ASSIGNMENT 2: ROAD SIGN CLASSIFICATION

COSC2673
MACHINE LEARNING
UNDERGRADUATE

By
Oisin Aeonn (s3952320)
&
Vince Quach (s3900481)

Table of Contents

Introduction

For *Assignment 2* in **COSC2673 Machine Learning** at **RMIT University**, we selected '**Project 1: Classify Images of Road Traffic Signs**'. The primary goal of this project was to develop an end-to-end machine learning algorithm capable of performing an n-ary classification of a road sign's shape and type. We used the provided modified version of the **BelgiumTS Dataset** for training and validation. Additionally, we aimed to create a robust and generalisable model that can accurately classify road signs from an independently sourced dataset, ensuring its effectiveness in a real-world scenario.

Road sign recognition plays a crucial role in intelligent transportation systems and autonomous vehicles, enhancing overall road safety. By employing advanced machine learning techniques, including deep learning architectures such as Convolutional Neural Networks (CNNs), and leveraging insights from domain experts, we developed a reliable road sign classification model that could be integrated into these systems. In the subsequent sections of this report, we will detail our approach, present our ultimate judgement, discuss the challenges encountered and insights gained throughout the project.

General Approach

Our approach to develop a robust and generalisable road sign classifier followed the comprehensive Machine Learning Workflow methodology (Datacamp, 2022 & Alavi et al. RMIT 2024), with some key elements borrowed from Cross-Industry Standard Process for Data Mining (CRISP-DM) (RunAI, 2024). We settled on creating side-by-side shape, and type classifier.

As seen in the **Introduction**, we began by defining the problem of road sign classification and its significance in the context of intelligent transportation systems. Next, we conducted an extensive Exploratory Data Analysis (EDA), outlier handling, labelling of the dataset, before pre-processing the dataset. We could finally then split the dataset into training and validation sets. Additionally, we utilised Linear Discriminant Analysis (LDA) to explore the relationships between different road signs, as suggested in the original 2013 paper by Mathias et al. [pp. 4-8] on traffic sign recognition. As well as utilised a few unsupervised techniques such as K-Means to extrapolate, and cluster images together to extract relevant topics.

We investigated various machine learning algorithms and justified our selection based on their suitability for image classification. The selected models were optimised using dimensionality reduction and hyperparameter tuning techniques. Our hyperparameter tuning involved several approaches, including manual tuning, Bayesian optimisation, and grid search. Primarily, we tuned regularisation parameters (L1 and L2) to prevent overfitting, batch size for optimal performance, adjusting tree and neural network architectures, dropout rates to encourage robust feature learning, transfer learning for improved generalisation, and early stopping to mitigate overfitting. Model evaluation was performed using the appropriate classification metrics: F1-score, accuracy, precision, and recall. These metrics enabled us to assess model performance on our validation set, identifying the strengths and weaknesses of each model and informing our final model selection.

Finally, for our independent evaluation, which simulates real-world implementation, we rigorously collected an independent dataset using the scientific method. This dataset consisted of road sign images from various countries, to assess the generalizability and performance of our trained models in practical scenarios. By applying our models to this unseen data, we aimed to evaluate their effectiveness and robustness in accurately classifying road signs across diverse geographical locations and imaging conditions.

By following this machine learning workflow methodology, we developed a robust and accurate road sign classification model, emphasising the iterative nature of model development and continuous refinement to achieve the best possible performance and generalisation capability. We will now go into depth explaining the rationale behind each step.

Exploratory Data Analysis (EDA)

We began the EDA phase by examining the dataset, which consists of 3,699 images categorised using a folder hierarchy. To understand the dataset, we plotted and made observations of the following key features:

- ***Sharpness: The distribution is positively skewed, indicating most images have lower sharpness.***
- ***Pixel Intensity and Image Similarity: Both follow a roughly Gaussian distribution.***
- ***Entropy: The distribution is negatively skewed, with most images having higher entropy.***
- ***Image Size and Format: All images are consistently 28×28 pixels, grey scale, and in .PNG format.***

The image similarity analysis demonstrated significant diversity within the dataset, suggesting that it has already undergone pre-augmentation. This solidified the recommendation not to augment the dataset further, as it has been carefully curated and pre-augmented for our task. No images in the dataset are exactly equal to another, but there are some that are very similar, as indicated by the similarity score.

This initial EDA provided valuable insights into the dataset's characteristics and diversity, guiding our subsequent steps in the machine learning workflow. In the next step, we will go further into detail of how we ingested the data and further explore the variance in the labelling.

Handling Low Quality Data

In the dataset there are examples of road signs that are obstructed or irregular (see Figure 1), creating edge cases for certain classes. We removed this low quality data to attempt to reduce bias and maintain consistency across different sign types and shapes. Including challenging examples is often beneficial, but inconsistencies across classes can create bias, leading the model to potentially overpredict a class that has an edge case when given an abnormal input leading to a misclassification. For example in the Figure 1 case any blurry input may be classified as a Hexagon Stop Sign. Removing edge cases from certain categories during training could make the model struggle with similar cases during evaluation. To balance data quality and consistency, we removed identified edge cases, benchmarking the performance on the original and cleaned datasets, and verified performance improvement while maintaining parity across all traffic sign categories.

