three streams instead of two. Apart from that, the researchers included the label layer which produces the classification vector of n dimensions, presenting the chance of the input example to be in any of the classes. Therefore, the network can perform the detection and classification tasks at the same time.

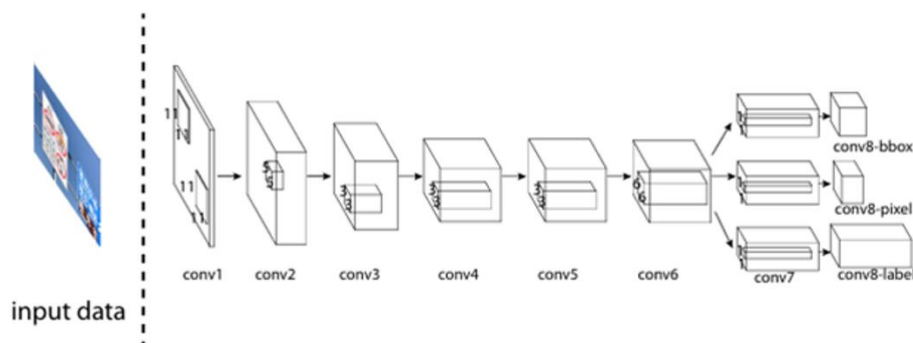


Figure 2.1.2. 4: The architecture of multi-class network.[1]

Data augmentation techniques is implemented while training to handle class imbalance in traffic sign classification. Classes with fewer than 100 instances will be ignored. Classes with more

than 100 examples but fewer than 1000 will then be mapped to one. For classes with more than 1000 instances, no modifications were made. The augmentation process consisted of random rotations within a range of $(-20^\circ/+20^\circ)$, scaling within the range from 20 to 200 pixels, and perspective distortion of standard templates. The above-mentioned augmented templates were then composited into images without traffic signs together with the transformed template with some random noise.

In terms of proposal performance, the traffic sign candidates were detected using Selective Search, Edge Boxes, and BING with each method was all trained through the same augmented training set. However, their average recall, which measure how well they identify relevant objects, was found to be below 0.7 out of a maximum of 1. This indicates that they are not effective for detecting small traffic signs in large, high-resolution images, the result as shown as Figure 5.

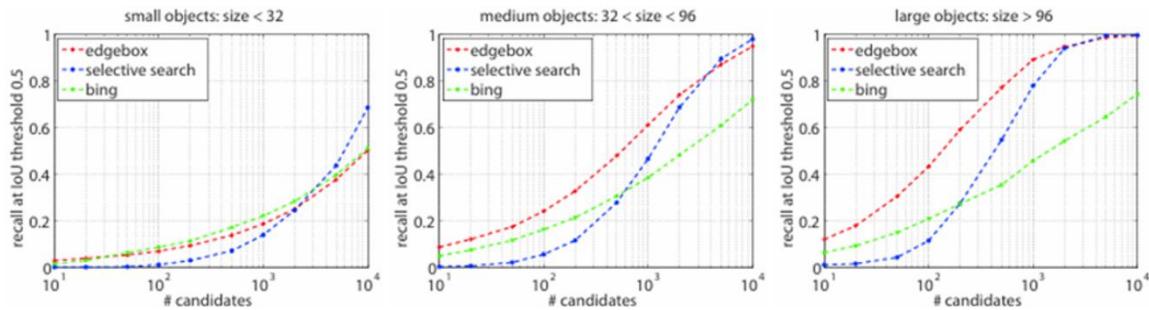


Figure 2.1.2. 5: Results for object localization methods for traffic sign across small, medium and large signs

Based on the experimental results done on a Linux PC with an Intel Xeon E5-1620 CPU, two NVIDIA Tesla K20 G-PU's and 32GB memory, the new approach achieves higher recall (0.91) and accuracy (0.88) compared to the Fast R-CNN method (0.56 and 0.50, respectively), the result as shown as Figure 6. Unlike the previous network, which performed well only for larger objects, this new network is effective for target objects that occupy only a small portion of an image, such as traffic signs.

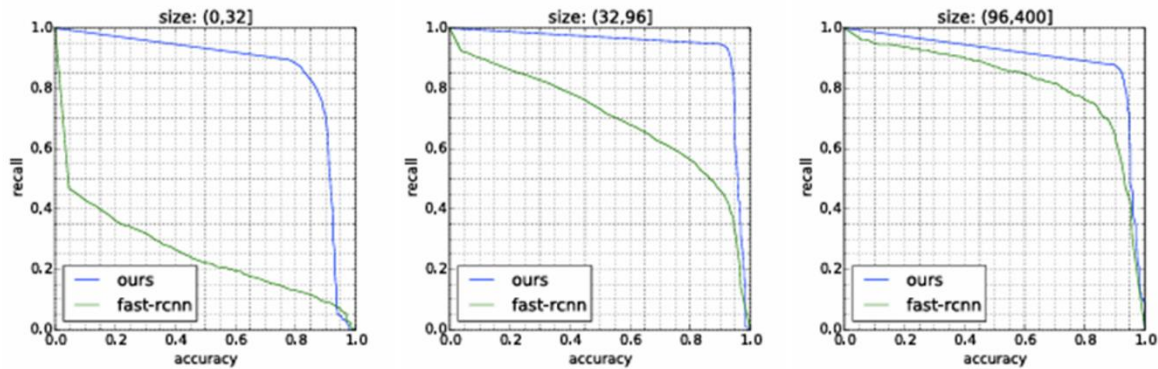


Figure 2.1.2. 6: Results for both Fast R-CNN and our method show simultaneous detection and classification of traffic sign across small, medium, and large signs

In conclusion, this research establishes a robust foundation for advancing traffic sign detection and classification technologies and serves as a reference for future studies. The development of Tsinghua-Tencent 100k as a benchmark dataset represents a significant step towards more realistic and challenging evaluation scenarios. However, there are some limitations to this study. The unavailability of source code for Faster R-CNN and another detector limits the scope of their comparative analysis. Additionally, the limited number of certain traffic sign classes may affect the accuracy or performance of the new benchmark. Therefore, more improvements should be made to this benchmark. The researchers should identify the less commonly seen traffic sign classes in this benchmark and improve processing speed to enable real-time deployment on mobile devices.

2.1.3 Lightweight deep network for traffic sign classification[11]

Done by: Chew Li Yang

Traditionally, machine learning techniques such as support vector machines (SVM) and extreme learning machines (ELM) have been applied using handcrafted features such as colour, shape, and texture. However, these techniques struggle with real world challenge like background complexities, partial occlusion and variation in illumination which leads to loss of critical information and reduced accuracy in classification tasks.

The approach used in the studies proposed is that of knowledge distillation [33] where a larger, deeper Convolutional Neural Network (CNN) model, the teacher network is used to train a smaller and comparatively less complex CNN model, the student network. The student network architecture is relatively simpler and has five convolutional layers and one fully connected layer, while the student network applies knowledge distillation in order to learn from the teacher model's softened output. The output provides detailed prediction probabilities, allowing the student model to generalize better even with fewer parameters. This knowledge transfer approach allows the student network to perform almost as well as the teacher network using only a fraction of the computational resources.

The teacher network integrates low-level and high-level features through dense connections [33], where low-level convolutional features focus on texture information, while high-level convolutional features capture semantic details. It uses dense connections to transfer feature maps between layers and reduces the number of parameters by using 1×1 convolutional filters and 3×3 convolutional kernels running in parallel to extract texture and semantic information more efficiently. In the student network, batch normalization [34] (BN) and ReLU[35] activation functions are used after each convolutional layer to improve nonlinearity, pooling layers help reduce overfitting and reduce feature map dimensions, average pooling preserves background information, and max pooling focuses on texture features. This knowledge transfer method enables the student network to match or behave similar to the teacher network using a smaller fraction of computing resources.

Fig. 2 The architecture of our teacher network

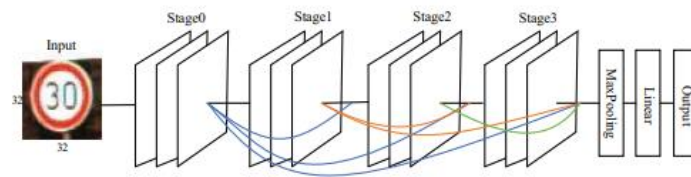


Figure 2.1.3.1 The architecture of our teacher network

In addition to this, this study applies network pruning to enhance the efficiency of the student network further. By targeting the batch normalization, that controls the dimension of the network, pruning can remove channels that are not important. As shown in Figure 2.1.3.2, the studies conducted experiments for which pruning the student network had a accuracy loss -

maintaining 99.38% accuracy on the GTSRB dataset while reducing the number of parameters to 227,851. When 70% of the channels were pruned, the performance of the student model continues to be at 99.08% accuracy with only 85,593 parameters, demonstrating the robustness of the model in resource-constrained environments. As shown in Figure 2.1.3.3 and Figure 2.3.3.4, the lightweight CNN model performs well on both GTSRB and Belgian Traffic Sign Classification (BTSC) datasets. The recognition rate of the student model is 99.61% on GTSRB and 99.13% on BTSC, and the number of trainable parameters is significantly reduced. The results show that there is a slight reduction of accuracy when the additional pruning layer is employed, which is acceptable due to the constraints of the platform that is embedded and mobile platforms in the context of autonomous driving systems.

Table 7 Performance comparison of the original and pruned student models on the GTSRB dataset

Model	Trainable parameters	Accuracy
Our student model	732,139	99.61%
Student model (50% pruned)	227,851	99.38%
Student model (70% pruned)	85,593	99.08%

Figure 2.1.3. 2 Table of Performance comparison of the original and pruned student models on the GTSRB dataset

Table 7 Performance comparison of the original and pruned student models on the GTSRB dataset

Model	Trainable parameters	Accuracy
Our student model	732,139	99.61%
Student model (50% pruned)	227,851	99.38%
Student model (70% pruned)	85,593	99.08%

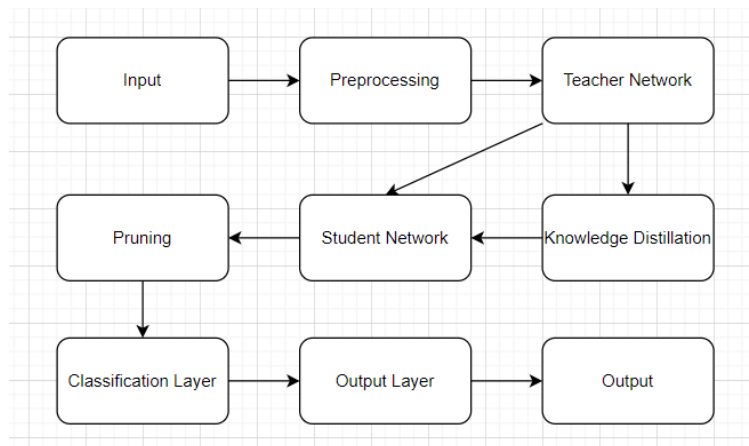
Figure 2.1.3. 3 Table of Performance comparison of the original and pruned student models on the GTSRB dataset

Table 8 Performance comparison on BTSC dataset

Method [Referenced]	Accuracy
Our student network	99.13
GDBM [42]	98.92
Our teacher network	98.89
Our student network (50% pruned)	98.89
Single CNN with 3 STNs [41]	98.87
OneCNN [43]	98.17 \pm 0.22
INNLP + SRC(Pf) [37]	97.83

Figure 2.1.3. 4 Table of Performance comparison on BTSC dataset

The presented approach in the study, knowledge distillation and network pruning, proposes a unique and effective method for classifying traffic signs. By leveraging a teacher-student framework, the smaller student network is trained to replicate the performance of a deeper and more complex CNN model, while maintaining a significantly reduced number of parameters and computational resources. The student model is improved further by removing any redundant channels during pruning thus ensuring it remains highly accurate even when embedded on mobile platforms that always have limited resources are in use. The approach of this method was validated on GTSRB and BTSC datasets with results showing that indeed lightweight models can achieve high recognition rates; hence making them suitable for use in real time applications in autonomous driving.

*Figure 2.1.3. 5 Block Diagram of Traffic Sign Classification Process*

2.1.4 Traffic Sign Recognition – How far are we from the solutions?

Done by: Teoh Ming Xue

With the release of the massive size and variety of traffic signs datasets from Germany [12] and Belgium [13], this journal looks into the Traffic Sign Recognition (TSR) which then split into two distinct topics: Traffic Sign Detection (TSD) and Traffic Sign Classification (TSC).

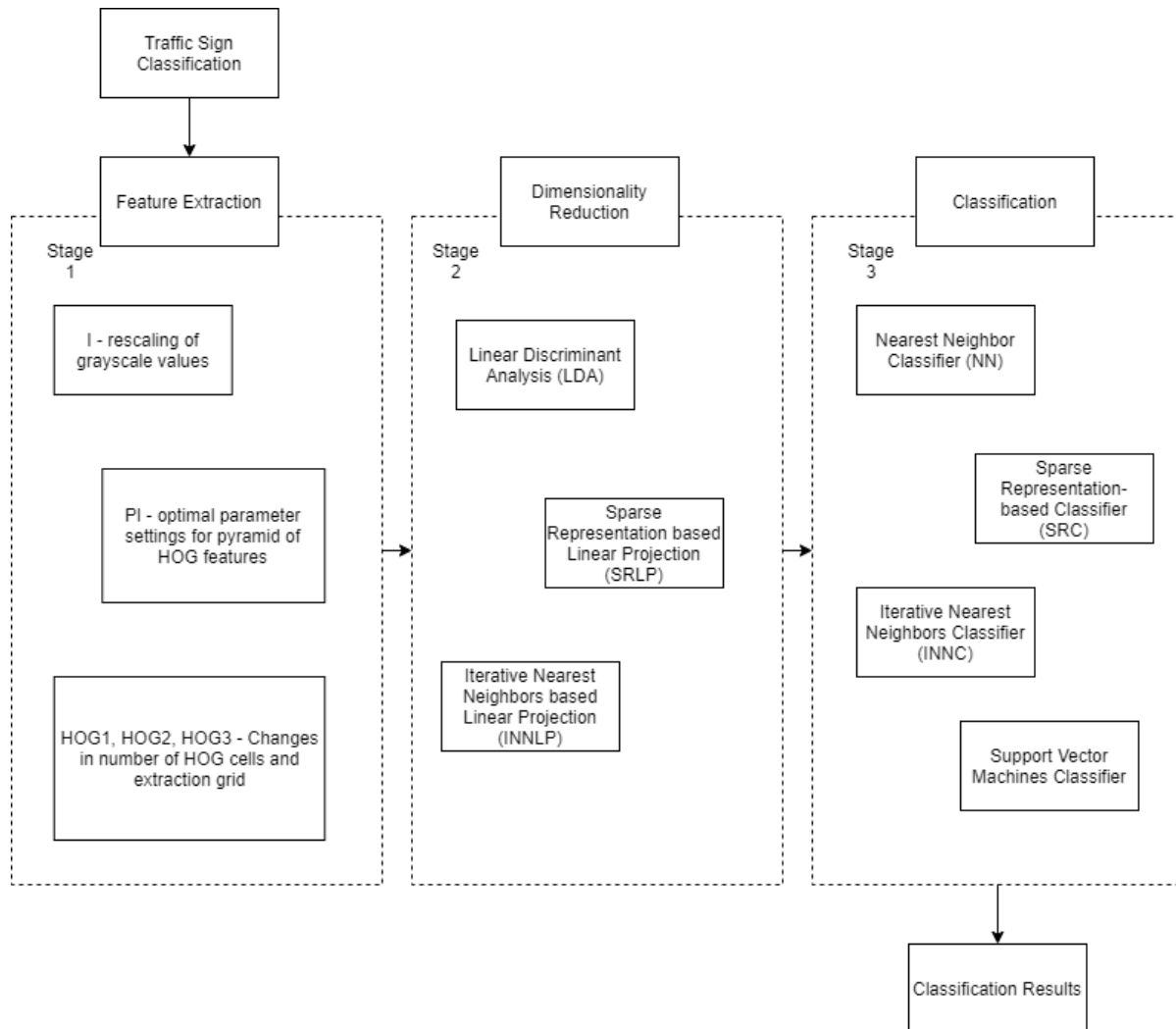


Figure 2.1.4. 1: Block diagram for traffic sign classification process

The journal had mentioned about the stages of the “Ours baseline”, it consists of three stages which includes feature extraction, dimensionality reduction and classification. In features extraction stages, the journal considers the purpose of traffic signs, recognizable features of classifications and their findings in their preliminary experiments testing. It summarized

classification recognize by their inner pattern and shape, thus concluding grayscale-based features will work more efficiently than the colour-dependent features, stating unimportance of colour. Traffic sign classification process can be summarized in Figure 2.1.4.1 as a block diagram.