Figure 1: "Stop Sign"

Data Ingestion and Pre-processing

After conducting the initial EDA, we proceeded to the data ingestion and pre-processing stage. We noticed that the image names themselves were insignificant for our task, and the crucial information was contained in the directory structure where the images were stored. The directory hierarchy represented the labels for each image, with the first level indicating the 5 shapes and the second level indicating the 17 types of road sign.

To prepare the data for a supervised machine learning model, we needed to create a data structure that explicitly captures the true labels for each image. We opted to create a Pandas DataFrame containing the following three lists:

1. ***image_paths: Stores the relative path, allowing access to the pixel data.***
2. ***shapes: Stores the corresponding shape label for each image.***
3. ***types: Stores the corresponding type label for each image.***

We implemented a custom data labelling function to iterate through the dataset directory and populate these lists. The function traversed the directory structure, extracting the image paths and their corresponding shape and type labels.

To convert the categorical labels into numerical representations suitable for our model, we utilised the LabelEncoder from the sklearn.preprocessing module. We encoded the shape and type labels separately, ensuring that each unique label was assigned a numerical value.

To ensure the integrity of our dataset labelling process, we performed a verification step. We confirmed that the entire training dataset was successfully labelled, satisfying the following condition: ∀ DataFrame ∃ Label (image_path) ∧ Label (shape) ∧ Label (type).

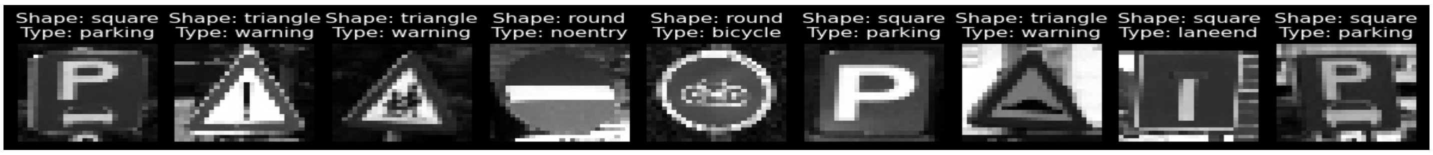


Figure 2: Images with their corresponding Shape and Type Labels.

By completing the data ingestion and pre-processing stage, we transformed the raw dataset into a structured format suitable for training and evaluating a supervised machine learning model as requested in the brief. Upon further exploration of the DataFrame (refer to Figure 2), we can observe that all images are classified correctly.

EDA (Part 2)

Now that we have a labelled DataFrame, we can do some further EDA on our dataset to identify any class imbalances and make further observations. Upon analysing the class distribution within the dataset, we observe a significant class imbalance across both shape and type categories as seen in Figure 3. Some classes have a considerably higher number of samples compared to others, which can potentially lead to biased predictions if not addressed properly. To address the class imbalance issue, we will test techniques like:

- ***Class Weighting: Assigning higher importance to minority classes during training.***
- ***Oversampling: Increasing the number of minority class samples through duplication or synthetic data generation.***
- ***Undersampling: Reducing the number of majority class samples to match the minority classes.***

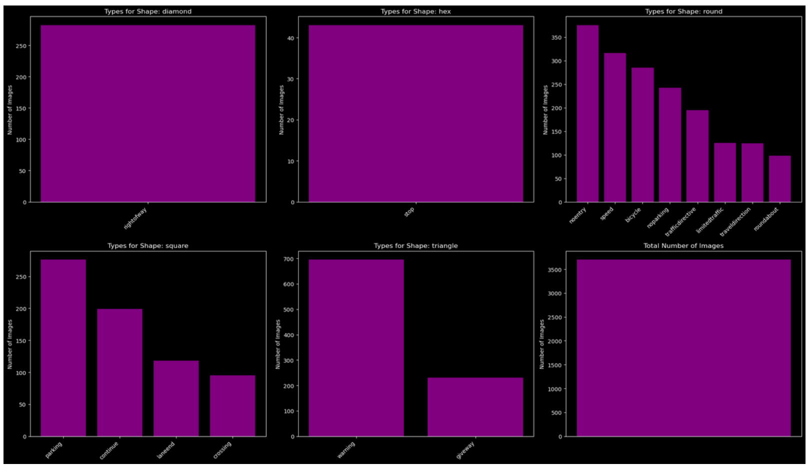


Figure 3: Class Distributions for Shape, and Type

Data Splitting and Leak Checking

We split the dataset into training and validation sets using an 80/20 ratio to ensure reliable evaluation of our model's performance. It's crucial to check for data leakage between the sets, as it can lead to overly optimistic performance estimates.