Other than that, the journal proceeds to the secondary stage, which is the dimensionality reduction. It highlighted that reduction of dimensionality is beneficial for its speed factors. Thus, the journal considered Linear Discriminant Analysis (LDA) [19] [20] by maximizing the inter-class variance while minimizing the intra-class variance. To get the best discriminate results among the classes, problem like eigenvector need to be solved. Besides, techniques like Sparse Representation based Linear Projection (SRLP) [15] and Iterative Nearest Neighbours based Linear Projection (INNLP) [21] provide embeddings reveals numerous structural affinities in the data compare to the direct Euclidean distances. Based on these two inspected techniques, we need to understand the existence of Locality Preserving Projections (LPP). As both techniques are closely related to the LPP variant, SRLP preserves the weights while INNLP replace those from the sparse representation with its own INN coefficients.

Coming at the third stages, which will be the classification stage. The journal had stated multiple methods during the classification stage. Multiple famous classifiers were used including Nearest Neighbours Classifier (NN), Sparse Representation-based Classifier (SRC) [22], Iterative Nearest Neighbours Classifier (INNC) [21] and Support Vector Machines Classifiers. To summarize all the classifier, NN picks the label of the training sample which provides the minimum least square errors to the query. For SRC and INNC, regularization techniques were used to promote sparsity of the coefficients, imposed weight and best combination search. On the other hand, SVM were used in this classification experiments since it was very famous techniques for out of the box classifications, kernels including Linear, Intersection Kernel, Polynomial and the Radial Basis Function to be used in as classifiers.

Classification Experiments:

Feature	Projection	Classifier	Accuracy	Training time	Testing time
I,PI,HOGs	INNLP	INNC ($K = 62$)	98.53%	none	~ 10m
		INNC ($K = 14$)	98.27%	none	~ 3m
	SRLP	SRC	98.50%	none	~ 9h
PI,HOGs	LDA	INNC	98.33%	none	~ 3m
		SRC	98.30%	none	~ 3h
	SRLP	RBFSVM	98.32%	~ 5h	~ 4m
PI	none	IKSVM	97.14%	~ 1day	~ 4m
	SRLP	POLYSVM	96.84%	~ 0.3h	~ 0.5m
	INNLP	LSVM	96.51%	~ 1m	~ 1s
HOG2		RBFSVM	97.08%	~ 0.5h	~ 3m
		IKSVM	96.81%	~ 0.5h	~ 0.5m
HOG2	SRLP	INNC ($K = 14$)	97.57%	none	~ 1m
		INNC ($K = 62$)	97.65%	none	~ 3.5m
		SRC	97.53%	none	~ 1.5h
	LDA	NN	96.97%	none	~ 5s

Figure 2.1.4. 2: Table Views of Performance and Running Times of different classifiers reported on GTSC (Times for Classifiers only, disregarding features computation and projection).

	I	PI	LDA I	LDA PI	SRLP I	SRLP PI	INNLP I	INNLP PI	Avg.
NN	77.15	88.52	92.46	97.47	93.06	97.20	93.61	97.08	92.07
SRC	91.00	95.42	94.67	97.34	95.15	97.55	94.80	97.83	95.60
INNC	86.15	94.79	94.71	97.67	95.15	97.47	94.83	97.55	94.79
LSVM	90.69	96.68	91.48	96.73	91.83	97.32	91.87	97.24	94.23
IKSVM	91.36	97.79	92.22	96.80	91.83	97.08	92.86	97.08	94.63
POLYSVM	89.94	96.61	91.79	96.45	92.85	97.00	93.13	96.96	94.34
RBFSVM	90.41	96.57	91.79	97.08	92.90	97.43	93.37	97.26	94.60
Avg.	88.10	95.20	92.87	96.64	93.25	97.29	93.50	97.29	

Figure 2.1.4. 3: Classification Rate (%) Results on BTSC

	LDA PI	LDA HOG2	SRLP PI	SRLP HOG2	INNLP PI	INNLP HOG2	Avg.
NN	95.83	96.97	95.26	96.56	93.28	96.47	94.05
SRC	96.33	97.20	96.69	97.51	96.70	97.53	95.87
INNC	96.52	97.47	96.54	97.57	95.15	97.65	95.86
LSVM	95.16	96.42	96.03	96.37	96.15	96.51	93.31
IKSVM	95.08	96.22	96.40	96.81	96.43	96.73	93.91
POLYSVM	95.36	96.58	96.84	96.93	96.68	96.84	95.22
RBFSVM	95.34	96.65	96.82	96.95	96.85	97.08	95.30
Avg.	95.66	96.79	96.37	96.96	95.89	96.97	

Figure 2.1.4. 4: Classification Rate (%) Results on GTSC

CR (%)	Team	Method
99.46	IDSIA [26]	Committee of CNNs
98.84	INI-RTCV [4]	Human Performance
98.53	ours	INNC+INNLP(I,PI,HOGs)
98.50	ours	SRC+SRLP(I,PI,HOGs)
98.31	sermanet [27]	Multi-scale CNNs
98.19	[28]	SRGE+HOG2
96.14	CAOR [4]	Random Forests
95.68	INI-RTCV [4]	LDA on HOG2

Figure 2.1.4. 5: Best Classification Results on GTSC

Since many combinations of different classifiers that comes with various feature and feature representation, we conclude some of those results can great differences in term of their training and testing time in Figure 2.1.4.5. It summarized the comparison of timings between different training and evaluation pipelines. For the best accuracy obtained on GTSC datasets, the journal used settings or features likes I, PI, different HOGs, INNLP projection with INNC Classifier set with number of NN iterations, $K = 62$ during training. The journal also stating the best performance over classification results must be obtained by merging multiple features to improve the classification rates. From both Figure 2.1.4.6 and 2.1.4.7, both GTSC and BTSC classification rate can see a significant improvement when merging multiple features mentioned previously. As for the best classification results on GTSC, we can observe from Figure 2.1.4.8, “Ours” from the journals that used INNC and INNLP projections gets 98.53% from the results. It is obvious that “Ours” results did not gain a competitive advantage in term of classification rate when comparing to the committee of Convolutional Neural Network (CNN) [23] which is 99.46%. However, it should be considered that our aims are faster and quicker training and testing time that outperformed them in this category as their CR only comes with time of 37 hours of training hours.

2.2 Comparison of the 4 Techniques

Done by: All Members

Table 2.2.1 Comparison of the 4 Techniques Used in 4 Papers

Article	Techniques	Advantages	Disadvantages
Article 2.1.1	<ul style="list-style-type: none"> • CLAHE (Contrast Limited Adaptive Histogram Equalization) for handling lighting variations. - L2 pooling and resizing techniques for computational efficiency. • CNN-based architecture for classification with 2 convolutional and 2 subsampling layers. 	<ul style="list-style-type: none"> • Efficient handling of varying lighting and weather conditions. • L2 pooling improves processing speed. - Simple CNN structure with high computational efficiency. 	<ul style="list-style-type: none"> • May struggle with small signs in complex scenes. • CNN-based methods can be computationally demanding, limiting their use in embedded systems.
Article 2.1.2	<ul style="list-style-type: none"> • Tsinghua-Tencent 100k dataset for realistic, diverse traffic signs. - Uses Selective Search, Edge Boxes, and BING for candidate generation. • CNN model trained with multi-class output, handling detection and classification simultaneously. 	<ul style="list-style-type: none"> • Achieves higher recall (0.91) compared to Fast R-CNN, especially for small objects like traffic signs. • Simultaneous detection and classification ensure efficient processing. 	<ul style="list-style-type: none"> • Proposal performance below 0.7 recall for small signs using Selective Search and Edge Boxes. • Detection is less effective for very small objects in large, high-resolution images.

Article 2.1.3	<ul style="list-style-type: none"> • Knowledge distillation uses a teacher network to train a smaller, efficient student network. • Network pruning further enhances efficiency by removing redundant channels to reduce model size. 	<ul style="list-style-type: none"> • Suitable for embedded and mobile applications in real-time. • High accuracy and recognition rates with significantly fewer parameters and resources. 	<ul style="list-style-type: none"> • Can be a slight reduction in accuracy with aggressive pruning • Effectiveness of knowledge distillation and pruning depends on careful tuning of hyperparameters like temperature and pruning thresholds.
Article 2.1.4	<ul style="list-style-type: none"> • - Sparse Representation-based Linear Projection (SRLP) and Iterative Nearest Neighbours-based Linear Projection (INNLP) for dimensionality reduction. • Classification using SVM, NN, and SRC (Sparse Representation-based Classifier). 	<ul style="list-style-type: none"> • Faster training and testing times compared to CNN-based models. • Multi-feature merging improves classification rates, achieving 98.53% on GTSC. 	<ul style="list-style-type: none"> • Lower accuracy compared to CNNs (e.g.99.46% by CNN on the same dataset). • Struggles with occlusions and variable lighting in real-world applications.

CHAPTER 3 – PROPOSED METHOD / APPROACH

3.1 Design Specifications

Done by: Teoh Ming Xue

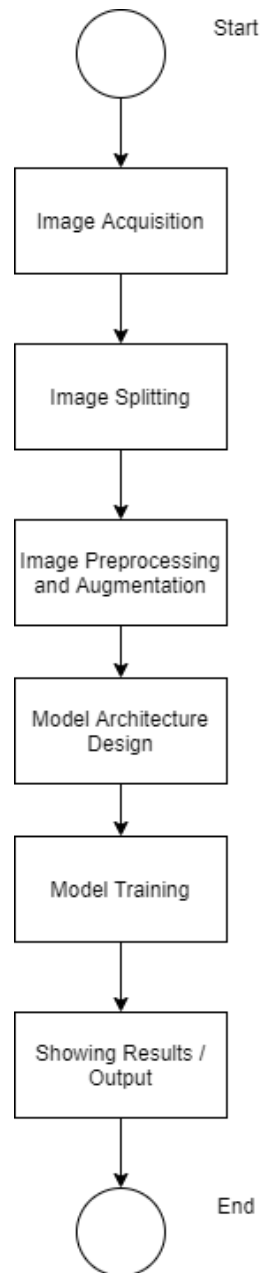


Figure 3.1.1: General Block Diagram of Proposed System

As for the description for our general block diagram, our proposed methods consist of 6 general phases in total, which range from image acquisition, image splitting, image preprocessing

and augmentation (combined steps), model architecture design, model training and showing results or outputs.

3.2 System Design

Done by: Teoh Ming Xue

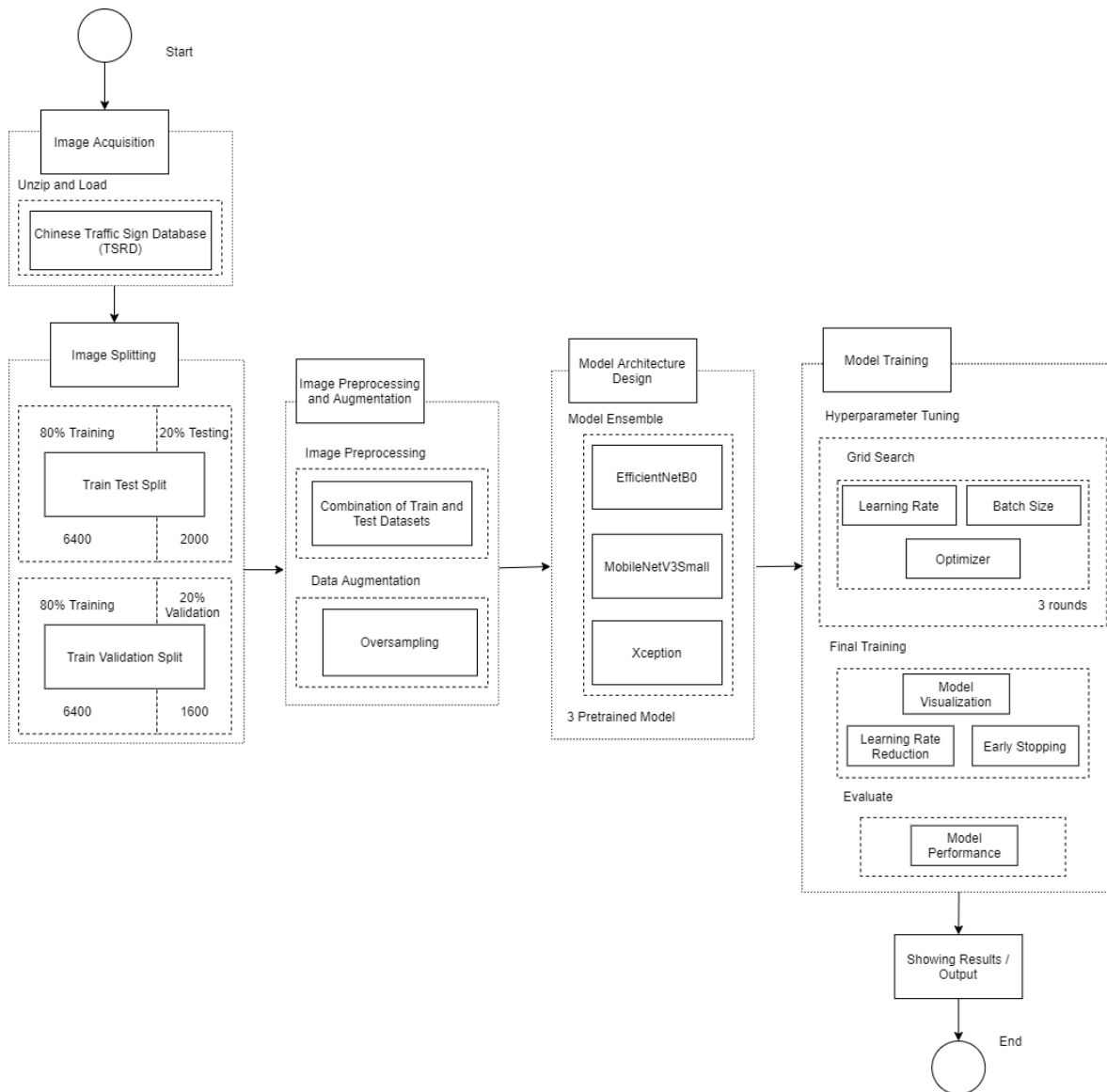


Figure 3.1. 2: Block Diagram with break down workflow

This block diagram provides a detailed breakdown explanation for the traffic sign classification process, highlighting each specification of hyperparameters settings and

justifications from the stages of image acquisition to performance evaluation. The goal is to build an efficient, accurate and lighter weight version of pretrained model in term of ensemble learning to classify traffic sign images.

Image Acquisition

The image acquisition process begins by unzipping and loading the Chinese Traffic Sign Database (TSRD), which contains a total of 6164 images across 58 categories, split into 4170 images for training and 1994 for testing. These images are first extracted from compressed .zip files to enable processing in Google Colab. Once unzipped, the dataset is accompanied by annotation files, which contain mappings between the filenames and their corresponding class labels. The images are resized to a uniform dimension of 256x256 pixels to ensure they meet the input requirements for the model and are normalized by scaling pixel values to a range of [0, 1]. Additionally, the class labels are extracted and stored for classification purposes. To ensure the data is prepared correctly, the number of samples for both training and testing sets is displayed, confirming successful dataset preparation and verification. This step sets the foundation for further preprocessing, model training, and evaluation in the pipeline.

Image Splitting

In this phase, the dataset is split into training, validation, and test sets while ensuring balanced class distributions. The combined dataset, consisting of 10,000 images across 58 traffic sign classes, is initially split into 80% training (6,400 images) and 20% test sets (2,000 images) using StratifiedShuffleSplit. This method maintains the proportion of each class in the training, validation, and test sets, ensuring that all splits reflect the overall class distribution. The training set is further divided into 80% training (6,400 images) and 20% validation (1,600 images) sets, again maintaining class balance. Stratified splitting is crucial in the context of traffic sign classification, where class imbalance can be a challenge. By ensuring a balanced distribution across the splits, the model is better equipped to generalize and avoid bias toward more frequent classes.

Image Pre-processing and Augmentations

After splitting, image preprocessing is applied, which includes resizing all images to a uniform size of 256x256 pixels and normalizing pixel values to a range between 0 and 1 again. These steps standardize the inputs, enabling the convolutional neural network (CNN) to process the data efficiently. Additionally, moderate oversampling is introduced as a data augmentation technique to handle class imbalance. Using RandomOverSampler, underrepresented traffic sign classes are oversampled, increasing their presence in the training set. This technique ensures that minority classes are adequately represented without excessively increasing the overall dataset size. By balancing the class distribution through oversampling, the model is better equipped to recognize rare traffic signs, which improves its robustness and accuracy in real-world scenarios. The overviews of the before and after of oversampling can be illustrated in Table 3.2.1.

Table 3.2.1 Number of samples before and after oversampling.

Dataset	Number of Samples (Training)		Number of Samples (Testing)	
	Before Oversampling	After Oversampling	Before Oversampling	After Oversampling
Chinese Traffic Sign Database (TSRD)	4170	6400	1997	2000

Model Architecture Design

The model architecture is designed to balance performance and efficiency by leveraging an ensemble of pre-trained models which are EfficientNetB0, MobileNetV3Small, and Xception, each chosen for their ability to extract rich features from images while minimizing computational overhead. By freezing the layers of these pre-trained models (all trained on ImageNet), we take advantage of transfer learning, allowing the model to leverage pre-existing knowledge while reducing the training time and computational cost. Instead of using traditional flattening methods, we employ **Global Max Pooling** to significantly reduce the dimensionality of the feature maps,

maintaining the most salient information from each model without inflating the number of parameters. After pooling, the outputs are concatenated and passed through dense layers. L2 regularization is applied to the dense layers to avoid overfitting by penalizing large weights. Additionally, the dense layers use LeakyReLU activation, which stabilizes gradient flow and prevents the vanishing gradient problem. Batch Normalization is applied to each dense layer to accelerate convergence and improve model stability, while Dropout (set at 30%) is included to mitigate overfitting by randomly dropping units during training. The final layer uses a softmax activation function to output probabilities for each of the traffic sign classes. This architectural design, with its focus on reducing the model size and computational requirements, allows the system to perform real-time traffic sign classification while maintaining high accuracy, which is essential for deployment in autonomous driving systems.

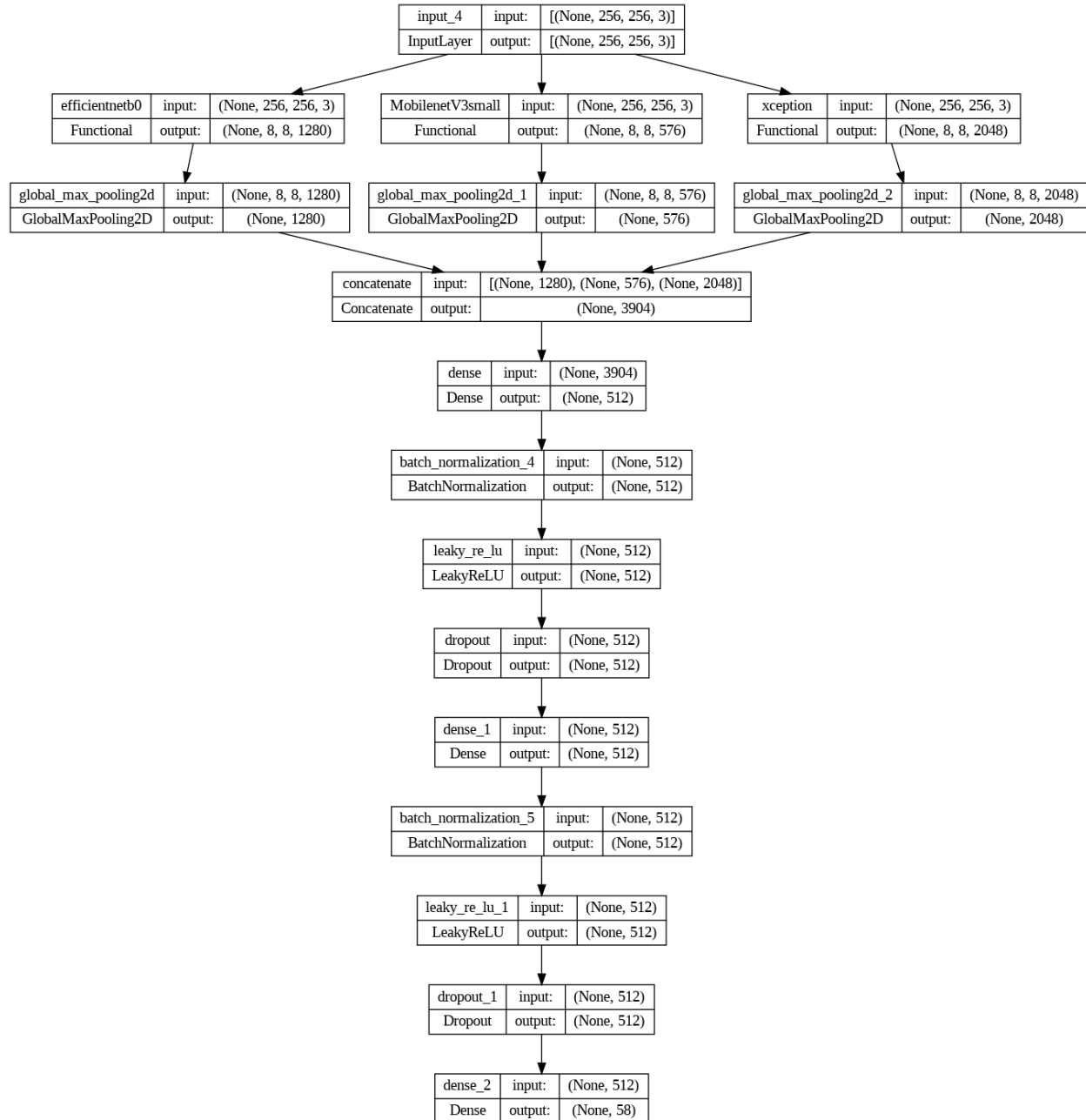


Figure 3.2.1 Model Architecture Design for Ensemble Learning.

Model Training

The model training phase for the traffic sign classification system involved several crucial steps to ensure both accuracy and efficiency, particularly for real-time applications in embedded systems. The training began with hyperparameter tuning through a grid search process aimed at

identifying the optimal combination of learning rate, batch size, and optimizer. These hyperparameters play a pivotal role in the model's learning and computational performance. The grid search tested **learning rates of 0.001 and 0.0001**, with the latter emerging as the preferred choice to ensure gradual and stable learning, avoiding overshooting during optimization. **Batch sizes of 32 and 64** were evaluated, with a batch size of 64 selected for its ability to balance memory usage and computational speed, which is essential in real-time systems. Three optimizers which are **Adam, RMSprop, and SGD** were tested, with RMSprop proving to be the most effective for this model. Its ability to dynamically adjust the learning rate based on the parameter updates made it well-suited for the complexity of convolutional neural networks (CNNs). After evaluating various combinations, the optimal configuration was identified in Table 3.2.2. This combination balanced computational efficiency and learning effectiveness, which is critical for a model that needs to perform in real-time conditions.

Table 3.2.2 Summary of hyperparameter tuning.

Hyperparameters	Tested Values	Optimal Value
Batch Size	32, 64	64
Learning Rate	0.001, 0.0001	0.0001
Optimizer	Adam, RMSprop, SGD	RMSprop

Once the optimal hyperparameters were selected, the final model was trained on the complete training and validation datasets. The model was trained for up to **10 epochs**, with **early stopping** and **learning rate scheduling** in place to ensure that the training process was both efficient and prevented overfitting. Early stopping was configured to halt training if the validation performance did not improve for 10 consecutive epochs, ensuring that the model did not waste time on unnecessary training. Additionally, the learning rate scheduler, ReduceLROnPlateau, was employed to reduce the learning rate by half whenever the validation loss plateaued, further refining the model's learning process. These techniques helped the model achieve strong generalization capabilities, allowing it to perform well not only on the training data but also on unseen validation and test data.

After training, the model was evaluated on the test set, and it maintained the validation accuracy of 94.63%. This performance validated the effectiveness of the chosen hyperparameters and the training approach. The model was then saved for future use and is ready to be deployed in real-time traffic sign classification systems. The use of RMSprop as the optimizer, combined with a moderate batch size and a learning rate that ensured stable and gradual learning, resulted in a model that is both efficient and accurate. This aligns with the overall goal of the project: to develop a lightweight, real-time traffic sign recognition system capable of operating within embedded systems, where computational resources are often limited.

Showing of Results / Output (Performance Evaluation)

The final phase of the project focused on evaluating the performance of the ensemble model across the training, validation, and test sets to assess its generalization ability and classification accuracy in real-world traffic sign recognition tasks. The evaluation process began by analyzing the model's performance on the training set, where it achieved a high accuracy of **96.41%**. This result indicates that the model was able to effectively learn patterns from the training data without overfitting. On the validation set, the model reached an accuracy of **94.63%**, which closely mirrors its performance on the test set, suggesting that the model generalizes well to unseen data. Finally, the test set results, with an accuracy of **94.60%**, confirm that the model performs reliably on new, real-world data, a key aspect of any robust traffic sign classification system.

In addition to accuracy, other metrics such as precision, recall, and F1-score were also computed to provide a more comprehensive understanding of the model's performance. These metrics are particularly useful in assessing how well the model handles each class of traffic signs, especially in terms of balancing false positives and false negatives. The model exhibited consistently high precision and recall across the datasets, which is crucial for ensuring that important traffic signs are correctly identified and classified in real-time applications.

To further illustrate the model's performance, confusion matrices were generated for the training, validation, and test sets. These matrices visually demonstrate how well the model predicted each class, while also highlighting misclassifications. On the training set, the confusion matrix showed near-perfect results, with very few errors, confirming the effectiveness of the model during training. However, on the validation and test sets, the confusion matrices revealed some

misclassifications, particularly among traffic signs with similar shapes or colors. This suggests that while the model is highly accurate, there are certain classes that pose challenges, particularly under real-world conditions where lighting or occlusions may distort the appearance of the signs.

To assess the model's learning behavior, learning curves were plotted, displaying the changes in accuracy and loss over the course of training. The accuracy curves for both the training and validation sets increased steadily over time, indicating effective learning and minimal overfitting. Similarly, the loss curves decreased consistently, further supporting that the model was optimizing well throughout the training process.

Finally, a qualitative analysis was conducted by examining examples of correctly classified and misclassified traffic sign images from the test set. The correctly classified images demonstrated the model's ability to handle a wide range of traffic sign types, from common signs like stop signs and speed limits to more complex signs. However, the misclassified images revealed the model's difficulty in distinguishing between traffic signs with similar features, such as those that share the same shape or color scheme. This visual analysis underscored the strengths of the ensemble model in handling diverse traffic signs but also highlighted areas where further refinement or augmentation could improve performance, particularly in real-world scenarios.

In summary, the model performed well across all datasets, achieving a test accuracy of **94.60%**. The ensemble approach, utilizing EfficientNetB0, MobileNetV3Small, and Xception, provided a balance between accuracy and model efficiency, making it suitable for real-time traffic sign classification applications. Nonetheless, further improvements could be made in areas where visually similar signs are often misclassified, which could involve more targeted data augmentation or model fine-tuning strategies.

3.3 Implementation Challenges

Done by: Wong Sum Hui

In the implementation phase of the traffic sign classification system, several challenges were encountered, notably class imbalance in the dataset, missing classes in the test set, and computational costs associated with using ensemble learning techniques and hyperparameter

tuning. These issues were critical to address to ensure the model's reliability, particularly in real-world traffic sign classification for autonomous systems.

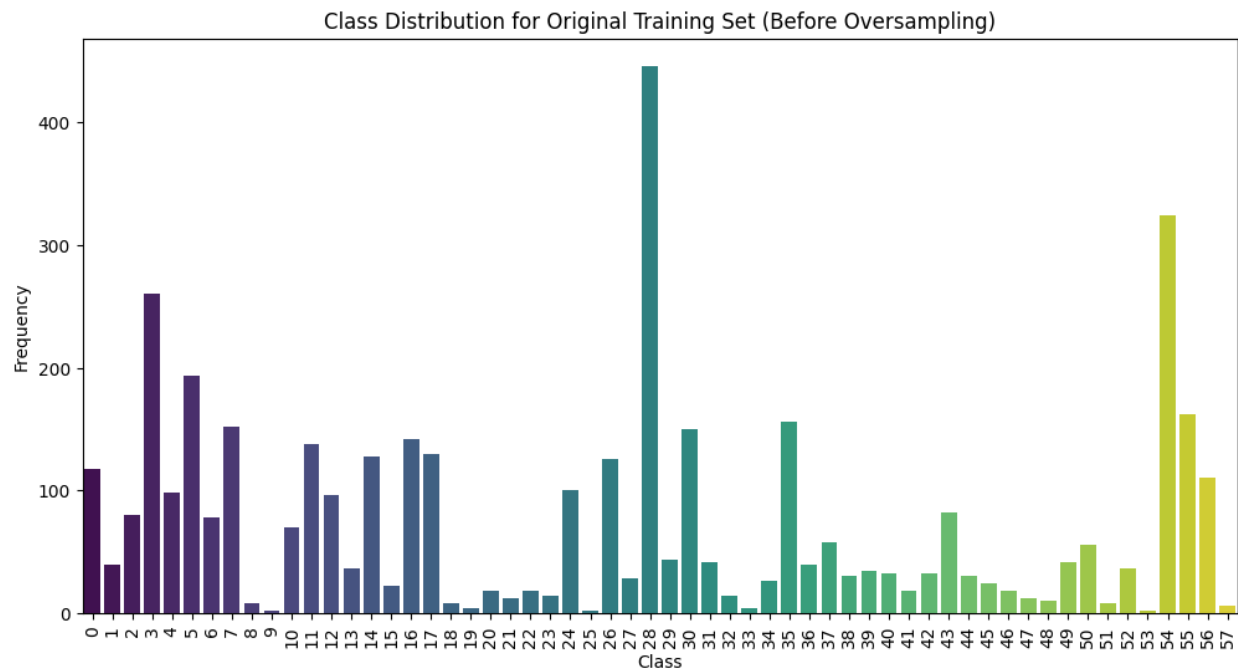


Figure 3.3.1 Class Distribution for Original Training Set.