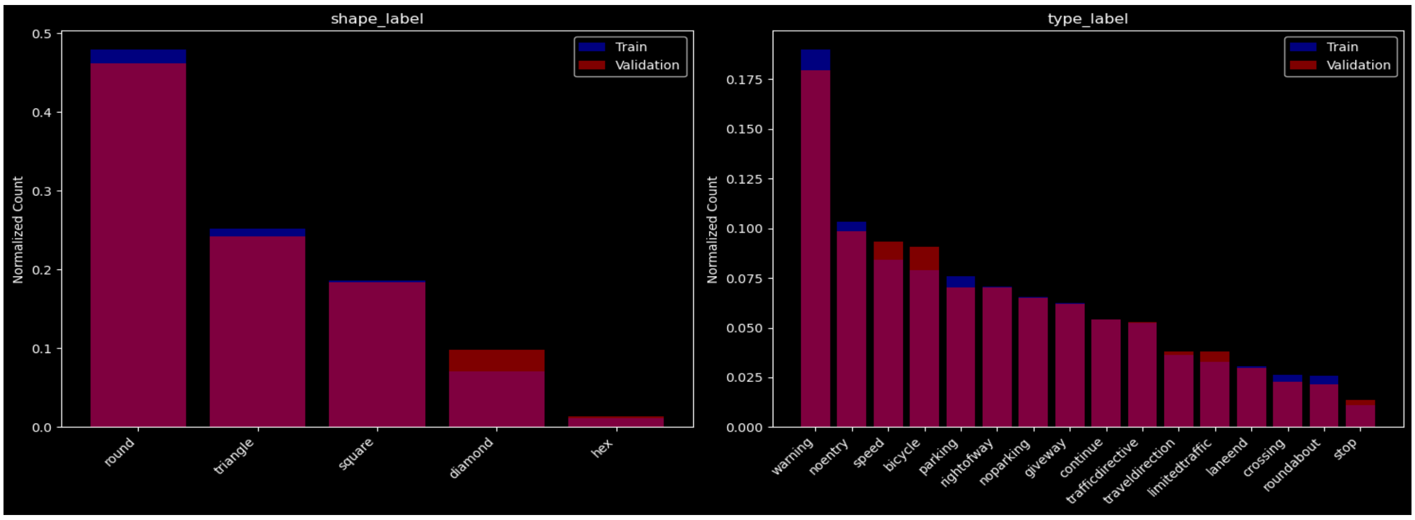


Figure 4: Training and Validation Data Leakage

Figure 4 illustrates the normalised distribution of samples, where purple indicates overlap, blue represents more training data, and red represents more validation data. The analysis shows minimal data leakage, with small discrepancies considered acceptable.

Dimensionality Reduction with Principal Component Analysis (PCA)

PCA is explored as an optional step to reduce the dimensionality of the high-dimensional image data. By extracting essential features and discarding irrelevant information, PCA aims to improve the predictability and efficiency of our traffic sign classification model. Experimenting with different numbers of principal components and evaluating their impact on model performance helps determine the optimal level of dimensionality reduction for our specific task. We predict this will make a small impact on performance, and is worth implementing.

Fixing Class Imbalances

Due to the relatively small size of our dataset and the presence of significantly underrepresented classes, such as the Hexagon Stop Sign, which constitutes only around 1% of the total samples, undersampling is not feasible as it would result in insufficient data for generalisation. Therefore, to address class imbalance, we are left with two optional steps in our notebook: class weights and oversampling. Class weights involve assigning higher importance to minority classes during model training, allowing them to have a greater influence on the model's learning process. On the other hand, oversampling techniques aim to increase the number of samples in underrepresented classes to achieve a more balanced class distribution. However, it's important to be aware of the potential drawbacks associated with these methods. Oversampling, in particular, can lead to overfitting if not used carefully, as the model may learn to memorise the duplicated samples instead of generalising well to unseen data. Additionally, these techniques assume that the class distribution follows a Gaussian distribution in the real world, which may not always be the case. To make an informed decision on whether to incorporate oversampling or class weights into our final model, we will compare the performance of models trained with and without these techniques.

Evaluation Metrics

Before diving into our model development, it's crucial to establish a clear understanding of the metrics used to evaluate a model's performance. We will primarily focus on utilising confusion matrices, which provide a comprehensive view of the model's performance by presenting the counts of true positive, true negative, false positive, and false negative predictions for each class. The major benefit of confusion matrices is the ability to quickly glance at them and look for a diagonal trend from top left to bottom right to see the correct predictions. This allows us to calculate various evaluation metrics such as accuracy, precision, recall, and F1-score.

In addition to confusion matrix-based metrics, we will also consider other evaluation metrics such as the training error, loss function, and number of epochs. Monitoring the loss over epochs helps us assess the model's learning progress and identify potential issues like overfitting or underfitting.

Primarily we will focus on accuracy as it is the overall correctness of all predictions. As optimists, we put our target accuracy high aiming for 90% on shape, and 80% for type in our validation set. We predicted that when using an unseen dataset for our independent evaluation that performance would go down by 10-15% as we would have a fair amount of variance introduced with road signs from different countries. For further details on evaluating classification models, please refer to the **COSC2673 Machine Learning** Lecture 1 Discussion Forum Post by Me (**Oisin**).

Non-Neural Network Models

Before establishing a baseline model, we explored several non-neural network algorithms to ensure we didn't overlook a simple solution for road sign classification. These models, such as logistic regression and tree-based algorithms (Decision Trees, Random Forests, and XGBoost), are generally more simple, interpretable, and less computationally expensive compared to neural networks. They provide insights into the relationship between input features and the target variable, making them valuable in the initial stages of model development.

Evaluating the performance of these models on our specific classification task allowed us to assess their suitability and determine if they could achieve satisfactory results without the need for more complex architectures like neural networks. Some models, for example logistic regression, can effectively handle high-dimensional data like that which is present our image dataset without extensive feature engineering or dimensionality reduction techniques.

On the other hand tree-based ensemble methods, such as Random Forests and XGBoost, leverage multiple weak learners to create a strong and robust classifier, often outperforming individual models. By exploring non-neural network models first, we gained valuable insights and made informed decisions about the need for more advanced techniques as well as what performance metrics we should minimally get given the computational cost.