One of the major challenges was class imbalance in the training set, as highlighted in the “Class Distribution for Original Training Set” figure 3.3.1. The distribution of traffic sign samples was heavily skewed, with certain classes, such as class 28, having over 400 samples, while many other classes had significantly fewer. This imbalance could cause the model to become biased towards the more frequent classes, reducing its ability to correctly classify less common traffic signs. Models trained on such imbalanced data often overfit to the majority classes, leading to poor performance when encountering minority classes. To mitigate this, moderate oversampling was employed to balance the training data. This technique involved oversampling the minority classes to ensure that the model could learn from a more balanced representation of traffic signs, without excessively inflating the dataset size. However, oversampling comes with the risk of overfitting to the minority classes if not handled carefully, which is why the dataset size was capped to prevent excessive oversampling.

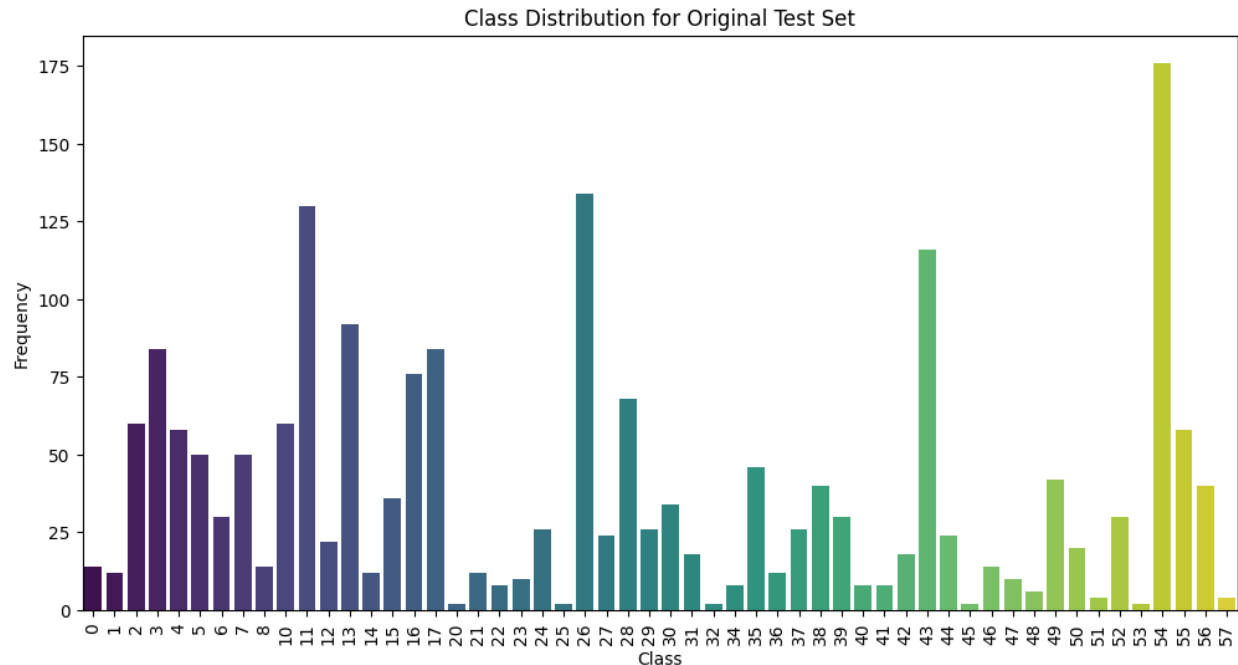


Figure 3.3.2 Class Distribution for Original Test Set.

Another challenge was the issue of missing classes in the test set, as seen in the "Class Distribution for Original Test Set" figure 3.3.2. Some classes present in the training data were missing or severely underrepresented in the test data. This made it difficult to evaluate the model's true performance across all traffic sign categories, as it could not be tested on certain signs. The absence of these classes in the test set can create a false sense of performance, where the model appears to perform well on the available test data but may fail when confronted with unseen or rare traffic signs in real-world scenarios. This further underscored the importance of ensuring that the model was well-regularized and generalizable across different categories, despite the limitations of the testing data.

In terms of computational costs, the use of ensemble learning added complexity. The model combined three different pre-trained CNN architectures (EfficientNetB0, MobileNetV3Small, and Xception), which increased the computational overhead compared to a single model. While ensemble models are often more accurate because they aggregate the strengths of multiple architectures, they come at the cost of higher memory and processing requirements. This issue was particularly relevant during hyperparameter tuning using grid search. Grid search required training and evaluating the ensemble model multiple times across various configurations of learning rates,

batch sizes, and optimizers. While it helped in identifying the optimal hyperparameters (RMSprop with a learning rate of 0.0001 and a batch size of 64), this process was computationally expensive. The complexity of the ensemble model, combined with the exhaustive search through different hyperparameter combinations, significantly increased the training time and resource consumption.

In summary, while the implementation of this traffic sign classification system presented challenges related to class imbalance, missing classes and computational costs. To address the problem, careful handling of these issues through oversampling, regularization, and ensemble learning allowed the model to achieve robust performance. Despite the higher computational requirements of the ensemble model, it was well-suited for the task due to its improved accuracy and generalization capabilities. These adjustments and strategies ensured that the model was optimized for real-time traffic sign recognition in autonomous driving systems.

CHAPTER 4 – SYSTEM IMPLEMENTATION

4.1 Hardware Setup

Done by: Ang Qiao Yi

The hardware involved in this project involve a high-performance laptop which the specifications are listed out at Table 4.1. The hardware played a crucial role in the development and deployment of traffic sign classification project.

Table 4.1 Specifications of laptop

Description	Specifications
Model	Asus ROG Strix G15 G513
Processor	AMD Ryzen 7 5800H
Operating System	Windows 11
Graphic	NVIDIA GeForce RTX3050 4GB GDDR6
Memory	32GB DDR4 3200Mhz RAM
Storage	1TB Samsung SSD

4.2 Software Setup

Done by: Chew Li Yang & Wong Sum Hui

For the development of this traffic sign classification system, a variety of powerful libraries and frameworks were used to ensure efficient model training, data handling, and evaluation.

Table 4.2.1 Specifications of the software used in the development.

Component	Descriptions
Development Environment	Google Colab - Cloud-based Jupyter notebook with GPU support for deep learning tasks.
Libraries and Frameworks	TensorFlow/Keras for building CNN models, NumPy and Pandas for data handling and manipulation.
Image Processing	OpenCV for resizing and normalizing images.

Class Imbalance Handling	Imbalanced-learn's RandomOverSampler to address dataset imbalance.
Model Components	EfficientNetB0 , MobileNetV3Small , and Xception as base architectures for ensemble learning.
Optimization	Adam, RMSprop, and SGD as optimizers. ReduceLROnPlateau and EarlyStopping as callbacks.
Visualization	Matplotlib and Seaborn for plotting data distribution, learning curves, and confusion matrices.
Metrics	Scikit-learn metrics (accuracy, precision, recall, F1-score) for performance evaluation.
Oversampling	Imbalanced-learn's RandomOverSampler for balancing the dataset by oversampling minority classes.

```
# Import necessary libraries
import os
import zipfile
import numpy as np
import pandas as pd
import cv2 as cv
import networkx as nx
from sklearn.model_selection import StratifiedShuffleSplit, ParameterGrid
from imblearn.over_sampling import RandomOverSampler
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Input, Dense, GlobalMaxPooling2D, BatchNormalization, Dropout, LeakyReLU, Concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam, RMSprop, SGD
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping
from tensorflow.keras.applications import EfficientNetB0, MobileNetV3Small, Xception
from tensorflow.keras.regularizers import l2
from tensorflow.keras.utils import plot_model
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
```

Figure 4.2.1 Example of Libraries imported in Google Colab

The NumPy and Pandas libraries were employed for efficient numerical computations and data manipulation, providing a foundation for handling image data and annotations. OpenCV (imported as cv) was used for image preprocessing, such as resizing and normalizing images. To address the class imbalance issue, Imbalanced-learn's RandomOverSampler was incorporated to balance the dataset effectively, particularly for minority traffic sign classes.

TensorFlow and Keras were the primary deep learning libraries utilized for building the convolutional neural network (CNN) models. Pre-trained architectures such as EfficientNetB0, MobileNetV3Small, and Xception were loaded via Keras to implement a powerful ensemble learning approach. Essential layers like GlobalMaxPooling2D, Dense, and Dropout were combined to build the custom architecture. Optimizers such as Adam, RMSprop, and SGD were used for hyperparameter tuning to optimize model performance.

In terms of callbacks, EarlyStopping and ReduceLROnPlateau were used to prevent overfitting and dynamically adjust the learning rate, respectively. The Matplotlib and Seaborn libraries were used for visualization of results, including class distribution, confusion matrices, and learning curves. Finally, Scikit-learn's metrics were utilized to calculate performance metrics like accuracy, precision, recall, and F1 score.

This entire development process was performed using Google Colab, a cloud-based Jupyter notebook environment that provides access to powerful GPUs such as the TPU v2, A100 and L4 GPU, facilitating faster model training and execution of computationally intensive tasks like ensemble learning and grid search.

CHAPTER 5 – SYSTEM EVALUATION AND DISCUSSION

5.1 Testing Setup and Results

Done by: Ang Qiao Yi & Wong Sum Hui & Chew Li Yang

Testing Setup

The testing environment for this model was conducted using **Google Colab** with a TPU v2 GPUs to accelerate the training and evaluation processes. The dataset used for this project was the Chinese Traffic Sign Recognition Dataset (TSRD) [1], which includes 58 traffic sign classes. The dataset was split into 80% training, 20% validation, and 20% testing sets, with moderate oversampling applied to the training set to mitigate class imbalance. This ensured better representation for minority classes, improving the model's ability to generalize across all traffic sign classes. The final test set comprised 2,000 images, representing various real-world traffic signs from different lighting, weather, and environmental conditions.

Results

The evaluation of the ensemble model was conducted across three main phases: training, validation, and testing. The performance metrics, including accuracy, precision, recall, F1-score, confusion matrices and finally learning curve, were analyzed for each phase to assess the model's classification capabilities.

Training Results

Table 5.1.1 Summary of training results.

Metrics	Training Values (%)
Accuracy	96.41
Precision	96.82
Recall	96.41
F1 Score	95.91

During the training phase, the model achieved a **training accuracy of 96.41%**, indicating that 96.41% of the training images were correctly classified. This high accuracy suggests that the

model was able to effectively learn the distinguishing features of the traffic signs in the training dataset, thus accurately recognizing most of the classes.

In terms of **precision**, the model attained **96.82%**, meaning that when the model predicted a certain class, it was correct 96.82% of the time. This high precision demonstrates that the model has a low rate of false positives, which is critical in traffic sign classification to avoid incorrect predictions that could lead to inappropriate responses by an autonomous vehicle.

The model's **recall** was **96.41%**, meaning it successfully identified 96.41% of the relevant traffic signs across all classes. A high recall indicates the model's ability to detect most of the traffic signs, reducing the risk of missed signs (false negatives), which is important for ensuring road safety in real-time applications.

The **F1-score** which represented the harmonic means of precision and recall was **95.91%**, balancing the need for both accuracy in classification (precision) and completeness in detection (recall). The high F1-score reflects that the model is both accurate in its predictions and thorough in capturing the relevant signs.

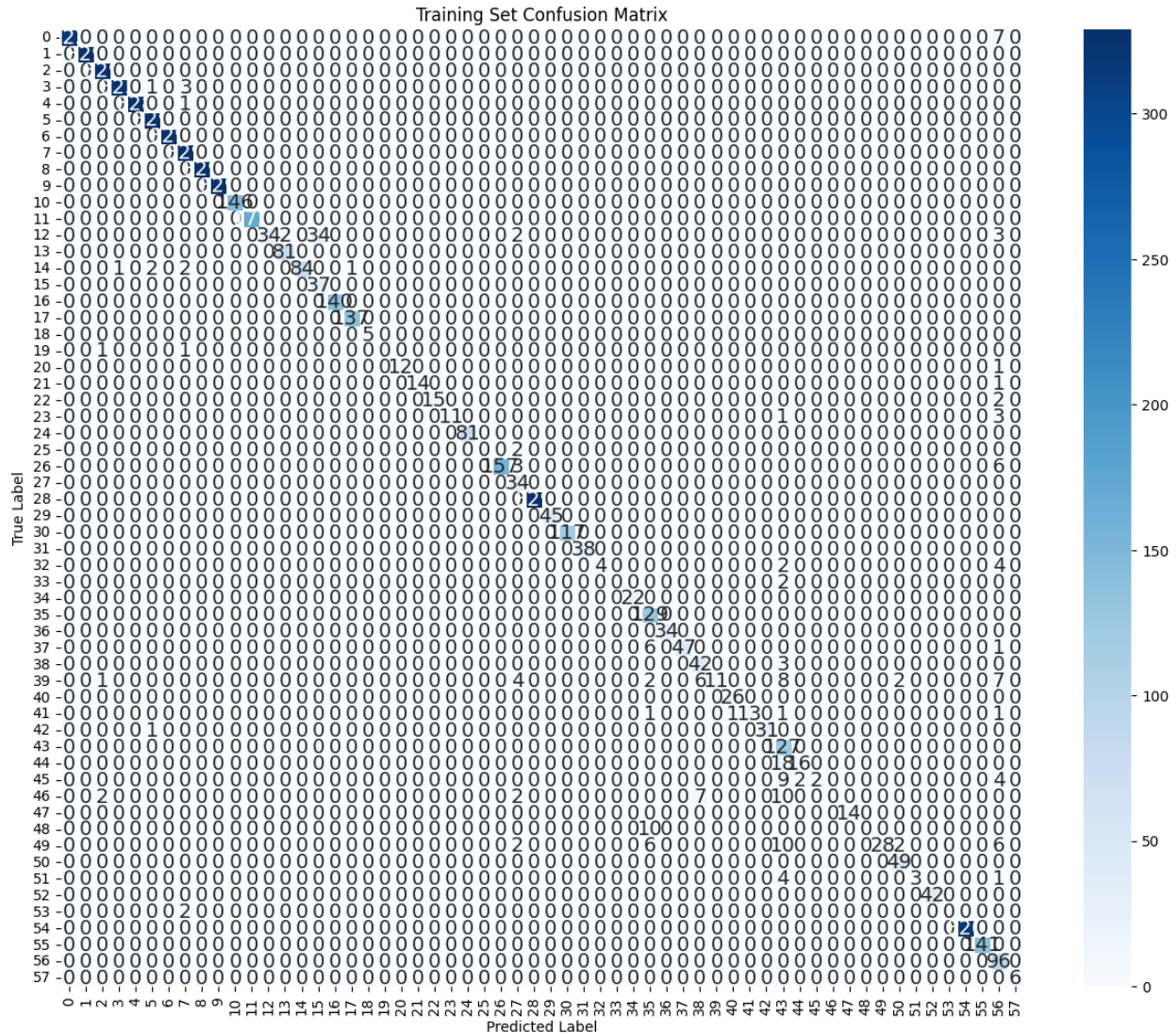


Figure 5.1.1 Confusion matrix of training sets.

The confusion matrix for the training set (Figure 5.1.1) highlights the model's performance on each class. The majority of predictions fall along the diagonal, signifying correct classifications. As for the strong classes such as class 0 to 8 (mostly speed limit traffic signs) and class 28 (Cars only signs) were classified with near-perfect accuracy, reflecting the model's strong ability to recognize these common and distinct traffic signs. In contrast, weak classes like some confusions were noted between class 19 (recommended speed limits of 50 km/h) and class 33 (traffic light signs), likely due to the subtle visual differences between these signs and due to class imbalance (for the mentioned weak classes only had less than 4 images).