Hyperparameter Tuning

To find the optimal hyperparameters for each model, we employed a grid search, which is a brute-force method that exhaustively searches through a specified parameter grid cross validating the results to find the optimal combination. We also tried a Bayesian search, but found we got the optimal results using a grid search using the default of 5 for cross validation. The hyperparameters tuned for each model, and their associated best combination for shape, and type were:

- ***Decision Tree: criterion=entropy, max_depth=none, min_samples_leaf=1, min_samples_split=5***
- ***Random Forest: max_depth=20, min_samples_leaf=1, min_samples_split=2, n_estimators=200***
- ***XGBoost: subsample=0.1, n_estimators=5, max_features=sqrt, max_depth=100, learning_rate=X***

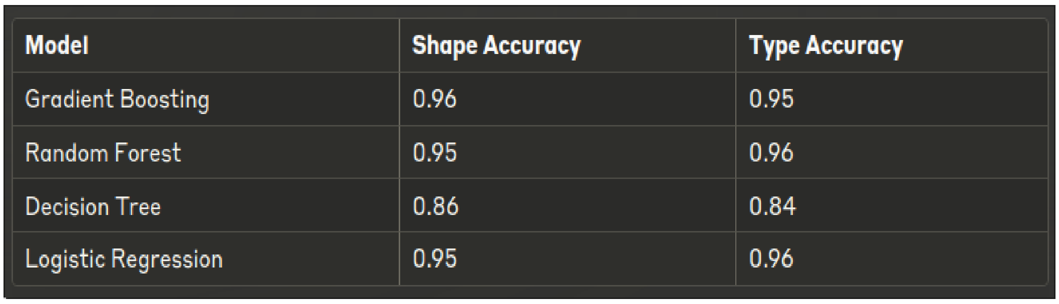
Where X is 0.8 for shape, or 1.0 for type.

The rationale for the selected hyperparameters to tune, and the chosen parameter grid was based on a few key considerations. We aimed to keep the number of grid parameters between 2 and 4 values, with a preference for 3, to strike a balance between computational efficiency and thorough exploration of the hyperparameter space. For more computationally expensive models like XGBoost, we opted for 2 values to manage the training time. The parameter inputs were varied to cover a spectrum from default settings to more aggressive configurations, allowing us to find the optimal trade-off between model complexity and generalisation performance. In the case of the simpler Decision Tree model, we were able to explore a wider range of inputs within a reasonable amount of time, enabling a more comprehensive search for the best hyperparameter combination.

Results

The validation results in Figure 5 show that Gradient Boosting, Random Forest, and Logistic Regression achieve very high accuracy scores all above 95% for both shape and type classification tasks, while Decision Trees exhibit a relatively lower accuracy of ~85% compared to the other models which is explainable by the ensemble combination of weak learners being optimal for tasks dealing with images. However, these high validation scores are difficult to take seriously as our model includes highly augmented images that could contribute to our model just remembering answers, verses being generalisable.

When applying PCA we don’t find a significant difference on our validation dataset, but we predict this will improve when applying our model to the independent dataset.



Model	Shape Accuracy	Type Accuracy
Gradient Boosting	0.96	0.95
Random Forest	0.95	0.96
Decision Tree	0.86	0.84
Logistic Regression	0.95	0.96

Figure 5: Non-Neural Network Validation Results

Neural Networks

After exploring various non-neural network models and evaluating their validation performance, we proceeded to investigate neural network architectures. Neural Networks are an obvious next step as they are often praised to be able to solve any problem by *"approximat[ing] any function to any desired degree of accuracy"* (**George Cybenko 1991**). Given the success of neural networks, particularly CNNs, in image classification tasks, we decided to that we would leverage their ability to learn hierarchical representations and capture complex patterns in visual data. However, we'll start off small, and simple with an MLP baseline model.

Baseline Model: Multi-Layer Perceptron (MLP)

An MLP is a type of feedforward neural network that can learn to classify images by automatically extracting relevant features from the input data. MLPs consist of multiple layers of interconnected neurons that transform the input image through a series of non-linear operations, enabling them to learn complex decision boundaries and capture intricate patterns.

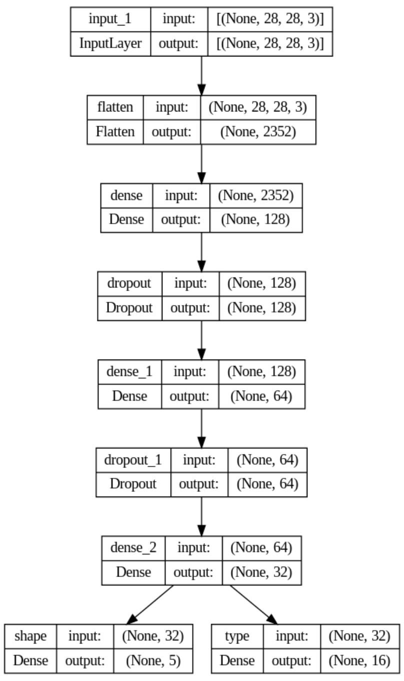
Our Model Architecture (see Figure 6)

We used a Flatten layer to convert the 2D input image (28×28) into a 1D vector because it is necessary to prepare the data for the subsequent fully connected layers. Flattening the input allows the model to treat the image as a simple vector, enabling it to learn global patterns and relationships between pixels.

We experimented with the Dense layer architecture, starting with 128 units in the first hidden layer to provide sufficient capacity for capturing intricate features. For subsequent layers, we gradually compressed the learned representations by halving the units in each layer, going from 128 to 64 and then to 32. We hoped this approach would help the model abstract higher-level features, focus on essential information, reduce overfitting, and maintain expressive power while learning more abstract and discriminative patterns for classification. ReLU activation introduced non-linearity to learn complex patterns effectively.

We included a Dropout layer with a rate of 0.3 because we wanted to experiment with regularising the model and preventing overfitting. Dropouts encourage the model to learn more robust and generalised features by randomly setting some input units to 0 during training. This reduces the reliance on specific neurons and enhances the model's ability to generalise to unseen data, he hoped this would potentially lead to greater performance on the independent evaluation.

All of our neural networks used two separate output layers, one for shape (5 classes) and another for type (17 classes), because the task involves predicting both the shape and type of road signs. Softmax activation in each output layer produces probability distributions over the respective classes, providing clear and interpretable outputs for the classification task.


input_1 | InputLayer | input: [(None, 28, 28, 3)] | output: [(None, 28, 28, 3)]
flatten | Flatten | input: (None, 28, 28, 3) | output: (None, 2352)
dense | Dense | input: (None, 2352) | output: (None, 128)
dropout | Dropout | input: (None, 128) | output: (None, 128)
dense_1 | Dense | input: (None, 128) | output: (None, 64)
dropout_1 | Dropout | input: (None, 64) | output: (None, 64)
dense_2 | Dense | input: (None, 64) | output: (None, 32)
shape | Dense | input: (None, 32) | output: (None, 5)
type | Dense | input: (None, 32) | output: (None, 16)
Figure 6: NLP Baseline Model Architecture

Optimisation

When optimising our neural network models, we employed various techniques to improve performance and address fitting issues. To identify these problems, we monitored the loss function during training. If the model struggled to learn and the loss remained high, we thought the model was underfitting. On the other hand, if the training loss decreased while the validation loss increased, it suggested overfitting. To tackle underfitting, we experimented with increasing the number of neurons in the hidden layers, enhancing the model's capacity to learn complex patterns. However, we were cautious not to add too many neurons to avoid overfitting.

To prevent overfitting, we implemented several techniques:

- ***Early stopping: We monitored the validation loss and stopped training when the model's performance on unseen data started to degrade, striking a balance between model complexity and generalisation ability.***
- ***Regularisation: We applied regularisation (predominantly L1) to reduce overfitting by adding penalty terms to the weights, encouraging sparsity (L1) or keeping the weights small (L2).***
- ***Dropout: We incorporated dropout layers, which randomly set a fraction of the input units to 0 during training, forcing the network to learn more robust and distributed representations of the data.***

We performed our hyperparameter tuning manually to find the optimal combination of learning rate, batch size, number of layers, and neurons in each layer. This allowed us to efficiently explore the parameter space and find the best configurations for our classification task, given the computational constraints.

By employing these optimisation techniques and monitoring the relevant evaluation metrics, we aimed to improve the performance and generalisation ability of our neural network models, effectively addressing underfitting and overfitting issues.

Incremental Improvement

The Flatten layer remains unchanged to convert the 2D input image into a 1D vector. We increased the units in the dense layers to 512, 256, and 128, enhancing the model's capacity to learn complex patterns at different levels of abstraction. Batch Normalisation layers were added after each dense layer to normalise activations, improve stability, and reduce internal covariate shift. LeakyReLU replaced ReLU to address the "dying ReLU" problem and learn more complex patterns. The dropout rate was increased to 0.4 for stronger regularisation and prevention of overfitting. We switched to the Adam optimiser with a lower learning rate of 0.001 for stable convergence and introduced a learning rate scheduler (ReduceLROnPlateau) to dynamically adjust the learning rate based on validation performance, fine-tuning weights and finding the optimal solution.