Validation Results

Table 5.1.2 Summary of validation results

Metrics	Validating Value (%)
Accuracy	94.63
Precision	94.66
Recall	94.63
F1 Score	93.88

In the validation phase, the model achieved an accuracy of **94.63%**, which reflects its ability to generalize well to unseen data. Although slightly lower than the training accuracy, this is expected as validation data is not part of the model's training process, yet the model still performed well in classifying previously unseen signs.

The model's precision on the validation set was **94.66%**, indicating that when the model predicted a class, it was correct 94.66% of the time. This high precision indicates that the model maintained its ability to avoid false positives even on new data.

The recall for the validation set was **94.63%**, demonstrating that the model continued to identify the majority of relevant traffic signs in the validation set, ensuring that important signs were not missed. The F1-score, which combines precision and recall, was **93.88%**, confirming that the model balanced accuracy and thoroughness in identifying traffic signs. The summary of validation results can be illustrated in Table 5.1.2.

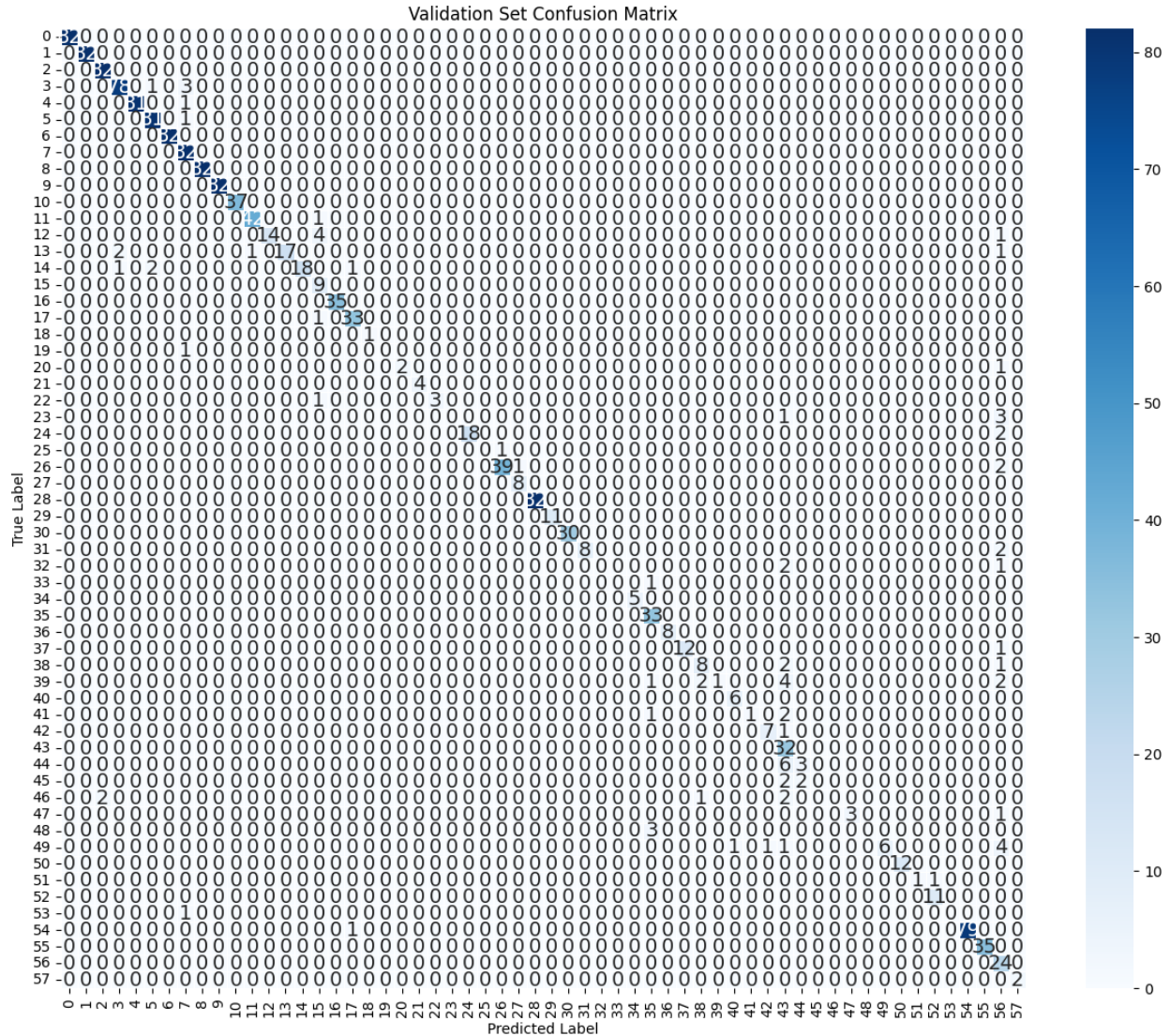


Figure 5.1.2 Confusion matrix of validation sets.

The confusion matrix for the validation set (Figure 5.1.2) highlights how the model performed for individual classes. Similar to the training phase, Class 0 to 9 (mostly speed limits signs) and Class 28 (designated road for cars) continued to show strong performance. However, as observed earlier, confusion persisted between Class 19 (speed limit 50 km/h) and Class 33 (traffic light signs). This indicates that the model's weakness in distinguishing certain visually similar signs persisted even in the validation phase.

Testing Results

Table 5.1.3 Summary of testing results

Metrics	Testing Values (%)
Accuracy	94.60
Precision	95.08
Recall	94.60
F1 Score	93.93

The model achieved a test accuracy of **94.60%**, demonstrating that it was able to generalize well to entirely unseen test data. This is an important indicator of the model's robustness and ability to function in real-world environments, such as recognizing traffic signs in different conditions or locations.

The test set precision was **95.08%**, showing that the model was still highly reliable in making accurate predictions without confusing signs with other classes. The recall for the test set was **94.60%**, indicating that the model detected most of the relevant traffic signs in the test data. This ensures that important traffic signs would not be missed during real-time operations. The F1-score, balancing precision and recall, was **93.93%**, further supporting the conclusion that the model was both accurate and thorough in identifying traffic signs in the test phase.

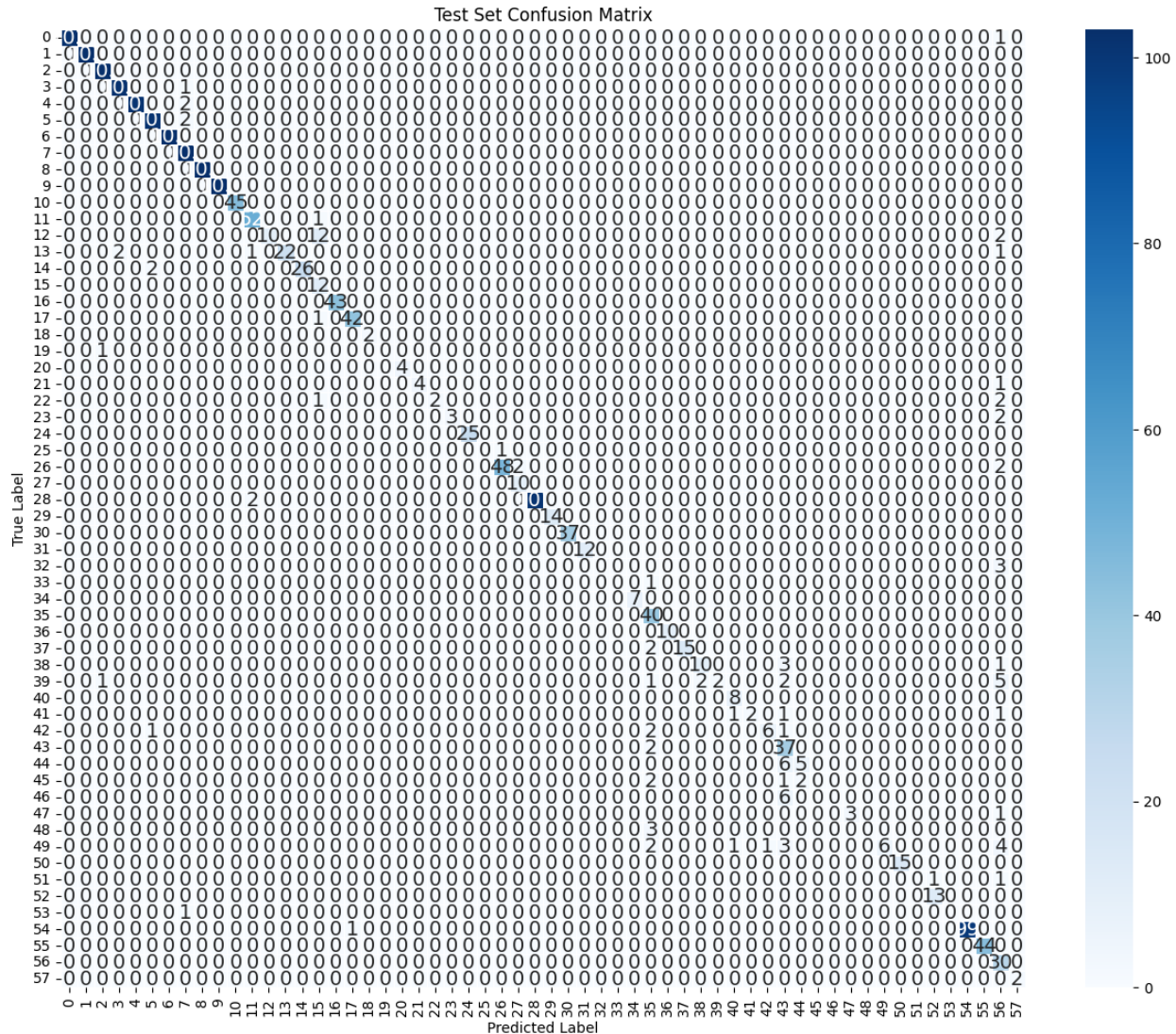


Figure 5.1.3 Confusion matrix of testing sets.

The test confusion matrix (Figure 5.4.1) provides a detailed breakdown of the model's performance across all traffic sign classes. Classes like Class 0 to 9 (Mostly speed limits signs) and Class 28 (designated sign for cars only) continued to perform exceptionally well, demonstrating the model's consistency. However, the confusion between Class 43 (intersection warning) and Class 46 (zigzag road warning) persisted, indicating that these signs are particularly difficult for the model to differentiate because of their similarity in shape and color shown in figure 5.1.4 and 5.1.5. Also, Class 35 (Pedestrian Crossing Sign) and Class 48 (Construction Occurrence sign) both exist with a human symbol which causes the model to have confusion overtimes.



Figure 5.1.4 Class 43 – Intersection Warning Signs



Figure 5.1.5 – Zigzag Road Warning Signs

Learning Curve

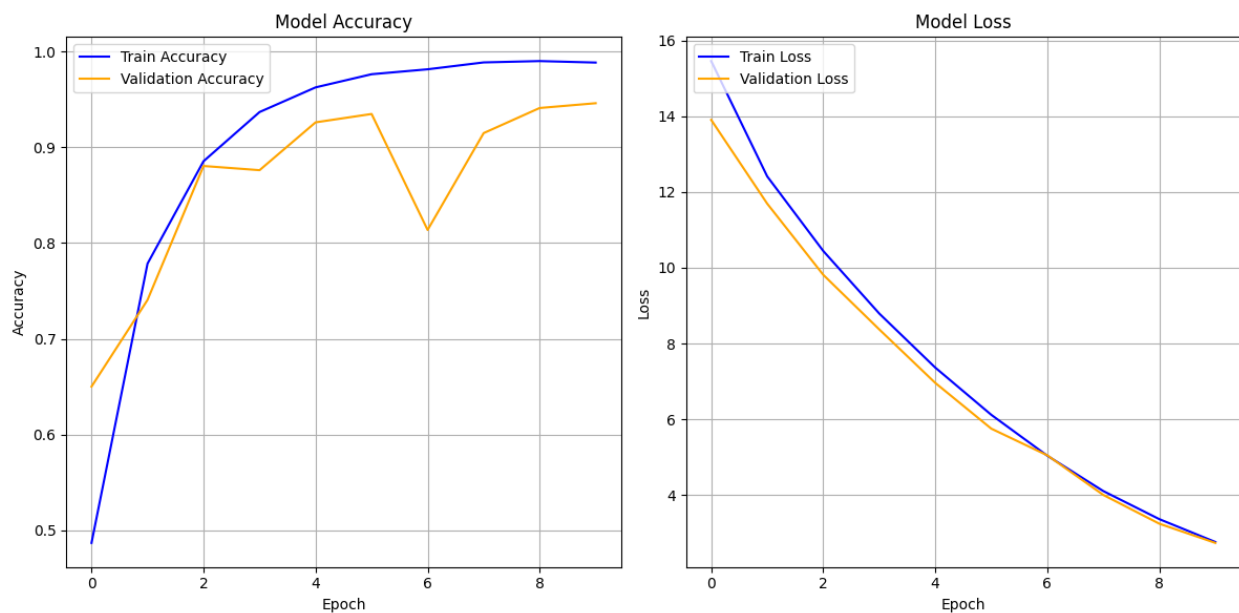


Figure 5.1.6 Learning Curve for the model accuracy and loss.

The learning curves, as shown in the combined Figure 5.1.6 (Model Accuracy and Loss), provide a comprehensive view of how the model performed throughout the training process. The left panel shows the model accuracy, while the right panel illustrates the model's loss across both training and validation sets.

In the accuracy curve (left panel), the training accuracy improves steadily, reaching approximately 96% by the end of training. The model shows rapid learning during the first few epochs, where the training accuracy quickly rises from around 50% to 95% by the third epoch. This indicates that the ensemble model, combining EfficientNetB0, MobileNetV3Small, and Xception, quickly adapted to the features of the traffic signs, demonstrating its efficiency in learning complex patterns from the dataset. The validation accuracy follows a similar trend, which is a good indicator that the model did not overfit to the training data. While there is a small dip in validation accuracy between epochs 5 and 6, this is followed by a recovery, suggesting that the model experienced minor difficulties generalizing at that point but adjusted itself effectively by the end of training. The final validation accuracy is close to the training accuracy, indicating strong generalization and minimal overfitting.

The loss curve (right panel) complements the accuracy curve by showing the decrease in both training and validation loss over time. The rapid reduction in training loss, particularly in the first few epochs, confirms that the model was learning effectively. The smooth decline in validation loss, mirroring the training loss curve, further confirms that the model was able to generalize well on unseen data. By the end of training, the model's loss was low for both the training and validation sets, which indicates that the ensemble model was not just memorizing the training data but was capturing the underlying patterns that allowed it to perform well on new data.

In summary, the learning curves reveal a well-trained model with a strong ability to generalize, thanks to its efficient use of the ensemble learning approach. The minimal gap between training and validation accuracy and loss further highlights the model's robustness, making it suitable for real-time traffic sign classification where both accuracy and generalization are critical.

Summary of Results

The ensemble model, integrating EfficientNetB0, MobileNetV3Small, and Xception, delivered strong performance across all phases of evaluation. The model maintained high accuracy, precision, recall, and F1-scores during training, validation, and testing. Despite some difficulties in distinguishing between visually similar signs, such as speed limit signs and traffic light signs, the model consistently demonstrated reliable classification performance for the majority of traffic sign classes.

Overall, this evaluation confirmed that the ensemble model is suitable for real-time traffic sign classification, which is crucial for applications in autonomous driving and advanced driver assistance systems (ADAS). Its high precision and recall ensure that traffic signs are classified accurately and that important signs are not missed, thereby contributing to safer road navigation.