Performance

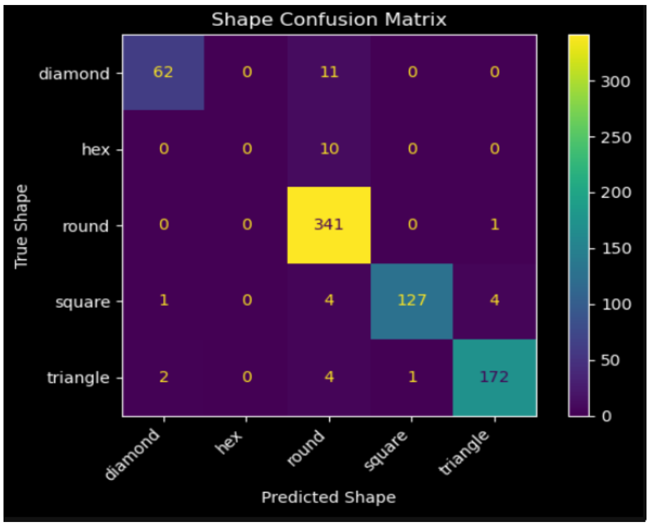


Figure 7: Baseline Model Shape Performance

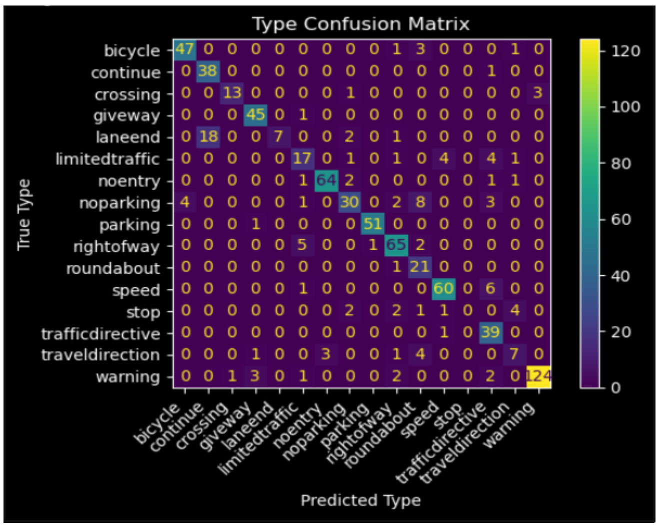


Figure 8: Baseline Model Type Performance

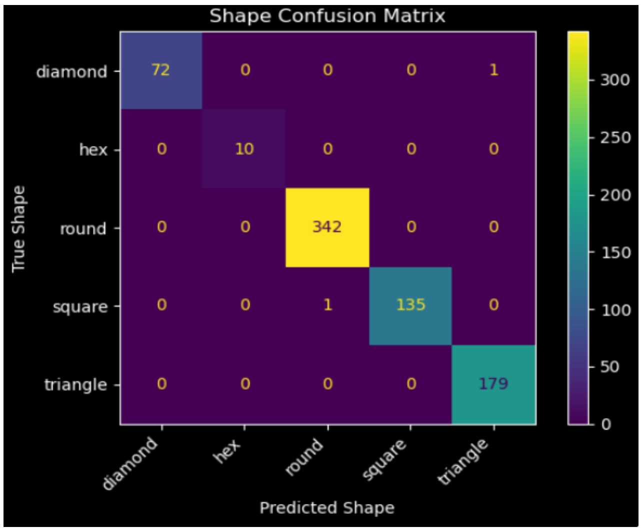


Figure 9: Improved MLP Shape Performance

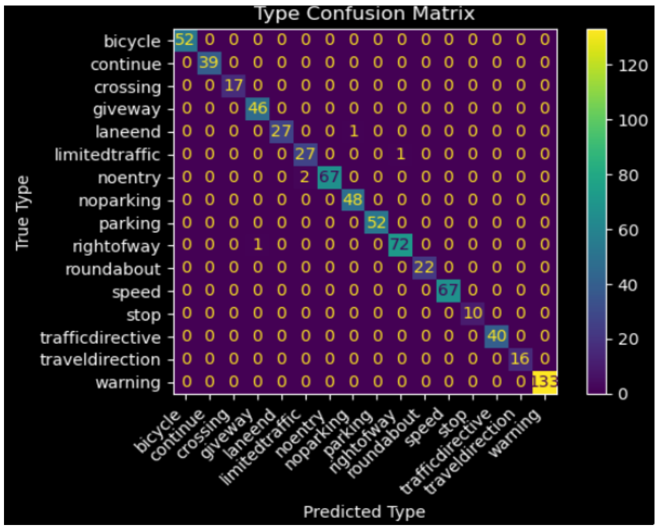


Figure 10: Improved MLP Type Performance

In terms of performance our optimised model significantly outperformed the baseline model in terms of overall accuracy. For shape classification, the accuracy improved from 92% to an impressive 100%, while for type classification, it increased from 82% to 99%. Figures 7-10 present confusion matrices with high diagonal patterns, indicating numerous correctly predicted images. Notably, in our improved model (Figures 9 and 10), the hexagon class is classified with 100% accuracy, a substantial improvement from 0% in the original model. All other classes also exhibited enhanced performance. These results demonstrate the effectiveness of our optimisation techniques in achieving near-perfect classification accuracy.

Advanced Model: Convolutional Neural Networks (CNNs)

CNNs on the other hand are more powerful than MLPs. They have emerged as an efficient architecture for image classification tasks. CNNs introduce convolutional layers that apply learned filters to the input image, enabling them to capture local spatial patterns and hierarchical features while being invariant to translations and distortions. These features of CNNs make them particularly well-suited for road sign classification, as they can effectively handle the variations in shape, orientation, and appearance of traffic signs. Thus, why we opted, and knew this would be the best methodology for the task.

CNN Architecture

When constructing our CNN architecture, we followed an approach similar to our MLP findings to create a simplified version that works effectively. We used two convolutional layers with 16 and 32 filters, respectively, followed by max pooling layers, as we predicted this configuration would be effective in capturing relevant features while maintaining computational efficiency. The output of the convolutional layers is flattened and passed through a dense layer with 64 units, allowing for the learning of high-level representations. We incorporated two separate output layers for shape and type classification, both using the softmax activation function, because it enables the model to produce probability distributions over the respective classes. We compiled the model with the Adam optimiser and sparse categorical cross-entropy loss, as this combination has been shown to work well for multi-class classification tasks. See **Section 4.2** Convolutional Neural Networks (CNNs) in **Appendix A: Jupyter Notebook**.

Incremental Improvement

After analysing the initial CNN architecture results, which showed a validation loss of 0.0970, validation shape loss of 0.0323, validation type loss of 0.0647, validation shape accuracy of 0.9918, and validation type accuracy of 0.9851 (see Figure 11), we applied optimisation techniques similar to those used in the MLP to further improve the model's performance.