Table 5.1.4 Summary of training, validation and testing results.

Phase	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Training	96.41	96.82	96.41	95.91
Validation	94.63	94.66	94.63	93.88
Testing	94.60	95.08	94.60	93.93

5.2 Comparison with previous works

Done by: Ang Qiao Yi

In this section, we compare our ensemble model with a previously proposed traffic sign classification model presented by [35]. The goal is to assess the improvements made by our model in terms of model size, total parameters, accuracy, and other key performance metrics, with a particular focus on real-time efficiency and deployment potential for embedded systems. [35] utilized an ensemble learning technique with **ResNet50**, **DenseNet121**, and **VGG16**, applying majority voting to combine the predictions of each model to evaluate the Chinese Traffic Sign Database (TSRD) [12]. This ensemble model showed strong accuracy but came with higher computational costs and memory requirements due to its large parameter size and model footprint.

Table 5.2.1 Comparison of Our Ensemble Model with Previous Work.

Model	Total Params (M)	Model Size (MB)	Accuracy (%)	F1-Score (%)	Precision (%)	Recall (%)
Paper's Model [35]	163.18	622.49	96.41	96.16	96.16	96.16
Our Ensemble Model	28.15	107.37	94.60	93.93	95.08	94.60

Our ensemble model significantly reduces the total parameters from 163.18 million in [35] approach to just 28.15 million, an **80% reduction**. This smaller parameter count also results in a substantially smaller model size which is **107.37 MB** compared to their **622.49 MB** model. The reduction in size is particularly valuable for real-time traffic sign recognition systems designed for embedded platforms, where memory and computational resources are often constrained. This reduction was achieved by utilizing more efficient architectures like EfficientNetB0, MobileNetV3Small, and Xception, which are designed for both computational efficiency and strong feature extraction.

While our model achieves slightly lower accuracy (94.60%) compared to [35] model (96.41%), the trade-off in accuracy is marginal considering the substantial gains in model size and computational efficiency. Our ensemble model also performs well in terms of F1-score, precision, and recall, showing that it can still accurately classify traffic signs across various conditions. The small accuracy gap (around **1.81%**) suggests that our model is still highly effective, particularly when weighed against the practical benefits of smaller size and lower computational demands.

In summary, our ensemble model demonstrates significant improvements in terms of model size and efficiency while maintaining competitive classification performance. These enhancements make our approach more viable for deployment in real-time systems, such as autonomous vehicles, where computational resources are often limited, and efficiency is paramount. The smaller model size and faster computation times enable more practical implementations without sacrificing too much accuracy, ultimately contributing to safer and more effective traffic sign recognition in real-world applications.

5.3 Results Analysis (Case Study)

Done by: Teoh Ming Xue

In this section, we examine the strengths and weaknesses of the traffic sign classification model by analyzing several correctly and incorrectly classified images. The analysis of these images allows us to identify patterns in the model's performance and better understand which traffic sign types are most successfully recognized, as well as where the model struggles, which can inform future improvements.

5.3.1 Strength and Weaknesses of Pipelines

Strengths of the Model



Figure 5.3.1.1 Correctly Classified Images.

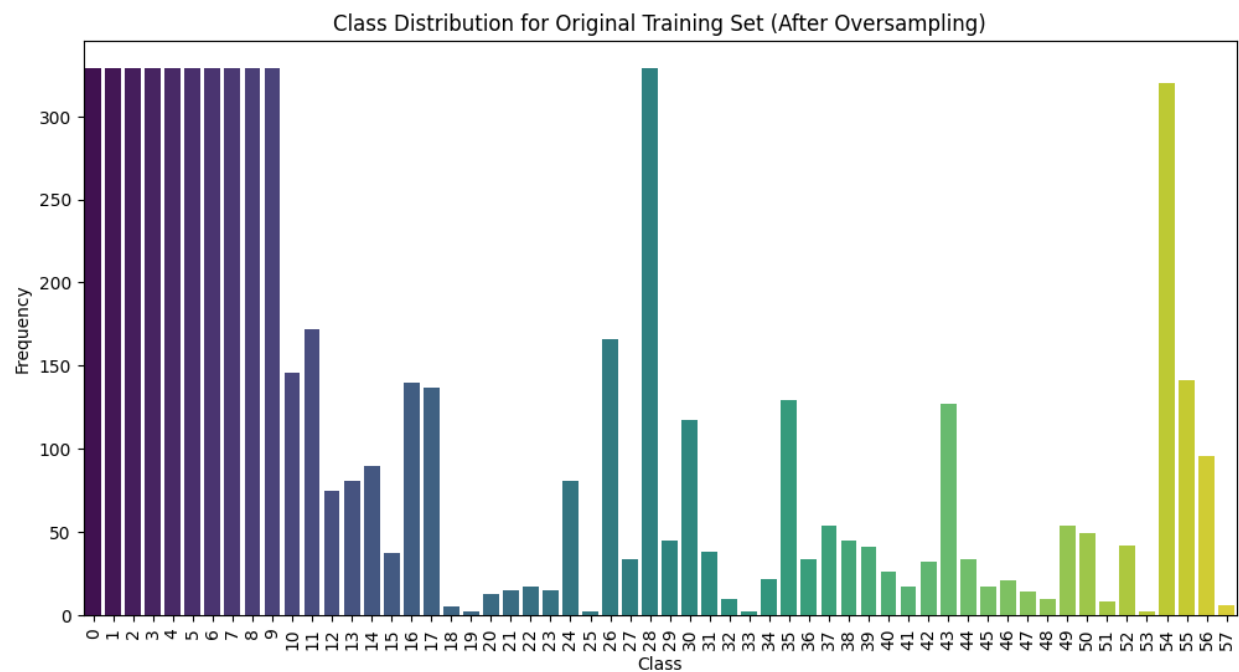


Figure 5.3.1.2 Class Distribution for training (after oversampling).

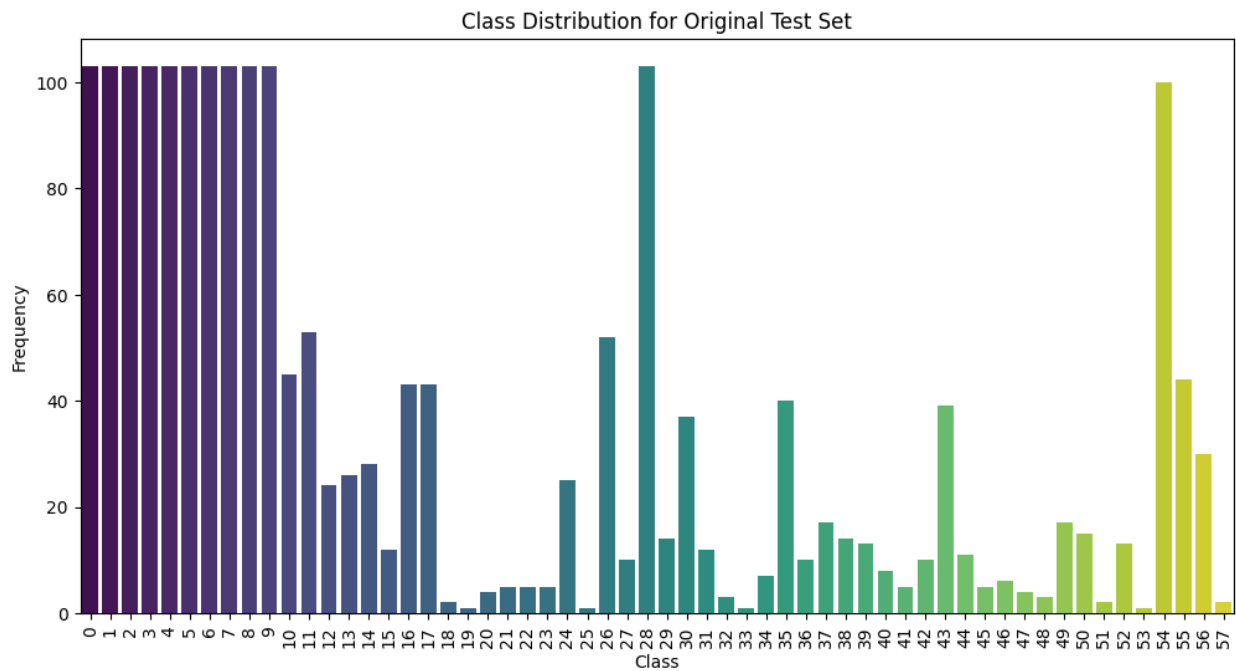


Figure 5.3.1.3 Class Distribution for testing (after oversampling).

The model demonstrated a strong ability to correctly classify a wide variety of traffic signs, as seen in the examples of correctly classified images (Figure 5.3.1.1). This performance is particularly notable for signs that have clear, distinct features and are well represented in the training dataset by achieving success classified of 1892 images out of 2000 images.

- Class 0 to 7 (Mostly speed limits traffic signs):** The model showed a high degree of accuracy for these types of signs, likely due to their prominent, **easily distinguishable design with bold numerals and circular shapes**, which are common features learned effectively during training. The main reason might be contributed to the **oversampling effects** that had been done earlier to make sure all imbalance classes were filled. The increased representation of speed limit signs in the training data, as a result of oversampling, contributed to faster learning and better classification results. The model was able to generalize these signs more effectively, improving its overall accuracy. The figures in Figure 5.3.1.2 to Figure 5.3.1.3 show the results of this oversampling effect, as well as the visualization of correct classifications in Figures 5.3.1.4 to 5.3.1.7 that demonstrate the model's proficiency in recognizing these signs.



Figure 5.3.1.4 Class 0 (Speed limits of 5 km/h)



Figure 5.3.1.5 Class 3 (Speed limits of 40 km/h)



Figure 5.3.1.6 Class 4 (Speed limits of 50 km/h)



Figure 5.3.1.7 Class 8 (No left turn and going straight)

- **Class 9 (No going straight and right turn signs), Class 54 (No Parking) and Class 55 (No Entry):** The model performed exceptionally well even for **blurry and low-resolution images**. As shown in Figures 5.3.1.8 to 5.3.1.10, the details of the images are not fully visible and contain a certain degree of visual noise. Despite these challenges, the model was able to accurately classify the traffic signs. This highlights the model's robustness and ability to generalize well, even in less-than-ideal conditions where visual clarity is compromised, making it suitable for real-world applications where image quality can vary significantly.



Figure 5.3.1.8 Class 9 (No going straight or right turn signs).



Figure 5.3.1.9 Class 54 (No Parking)



Figure 5.3.1.10 Class 55 (No Entry)

The high precision and recall scores achieved during testing (Precision: 95.08%, Recall: 94.60%) further support the model's strong capability in identifying these commonly seen traffic signs in a variety of real-world conditions. These results are crucial for real-time traffic sign recognition, as they indicate the model's reliability in handling frequent and essential signs in autonomous driving scenarios.

Weaknesses of the Model



Figure 5.3.1.11 Misclassified Images.

Despite the overall strong performance, certain classes of traffic signs posed challenges for the model, resulting in misclassifications. The misclassified images in Figure 5.3.1.11 highlight several recurring patterns of difficulty between several classes:

- Class 15 (No U-Turn Signs):** The primary reason for the model's weakness in distinguishing between class 12 (no right and left turns), class 22 (left turn) as class 15 (no U-turn) in Figure 5.3.1.12 is due to the visual similarities in the design of these traffic signs. All three signs share circular shapes with red borders and arrows, making them visually similar, especially in conditions such as low resolution or poor lighting. This **similarity in shape and color can cause confusion for the model**, leading to misclassification, as it relies heavily on these features for sign identification. To enhance performance, the model could benefit from more refined training on distinguishing finer details, such as arrow direction, as well as more robust data that accounts for these subtle differences under varied conditions.



Figure 5.3.1.12 Class 15 (No U-Turn Signs)

- Class 35 (Pedestrian Crossing Signs):** The model struggled to correctly distinguish class 35 (Pedestrian Crossing) from other similar classes like class 48 (Roadworks), class 42 (Slow down signs), and class 37 (Children Crossing). The confusion arises primarily because all of these signs share a common triangular shape with a yellow background and feature black symbols in the center. The figures in these signs, such as pedestrians, construction workers, and children, are relatively small and similar in appearance, making it difficult for the model to differentiate between them effectively. This indicates a **weakness in the model's ability to capture finer details**, especially when dealing with signs that share similar shapes and background colors. The small size of the distinguishing symbols within these signs likely challenges the model's ability to pick up on subtle visual differences, which are crucial for accurately identifying these traffic signs in real-world scenarios. This highlights the need for

better feature extraction or additional training to improve the model's precision in differentiating between similar cautionary signs.



Figure 5.3.1.12 Class 35 (Pedestrian Crossing Signs)

- **Class 43 (T-junction or side road signs):** The model's inability to distinguish between these classes (38, 39, 44, 46, 49, and 43) could be attributed to the subtle visual similarities between the traffic signs, such as shape and color patterns. All of these signs share a triangular shape with black symbols on a yellow background, which can confuse the model, especially when the internal symbols appear visually similar, like the curves or intersections indicated by these classes. The **directionality or curvature in the signs** may only differ slightly, such as a sharp turn versus a winding road, or a left intersection versus a zigzag symbol. These small differences, particularly when combined with potential image noise, slight angles, or obstructions in the dataset, make it difficult for the model to distinguish between them, leading to frequent misclassifications.



Figure 5.3.1.14 Class 43 (T-junction or side road signs).

The model has three main weaknesses: it struggles with **visually similar circular signs**, like no U-turn and directional signs; it **misclassifies triangular warning signs**, such as pedestrian


crossings and roadworks due to similar colors and symbols; and it **fails to distinguish subtle variations** in intersection and curve signs. These challenges arise from the model's difficulty in recognizing fine details in signs with similar shapes and color patterns.

5.3.2 Empirical Comparison

Done by: Ang Qiao Yi

In this section, we examine the empirical performance of the ensemble model across various test cases based on the Chinese Traffic Sign Recognition Dataset (TSRD). The analysis focuses on how well the model handles different classes of traffic signs, identifying its strengths and weaknesses based on real-world test data.

Table 5.3.2.1 Strengths of the Model.

Class	Sign Type	Strengths	Reasons	Contributing Factors
Class 0 – 7 (Speed Limits Signs), Class 28 (Cars Only) and Class 54 (No Parking)		High accuracy on easily distinguishable bold numerical and circular shapes	Simple and high-contrast design makes these signs easily identifiable	Oversampling during training helped balance class representation




Class 9 (No go straight and right), Class 54 (No Parking) and Class 55 (No Entry)		High accuracy even on blurry and low-resolution signs	Model learned key features that remain consistent despite poor image quality	Generalization from training on diverse image qualities helps performance
---	---	---	--	---





Table 5.3.2.2 Weaknesses of the model.

Class	Sign Type	Weaknesses	Reasons	Contributing Factors
Class 12 (No left and right turns), 15 (No U-Turn) and 22 (Turn Left)		Misclassification between similar circular signs	Visual similarities between arrows and circular shapes lead to confusion	Lack of distinctive features in direction arrows and red borders across classes

CHAPTER 5

<p>Class 38 (Sharp Right), 39 (Sharp Left), 44 (Side Road), 46 (Zig-Zag) and 49 (Repeated Zigzag)</p>		<p>Fails to distinguish between subtle variations in intersection symbols</p>	<p>Small differences in curve directionality and intersection type are easily missed</p>	<p>Similar shape and symbol placement, coupled with visual noise, make classification harder</p>
---	--	---	--	--

CHAPTER 5

Class 35 (Pedestrian Crossing), 37 (School Sign), 42 (Slow down) and 48 (Construction)	   	Misclassification of triangular warning signs and human figure-like objects	Signs with small, intricate human figures or symbols are harder to distinguish	Overlapping colors (yellow background) and shape confusion (triangle) contribute to errors
--	---	---	--	--

CHAPTER 6 – CONCLUSION AND RECOMMENDATION

6.1 Conclusion

Done by: Chew Li Yang

In this project, we aimed to develop a highly efficient and accurate traffic sign classification model for real-time use in autonomous driving systems. By employing an ensemble model that leverages EfficientNetB0, MobileNetV3Small, and Xception, we were able to reduce the model's size and complexity while maintaining competitive classification performance. This approach successfully addressed the critical needs for both accuracy and computational efficiency, making it suitable for deployment in embedded systems where resource limitations are significant.

The ensemble model achieved a test accuracy of 94.60%, with an F1-score of 93.93%. These results were comparable to other high-performing models in the field, such as the one in the referenced study [1], which achieved an accuracy of 96.41%. Although the accuracy of our model was slightly lower, it demonstrated a significant reduction in model size from 622.49 MB to 107.37 MB making it far more practical for real-time applications. This trade-off between accuracy and efficiency highlights the balance we sought to achieve with our model. The reduced computational overhead, combined with strong precision and recall values, demonstrates that the model is highly capable of performing under real-world conditions, including variations in lighting, weather, and image quality.

Throughout the training, validation, and testing phases, the model exhibited consistent strengths in recognizing well-defined and common traffic signs, such as speed limits and directional signs. However, it also revealed weaknesses in distinguishing between visually similar signs, especially those that share the same shape and color patterns, such as circular signs with arrows or triangular warning signs. These weaknesses suggest that while the ensemble model is robust, further improvements can be made to enhance its ability to differentiate between signs with subtle visual differences.

6.2 Recommendation (Future research directions)

Done by: Wong Sum Hui

Looking forward, there are several areas where this research can be expanded and improved. First, to address the issue of misclassification between visually similar traffic signs, additional data augmentation techniques or synthetic data generation could be employed to create more varied examples during training. This would allow the model to learn finer distinctions between similar signs. Additionally, incorporating more advanced techniques for feature extraction, such as attention mechanisms, could help the model focus on the most critical elements of each sign, reducing confusion between classes.

Another potential direction for future research is to explore more advanced real-time optimization techniques, such as pruning or quantization, which would allow for further reductions in model size and inference time without sacrificing accuracy. This would enhance the model's suitability for deployment in resource-constrained environments, such as low-power edge devices used in autonomous vehicles.

Finally, the model's generalizability could be improved by training on more diverse datasets that capture a wider range of traffic signs from different countries and environments. This would ensure the model's adaptability across various regions and driving conditions, ultimately contributing to safer and more reliable autonomous driving systems.

In conclusion, this project demonstrates that it is possible to create a highly efficient and accurate traffic sign classification model using an ensemble of pre-trained networks. While the model performed well in most cases, future work should focus on addressing its limitations in recognizing visually similar signs and improving its real-time efficiency. These improvements will pave the way for more robust and versatile traffic sign recognition systems in autonomous driving.

REFERENCES

- [1] S. Houben, et al., "Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark," *International Joint Conference on Neural Networks (IJCNN)*, 2013.
- [2] S. M. A. Islam, et al., "Traffic Sign Detection in Adverse Weather Conditions Using Region-Based Convolutional Neural Networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 7, pp. 2526-2536, July 2019.
- [3] A. García-Garrido, J. J. Yebes, L. M. Bergasa, R. Barea, and E. López, "Vision-based traffic sign recognition system for intelligent vehicles," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 12, no. 4, pp. 1176-1185, Dec. 2011.
- [4] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel, "Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark," *The 2013 International Joint Conference on Neural Networks (IJCNN)*, Dallas, TX, 2013, pp. 1-8.
- [5] J. Møgelmoose, M. M. Trivedi, and T. B. Moeslund, "Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 4, pp. 1484-1497, Dec. 2012.
- [6] Y. Yang, H. Luo, H. Xu, and F. Wu, "Towards Real-Time Traffic Sign Detection and Classification," *IEEE Transactions on Intelligent Transportation Systems*, vol.17,no.7, pp.2022–2031, Jul. 2016, doi: 10.1109/tits.2015.2482461.
- [7] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. IEEE CVPR*, 2005, vol. 1, pp. 886–893.
- [8] S. Salti, A. Petrelli, F. Tombari, N. Fioraio, and L. Di Stefano, "A traffic sign detection pipeline based on interest region extraction," in *Proc. IEEE IJCNN*, 2013, pp. 1–7
- [9] Y. Wu, Y. Liu, J. Li, H. Liu, and X. Hu, "Traffic sign detection based on convolutional neural networks," in *Proc. IEEE IJCNN*, 2013, pp. 1–7.
- [10] M. Liang, M. Yuan, X. Hu, J. Li, and H. Liu, "Traffic sign detection by ROI extraction and histogram features-based recognition," in *Proc. IEEE IJCNN*, Aug. 2013, pp. 1–8.

REFERENCES

- [11] A. K. Sangaiah, "Lightweight Deep Network for Traffic Sign Classification," in , vol. 74, no. 1-2, pp. 614-618, 2019. doi: 10.1007/s12243-019-00731-9.
- [12] Zhu, Z., Liang, D., Zhang, S., Huang, X., Li, B. and Hu, S. (2016). Traffic-Sign Detection and Classification in the Wild. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2110-2118. DOI: 10.1109/CVPR.2016.232.
- [13] B. Huval et al., "An empirical evaluation of deep learning on highway driving," *Robotics*, 2015. [Online]. Available: <https://doi.org/10.48550/arXiv.1504.01716>
- [14] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel, "Detection of traffic signs in real-world images: The German traffic sign detection benchmark," *The 2013 International Joint Conference on Neural Networks (IJCNN)*, Aug. 2013
- [15] F. Larsson and M. Felsberg, "Using Fourier descriptors and spatial models for traffic sign recognition," in *Proc. Scandinavian Conf. Image Analysis*, Heidelberg, Lecture Notes in Computer Science, vol. 6688, pp. 238-249, 2011.
- [16] S. Segvić, K. Brkić, Z. Kalafatić, V. Stanisavljević, M. Sevrović, D. Budimir, and I. Dadić, "A computer vision assisted geoinformation inventory for traffic infrastructure," in *Proc. IEEE Int. Conf. Intell. Transportation*, 2010, pp. 66-73.
- [17] R. Timofte, K. Zimmermann, and L. Van Gool, "Multi-view traffic sign detection, recognition, and 3d localisation," *Machine Vision and Applications*, December 2011.
- [18] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural Networks*, no. 0, pp. –, 2012. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0893608012000457>
- [19] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel, "Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark," in *International Joint Conference on Neural Networks* (submitted), 2013.
- [20] R. Timofte and L. Van Gool, "Sparse representation-based projections," in *BMVC*, 2011.
- [21] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, pp. 886–893, 2005.
- [22] P. Dollár, Z. Tu, P. Perona, and S. Belongie, "Integral channel features," in *BMVC*, 2009.

REFERENCES

- [23] R. Benenson, M. Mathias, T. Tuytelaars, and L. Van Gool, “Seeking the strongest rigid detector,” in CVPR, 2013.
- [24] R. Fisher, “The statistical utilization of multiple measurements,” *Annals of Eugenics*, vol. 8, pp. 376–386, 1938.
- [25] A. Martinez and A. Kak, “PCA versus LDA,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 23, no. 2, pp. 228–233, 2001.
- [26] R. Timofte and L. Van Gool, “Iterative nearest neighbors for classification and dimensionality reduction,” in CVPR, 2012.
- [27] J. Wright, A. Y. Yang, A. Ganesh, S. Sastry, and Y. Ma, “Robust face recognition via sparse representation,” *IEEE Transactions on Pattern Analysis and Machine Learning*, vol. 31, no. 2, February 2009.
- [28] D. Ciresan, U. Meier, J. Masci, and J. Schmidhuber, “Multi-column deep neural network for traffic sign classification,” *Neural Networks*, vol. 32, pp. 333–338, 2012.
- [29] X. Hou, S. J. Xie, and K. Wang, “Chinese Traffic Sign Database (TSRD),” available: <http://www.nlpr.ia.ac.cn/pal/trafficdata/recognition.html>.
- [30] S. Lim et al., “Efficient and Accurate Traffic Sign Recognition for Autonomous Driving Using an Ensemble of Pretrained CNNs,” *Journal of Sensor and Actuator Networks*, vol. 12, no. 33, 2023.
- [31] Hinton G, Vinyals O, Dean J (2014) Distilling the knowledge in a neural network. In: Proceedings of the advances in neural information processing systems (NIPS), pp 2644–2652
- [32] Huang G, Liu Z, Weinberger KQ (2017) Densely connected convolutional networks. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp 2261–2269
- [33] Ioffe S, Szegedy C (2015) Batch normalization: accelerating deep network training by reducing internal covariate shift. In: Proceedings of the international conference on machine learning (ICML), pp 448–456
- [34] Nair V, Hinton G (2010) Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the 27th international conference on machine learning (ICML), pp 807–814

REFERENCES

- [35] X. R. Lim, C. P. Lee, K. M. Lim, and T. S. Ong, "Enhanced Traffic Sign Recognition with Ensemble Learning," *J. Sens. Actuator Netw.*, vol. 12, no. 2, p. 33, Apr. 2023, doi: 10.3390/jsan12020033.

APENDICES

Setup for Libraries

```
!pip install imbalanced-learn
# Import necessary libraries
import os
import zipfile
import numpy as np
import pandas as pd
import cv2 as cv
import networkx as nx
from sklearn.model_selection import StratifiedShuffleSplit, ParameterGrid
from imblearn.over_sampling import RandomOverSampler
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Input, Dense, GlobalMaxPooling2D,
BatchNormalization, Dropout, LeakyReLU, Concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam, RMSprop, SGD
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping
from tensorflow.keras.applications import EfficientNetB0, MobileNetV3Small, Xception
from tensorflow.keras.regularizers import l2
from tensorflow.keras.utils import plot_model
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix
```

Image Acquisition

Done by: Chew Li Yang

```
# Unzip the dataset
def unzip_file(zip_path, extract_path):
    with zipfile.ZipFile(zip_path, 'r') as zip_ref:
        zip_ref.extractall(extract_path)

# Unzipping the dataset files
unzip_file('tsrd-train.zip', 'tsrd-train')
unzip_file('TSRD-Test.zip', 'TSRD-Test')
unzip_file('TSRD-Train Annotation.zip', 'TSRD-Train Annotation')
unzip_file('TSRD-Test Annotation.zip', 'TSRD-Test Annotation')
# Load the dataset as NumPy arrays
def load_data(image_dir, annotation_file):
    def preprocessing(image):
        image = cv.resize(image, (256, 256)) # Resize all images to (256, 256)
        image = image / 255.0 # Normalize the image
        return image

    images = []
    labels = []

    with open(annotation_file, 'r') as file:
        annotations = file.readlines()

    for annotation in annotations:
        parts = annotation.strip().split(';')
        image_file = parts[0]
        label = int(parts[-2]) # Use the second-last part as the class label
```

```

img_path = os.path.join(image_dir, image_file)
if os.path.exists(img_path):
    image = cv.imread(img_path)
    image = preprocessing(image) # Preprocess the image
    images.append(image)
    labels.append(label)

return np.array(images), np.array(labels)

# Load training and testing data and combine them
X_train, y_train = load_data('tsrd-train', 'TSRD-Train
Annotation/TsignRecgTrain4170Annotation.txt')
X_test, y_test = load_data('TSRD-Test', 'TSRD-Test
Annotation/TsignRecgTest1994Annotation.txt')
# Function to plot class distribution for a given dataset
def plot_class_distribution(labels, dataset_name):
    plt.figure(figsize=(12, 6))
    sns.countplot(x=labels, palette="viridis")
    plt.title(f'Class Distribution for {dataset_name}')
    plt.xlabel('Class')
    plt.ylabel('Frequency')
    plt.xticks(rotation=90)
    plt.show()

# Plot class distribution before oversampling
plot_class_distribution(y_train, 'Original Training Set (Before Oversampling)')
plot_class_distribution(y_test, 'Original Test Set')

```

Image Splitting, Image Preprocessing and Image Augmentation

Done by: Ang Qiao Yi & Wong Sum Hui

```
# Combine the datasets
X_combined = np.concatenate([X_train, X_test], axis=0)
y_combined = np.concatenate([y_train, y_test], axis=0)

# Moderate oversampling for rare classes (keep the overall dataset size under 10,000)
def moderate_oversample(X, y, max_size=10000):
    ros = RandomOverSampler(sampling_strategy='not majority', random_state=42) #
    Oversample only minority classes
    X_resampled, y_resampled = ros.fit_resample(X.reshape(X.shape[0], -1), y) # Reshape
    images to 2D for oversampling
    X_resampled = X_resampled.reshape(-1, 256, 256, 3) # Reshape back to original shape

    # Constrain the total dataset size to be under max_size
    if len(X_resampled) > max_size:
        X_resampled = X_resampled[:max_size]
        y_resampled = y_resampled[:max_size]

    return X_resampled, y_resampled

# Apply moderate oversampling
X_oversampled, y_oversampled = moderate_oversample(X_combined, y_combined,
max_size=10000)

# Stratified Split into Train and Test sets (split train into 80% and test into 20%)
sss_train_test = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_idx, test_idx in sss_train_test.split(X_oversampled, y_oversampled):
    X_train_new, X_test_final = X_oversampled[train_idx], X_oversampled[test_idx]
    y_train_new, y_test_final = y_oversampled[train_idx], y_oversampled[test_idx]
```

```
# Further Stratified Split of the training set into train (80%) and validation (20%)
sss_train_val = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_idx, val_idx in sss_train_val.split(X_train_new, y_train_new):
    X_train_final, X_val_new = X_train_new[train_idx], X_train_new[val_idx]
    y_train_final, y_val_new = y_train_new[train_idx], y_train_new[val_idx]
# Function to plot class distribution for a given dataset
def plot_class_distribution(labels, dataset_name):
    plt.figure(figsize=(12, 6))
    sns.countplot(x=labels, palette="viridis")
    plt.title(f'Class Distribution for {dataset_name}')
    plt.xlabel('Class')
    plt.ylabel('Frequency')
    plt.xticks(rotation=90)
    plt.show()

# Plot class distribution before oversampling
plot_class_distribution(y_train_final, 'Original Training Set (After Oversampling)')
plot_class_distribution(y_test_final, 'Original Test Set')
def display_dataset_info(X, y, dataset_name):
    num_samples = X.shape[0]
    num_classes = len(np.unique(y))
    print(f"Found {num_samples} validated image filenames belonging to {num_classes}
classes in {dataset_name}.")

# Display the number of samples and classes in each set
display_dataset_info(X_train_final, y_train_final, "Training Set")
display_dataset_info(X_val_new, y_val_new, "Validation Set")
display_dataset_info(X_test_final, y_test_final, "Test Set") # Include test set display
```