We incorporated L2 regularisation in the convolutional and dense layers to prevent overfitting by adding a penalty term to the loss function, as it constrains the model's weights and encourages simpler and more generalise representations. Batch normalisation was added after each convolutional layer to normalise the activations and improve training stability, as it reduces the internal covariate shift and allows for faster convergence.

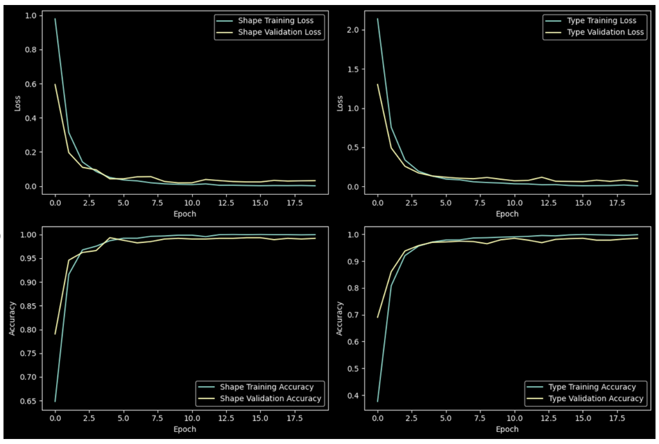


Figure 11: CNN Loss, and Accuracy Functions

We increased the number of filters in the convolutional layers to 32, 64, and 128, respectively, to enhance the model's capacity to capture more complex features at different scales, as deeper and wider networks have shown to be more effective in learning hierarchical representations. We also experimented with the LeakyReLU activation in the third convolutional layer to mitigate the problem of "dying ReLU" by allowing small negative values, which would help maintain gradient flow and prevents neurons from becoming inactive.

We expanded the dense layer to 256 units, and altered the dropout layer to have a rate of 0.6 to attempt to enhance the model's capacity and regularisation capabilities, as we previously mentioned dropouts randomly sets a fraction of the input units to zero during training, forcing the network to learn more robust and distributed representations.

Furthermore, we employed callbacks such as early stopping and learning rate reduction on plateau to prevent overfitting and fine-tune the model's performance. Early stopping monitors the validation loss and stops training if no improvement is observed for 10 epochs, ensuring that the model does not overfit to the training data. The learning rate is reduced by a factor of 0.1 if the validation loss does not improve for 5 epochs, allowing for fine-tuning of the model's weights in the later stages of training and helping the model converge to a better optimum.

These architectural choices and optimisation techniques were implemented to enhance the model's performance, generalisation ability, and computational efficiency for the road sign classification task, building upon the insights gained from the initial CNN architecture results.

Performance

Our optimised CNN model achieved remarkable performance (see Figures 12 and 13), with a validation accuracy of approximately 100% for both shape and type classification, surpassing the initial architecture as well as going far beyond our pre-set optimisation goals. No class is being misclassified, which is precisely what we aimed for. These results demonstrate the effectiveness of our optimisation techniques, and nature of CNNs in achieving near-perfect classification accuracy for road sign classification. We also expect a higher generalizability from the improved CNN model compared to the initial architecture, and MLPs, as it is a more robust and reliable model capable of performing in real-world scenarios.

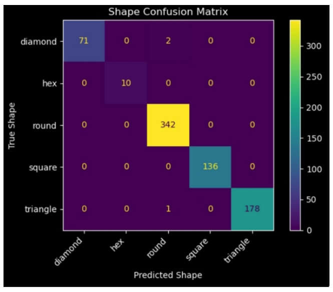


Figure 12: Optimised CNN Shape

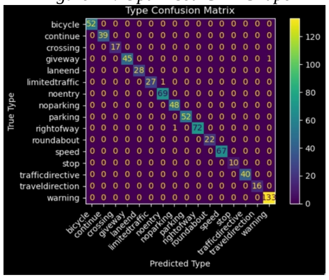


Figure 13: Optimised CNN Type

Advanced Tuning

We also experimented with a third model that utilised advanced tuning, such as transfer learning, to further improve generalizability. However, this proved to be challenging, as many pre-trained models require resizing images which led to distortion and compromising the quality of the extracted features. However, we will explore the performance of the generalizability of these in our independent evaluation.

Independent Dataset Sourcing

To collect our independent evaluation dataset, we followed a rigorous scientific method to ensure the data's quality and diversity. We sought insights from domain experts at **RMIT University**.

Based on the guidance from our domain expert and several scientific journals, we focused on gathering data from various sources to create a diverse and representative dataset. We collected 511 road sign images from the **German Traffic Sign Recognition Benchmark (GTSRB) Dataset**, a sister dataset to the **BelgiumTS Dataset** used for training our model. Additionally, we captured 68 **Greater Melbourne Region** road sign images using an iPhone 12 (**Oisin**) and an iPhone 13 (**Vince**). We also sourced 106 Belgium Road Signs from **Google Maps** to test the dataset against unseen data from the same country. In total, we created a dataset of 689 images for our independent evaluation, comparable to the validation dataset and equating to 20% of the training dataset size, as recommended by **Professor Alavi**. Please see Figure 14 for the true distribution of the number of samples per country and a detailed breakdown of their shapes and types.