Model Architecture Design

Done by: Teoh Ming Xue

```
# Model architecture with Global Max Pooling instead of Flatten
def create_ensemble_model(input_shape, num_classes, dropout_rate=0.3, l2_reg=0.01):
    efficientnet_base = EfficientNetB0(weights='imagenet', include_top=False,
input_shape=input_shape)
    mobilenet_base = MobileNetV3Small(weights='imagenet', include_top=False,
input_shape=input_shape)
    xception_base = Xception(weights='imagenet', include_top=False,
input_shape=input_shape)

    for layer in efficientnet_base.layers + mobilenet_base.layers + xception_base.layers:
        layer.trainable = False
    inputs = Input(shape=input_shape)
    # Global Max Pooling to reduce model size
    combined_output = Concatenate()([
        GlobalMaxPooling2D()(efficientnet_base(inputs)),
        GlobalMaxPooling2D()(mobilenet_base(inputs)),
        GlobalMaxPooling2D()(xception_base(inputs))
    ])

    x = Dense(512, kernel_regularizer=l2(l2_reg))(combined_output)
    x = BatchNormalization()(x)
    x = LeakyReLU(alpha=0.2)(x)
    x = Dropout(dropout_rate)(x)
    x = Dense(512, kernel_regularizer=l2(l2_reg))(x)
    x = BatchNormalization()(x)
    x = LeakyReLU(alpha=0.2)(x)
    x = Dropout(dropout_rate)(x)
```

```

outputs = Dense(num_classes, activation='softmax')(x)
    model = Model(inputs=inputs, outputs=outputs)
    return model

```

Hyperparameter Tuning with Grid Search

Done by: Teoh Ming Xue

Define hyperparameter grid for Grid Search

```

param_grid = {
    'learning_rate': [0.001, 0.0001],
    'batch_size': [32, 64],
    'optimizer': [Adam, RMSprop, SGD],
}

```

```

best_accuracy = 0

```

```

best_params = None

```

```

best_model = None

```

Grid search without data augmentation

```

for params in ParameterGrid(param_grid):

```

```

    print(f"Testing Parameters: {params}")

```

Create a new model for each combination of hyperparameters

```

model = create_ensemble_model((256, 256, 3), len(np.unique(y_train_final)))

```

Set the optimizer with the current learning rate

```

optimizer = params['optimizer'](learning_rate=params['learning_rate'])

```

Compile the model with the specified optimizer

```

model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

```

```

# Callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2, min_lr=1e-
6)

# Train the model directly on the raw data (no data augmentation)
history = model.fit(
    X_train_final, y_train_final,
    validation_data=(X_val_new, y_val_new),
    epochs=3,
    batch_size=params['batch_size'],
    callbacks=[early_stopping, reduce_lr]
)

# Evaluate on the validation set
val_loss, val_accuracy = model.evaluate(X_val_new, y_val_new)
print(f"Validation Accuracy: {val_accuracy:.4f}")

# Keep track of the best model based on validation accuracy
if val_accuracy > best_accuracy:
    best_accuracy = val_accuracy
    best_params = params
    best_model = model

# Best hyperparameters found from the grid search
print(f"Best Hyperparameters: {best_params}")
print(f"Best Validation Accuracy: {best_accuracy:.4f}")

```

Model Training

Done by: Teoh Ming Xue

```
!pip install pydot
!pip install graphviz
# Best hyperparameters found manually from the grid search
best_params = {
    'batch_size': 64,
    'learning_rate': 0.0001,
    'optimizer': RMSprop # Using RMSprop as found during grid search
}

# Final training with the best hyperparameters
best_model = create_ensemble_model((256, 256, 3), len(np.unique(y_train_final)))
# Compile the model with the best hyperparameters
final_optimizer = best_params['optimizer'](learning_rate=best_params['learning_rate'])
best_model.compile(optimizer=final_optimizer, loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
# Callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=1e-6)
# Train the final model with the best hyperparameters
history_final = best_model.fit(
X_train_final, y_train_final,
validation_data=(X_val_new, y_val_new),
epochs=10, # You can change the number of epochs based on your needs
batch_size=best_params['batch_size'], # Manually set the best batch size
callbacks=[early_stopping, reduce_lr],
)
```

```
best_model.summary()

# Save the best model plot
def plot_and_save_model(model, file_name='model_plot.png'):
    # Plot and save model architecture
    plot_model(model, to_file=file_name, show_shapes=True, show_layer_names=True)
    print(f"Model architecture saved to {file_name}")

# Call the function to plot and save the architecture of the best model
plot_and_save_model(best_model, file_name='best_ensemble_model_plot.png')

# Evaluate the final model on the validation set
val_loss, val_accuracy = best_model.evaluate(X_val_new, y_val_new)
print(f"Final Validation Accuracy: {val_accuracy:.4f}")
```

Evaluation and Performance Metrics

Done by: Chew Li Yang

```
# ----- Final Model Evaluation -----

# Evaluate the best model on the training set
train_loss, train_accuracy = best_model.evaluate(X_train_final, y_train_final)
print(f"Final Training Accuracy: {train_accuracy:.4f}")

# Generate predictions for the training set
y_pred_train = np.argmax(best_model.predict(X_train_final), axis=-1)

# Evaluate the best model on the validation set
val_loss, val_accuracy = best_model.evaluate(X_val_new, y_val_new)
print(f"Final Validation Accuracy: {val_accuracy:.4f}")
```

```
# Generate predictions for the validation set
y_pred_val = np.argmax(best_model.predict(X_val_new), axis=-1)

# Evaluate the best model on the test set
test_loss, test_accuracy = best_model.evaluate(X_test_final, y_test_final)
print(f"Final Test Accuracy: {test_accuracy:.4f}")

# Generate predictions for the test set
y_pred_test = np.argmax(best_model.predict(X_test_final), axis=-1)

# ----- Performance Metrics for Training, Validation, and Test Sets -----
-----

# Training set metrics
train_acc = accuracy_score(y_train_final, y_pred_train)
train_precision = precision_score(y_train_final, y_pred_train, average='weighted')
train_recall = recall_score(y_train_final, y_pred_train, average='weighted')
train_f1 = f1_score(y_train_final, y_pred_train, average='weighted')

# Validation set metrics
val_acc = accuracy_score(y_val_new, y_pred_val)
val_precision = precision_score(y_val_new, y_pred_val, average='weighted')
val_recall = recall_score(y_val_new, y_pred_val, average='weighted')
val_f1 = f1_score(y_val_new, y_pred_val, average='weighted')

# Test set metrics
test_acc = accuracy_score(y_test_final, y_pred_test)
test_precision = precision_score(y_test_final, y_pred_test, average='weighted')
test_recall = recall_score(y_test_final, y_pred_test, average='weighted')
test_f1 = f1_score(y_test_final, y_pred_test, average='weighted')
```

```
# Print the results for each set
print(f"Training Accuracy: {train_acc:.4f}")
print(f"Training Precision: {train_precision:.4f}")
print(f"Training Recall: {train_recall:.4f}")
print(f"Training F1 Score: {train_f1:.4f}")

print(f"Validation Accuracy: {val_acc:.4f}")
print(f"Validation Precision: {val_precision:.4f}")
print(f"Validation Recall: {val_recall:.4f}")
print(f"Validation F1 Score: {val_f1:.4f}")

print(f"Test Accuracy: {test_acc:.4f}")
print(f"Test Precision: {test_precision:.4f}")
print(f"Test Recall: {test_recall:.4f}")
print(f"Test F1 Score: {test_f1:.4f}")

# ----- Confusion Matrices -----
# Confusion matrix for the training set
# Confusion matrix for the training set
cm_train = confusion_matrix(y_train_final, y_pred_train)
plt.figure(figsize=(15, 12)) # Increase the figure size to give more space
sns.heatmap(cm_train, annot=True, fmt='d', cmap='Blues',
            xticklabels=np.unique(y_train_final),
            yticklabels=np.unique(y_train_final),
            annot_kws={"size": 14}, # Increase annotation font size
            linewidths=.5) # Add slight separation between cells
plt.title("Training Set Confusion Matrix")
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.xticks(rotation=90) # Rotate x-axis labels to avoid overlap
plt.show()
```

```
# Confusion matrix for the validation set
cm_val = confusion_matrix(y_val_new, y_pred_val)
plt.figure(figsize=(15, 12)) # Increase the figure size to give more space
sns.heatmap(cm_val, annot=True, fmt='d', cmap='Blues',
            xticklabels=np.unique(y_val_new),
            yticklabels=np.unique(y_val_new),
            annot_kws={"size": 14}, # Increase annotation font size
            linewidths=.5) # Add slight separation between cells
plt.title('Validation Set Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.xticks(rotation=90) # Rotate x-axis labels to avoid overlap
plt.show()

# Confusion matrix for the test set
cm_test = confusion_matrix(y_test_final, y_pred_test)
plt.figure(figsize=(15, 12)) # Increase the figure size to give more space
sns.heatmap(cm_test, annot=True, fmt='d', cmap='Blues',
            xticklabels=np.unique(y_test_final),
            yticklabels=np.unique(y_test_final),
            annot_kws={"size": 14}, # Increase annotation font size
            linewidths=.5) # Add slight separation between cells
plt.title('Test Set Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.xticks(rotation=90) # Rotate x-axis labels to avoid overlap
plt.show()
```



```
# Function to plot accuracy and loss over epochs
def plot_learning_curves(history):
    # Plot training & validation accuracy values
    plt.figure(figsize=(12, 6))

    # Plot accuracy
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Accuracy', color='blue')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy', color='orange')
    plt.title('Model Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(loc='upper left')
    plt.grid(True)

    # Plot loss
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss', color='blue')
    plt.plot(history.history['val_loss'], label='Validation Loss', color='orange')
    plt.title('Model Loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(loc='upper left')
    plt.grid(True)

    plt.tight_layout()
    plt.show()
```

```
# Call the function to plot learning curves
plot_learning_curves(history_final)

import tensorflow as tf

from tensorflow.python.framework.convert_to_constants import
convert_variables_to_constants_v2


# Convert Keras model to a ConcreteFunction
model = best_model
full_model = tf.function(lambda x: model(x))
concrete_function = full_model.get_concrete_function(tf.TensorSpec([1, 256, 256, 3],
model.inputs[0].dtype))


# Convert ConcreteFunction to a frozen graph
frozen_func = convert_variables_to_constants_v2(concrete_function)
frozen_graph = frozen_func.graph


# Calculate FLOPs
run_meta = tf.compat.v1.RunMetadata()
opts = tf.compat.v1.profiler.ProfileOptionBuilder.float_operation()


# Use TensorFlow's profiler to calculate FLOPs
flops = tf.compat.v1.profiler.profile(graph=frozen_graph, run_meta=run_meta,
options=opts)


# Convert to GFLOPs
gflops = flops.total_float_ops / 10**9
print(f"GFLOPs: {gflops:.2f}")
```

Best Model Saving for deployment

Done by: Ang Qiao Yi

```
# Save the best model

from google.colab import drive
drive.mount('/content/drive', force_remount=True)
model_path = '/content/drive/My Drive/final_ensemble_model_ver4.keras'
best_model.save(model_path)
print(f"Model saved to {model_path}")
```

Prediction Lists (Classified and Misclassified)

Done by: Ang Qiao Yi

```
# Function to plot images (classified or misclassified)
def plot_images(X, y_true, y_pred, indices, title, num_images=10):
    """
    X: Images dataset (e.g., X_test_final)
    y_true: Ground truth labels (e.g., y_test_final)
    y_pred: Predicted labels (e.g., y_pred_test)
    indices: Indices of images to plot (either classified or misclassified)
    title: Title for the plot
    num_images: Number of images to display
    """

    # Limit the number of images to display
    num_images = min(num_images, len(indices))

    # Create subplots
    fig, axes = plt.subplots(2, num_images // 2, figsize=(15, 6))
    axes = axes.ravel()
```

```

for i in range(num_images):
    idx = indices[i]
    axes[i].imshow(X[idx].squeeze()) # Assuming images are (256, 256, 3)
    axes[i].axis('off')

    # Add title with true and predicted labels
    true_label = y_true[idx]
    pred_label = y_pred[idx]
    axes[i].set_title(f'True: {true_label}\nPred: {pred_label}')

plt.suptitle(title)
plt.tight_layout()
plt.show()

# Function to list and plot classified/misclassified images
def list_and_plot_classified_misclassified(X, y_true, y_pred, num_images=10):
    """
    X: Images dataset (e.g., X_test_final)
    y_true: Ground truth labels (e.g., y_test_final)
    y_pred: Predicted labels (e.g., y_pred_test)
    num_images: Number of images to display

    """
    # Get indices of classified and misclassified images
    classified_indices = np.where(y_true == y_pred)[0]
    misclassified_indices = np.where(y_true != y_pred)[0]

    print(f"Number of correctly classified images: {len(classified_indices)}")
    print(f"Number of misclassified images: {len(misclassified_indices)}")

```

```

# Plot classified images
plot_images(X, y_true, y_pred, classified_indices, title='Correctly Classified Images',
num_images=num_images)

# Plot misclassified images
plot_images(X, y_true, y_pred, misclassified_indices, title='Misclassified Images',
num_images=num_images)

# Generate predictions for the test set
y_pred_test = np.argmax(best_model.predict(X_test_final), axis=-1)

# List and plot classified and misclassified images
list_and_plot_classified_misclassified(X_test_final, y_test_final, y_pred_test,
num_images=20)

```

UI Design

Step 1: Install Gradio if it's not already installed

```
!pip install gradio
```

Step 2: Import necessary libraries

```
import gradio as gr
```

```
import numpy as np
```

```
from tensorflow.keras.models import load_model
```

```
from tensorflow.keras.utils import plot_model
```

```
import cv2
```

```
from google.colab import drive
```

```
# Load the pre-trained model (replace 'model_path' with your model file)
drive.mount('/content/drive', force_remount=True)
model_path = '/content/drive/My Drive/final_ensemble_model_ver4.keras'
best_model = load_model(model_path)

# Preprocessing function (similar to how you preprocessed the training data)
def preprocess_image(image):
    # Resize the image to (256, 256), the size used during model training
    image = cv2.resize(np.array(image), (256, 256)) # Ensure image is a numpy array before
    resizing
    # Normalize the image
    image = image / 255.0
    # Expand dimensions to match model input (1, 256, 256, 3)
    image = np.expand_dims(image, axis=0)
    return image

# Prediction and Analysis Function
def classify_and_analyze(image, ground_truth=None, filter_type="All"):
    # Preprocess the image
    processed_image = preprocess_image(image)
    # Get the prediction from the model
    prediction = best_model.predict(processed_image)
    # Get the predicted class (highest probability)
    predicted_class = np.argmax(prediction, axis=1)[0]

    # Debug: Print the values being passed in for testing
    print(f"Predicted class: {predicted_class}")
    print(f"Ground Truth provided: {ground_truth}")
```

```
# Ensure that ground truth is properly captured and is not mistakenly set to 0 if None
if ground_truth is not None:
    ground_truth = int(ground_truth) # Ensure ground truth is an integer
    classification_result = "Correct" if ground_truth == predicted_class else "Misclassified"
else:
    classification_result = "Ground Truth not provided"

# Handle filter logic based on the user's selection
if filter_type == "Classified" and classification_result == "Misclassified":
    return "Image is misclassified. (Filtered out based on your selection)"
elif filter_type == "Unclassified" and classification_result == "Correct":
    return "Image is correctly classified. (Filtered out based on your selection)"

# Return the result with or without ground truth
return f"Predicted class: {predicted_class}, Ground Truth: {ground_truth if ground_truth
is not None else 'None'}, Result: {classification_result}"

# Create the Gradio interface
iface = gr.Interface(
    fn=classify_and_analyze, # Function that performs classification and filtering
    inputs=[
        gr.Image(), # Input as an image
        gr.Number(label="Ground Truth Class (Optional)", # Optional ground truth input
        gr.Dropdown(["All", "Classified", "Unclassified"], label="Filter Results", value="All")
    # Filter dropdown
    ],
```

```
outputs="text", # Output as text (predicted class and result)

    title="Traffic Sign Classification with Error Analysis", # Title of the UI

    description="Upload an image of a traffic sign, optionally provide the ground truth class,
and toggle between classified/unclassified images."

)

# Step 7: Launch the UI
iface.launch()
```