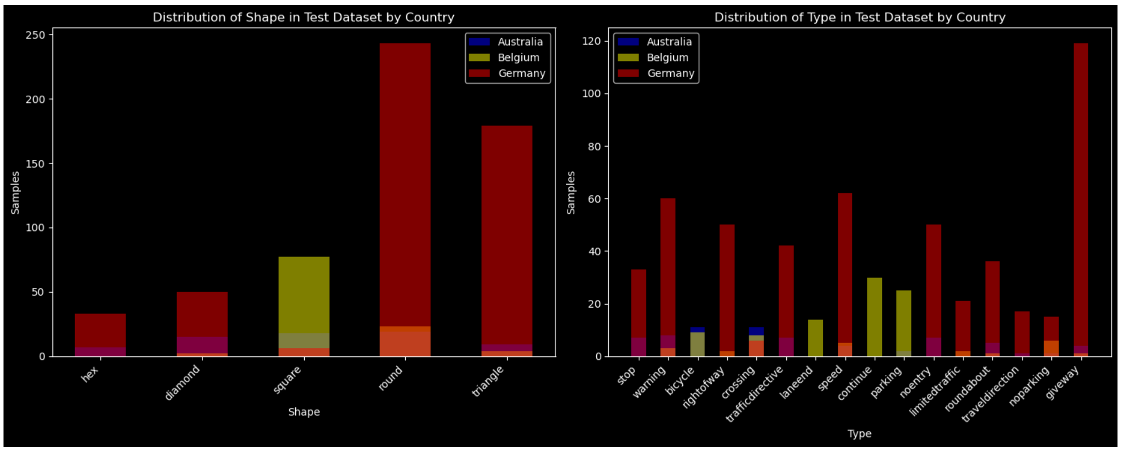


Figure 14: Independently Sourced Dataset Number of Samples per Country

During the data
collection process, we prioritised capturing images with varying angles, brightness levels, and sharpness/blurriness. We also as previously mentioned were using different cameras, which further simulated a real-world scenario. This approach helped us create a dataset that accounts for the diverse conditions in which road signs are encountered in real life. We also made sure to include both correct, poor-quality, and edge case examples of road signs to thoroughly test our model's ability to distinguish between different classes and handle challenging cases. Ensuring no overlap with our training / validation dataset was relatively easy. It just meant we had to be very careful to not find images that directly matched the training dataset, and find ones comparable to what a real-world test may look like.

To ensure compatibility with our trained model, we developed a custom pre-processing script that standardised the collected images. This script processes all .png and .ppm images that haven't already been transformed by resizing them to 28×28 pixels and converting them to Grey scale. By applying this pre-processing step, we guarantee that our test data is in the same format as the training data, enabling a fair evaluation of the model's performance. After pre-processing, we conducted an Exploratory Data Analysis (EDA) on the test dataset, similar to the analysis performed on the training data. This step allowed us to verify data consistency and identify any anomalies or irregularities. We ensured that the images were centred within the 28×28 frame and exhibited variance for sharpness, pixel intensity, entropy, and similarity in the object's appearance, reflecting real-world variations. See **Section 5.5 Independent Evaluation Ingestion & EDA in Appendix A: Jupyter Notebook**.

Independent Dataset EDA

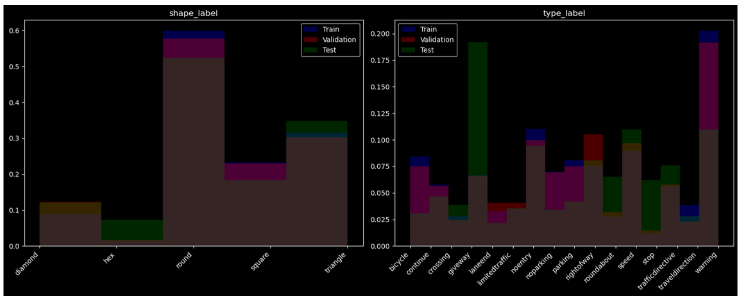


Figure 15: Showing number of samples across all 3 datasets for shape and type.

Figure 15 illustrates the normalised distributions of our training, validation, and test datasets, which are reasonably close, with extra test data for underrepresented classes. Blue represents training data, red indicates validation data, and green corresponds to test data. Other colours signify dataset overlaps: brown for all three datasets and yellow for validation and test only. Due to time constraints, we focused on easily accessible data, resulting in an abundance of certain classes like the hexagon stop or triangle giveway signs compared to less common signs like bicycle or no Parking. By following this meticulous data collection process, leveraging insights from domain experts, and applying appropriate pre-processing and EDA techniques, we created a high-quality and diverse independent evaluation dataset. This dataset will enable us to assess our trained model's generalisation capabilities and robustness when exposed to unseen examples, providing a reliable measure of its performance in real-world scenarios.

Independent Evaluation Performance

To simplify the evaluation of our model's performance, we created a script that runs all of our previously trained models and presents the results in a tabulated form at the end of our Notebook. The raw results of our independent evaluation are as follows:

Model	Shape Accuracy	Shape Precision	Shape Recall	Shape F1-score	Type Accuracy	Type Precision	Type Recall	Type F1-score
Multi-Layer Perceptron (MLP)	*0.63*	*0.64*	*0.63*	*0.60*	*0.40*	*0.54*	*0.40*	*0.44*
Convolutional Neural Network (CNN)	*0.84*	*0.86*	*0.84*	*0.83*	*0.74*	*0.78*	*0.74*	*0.75*
Improved CNN	*0.88*	*0.89*	*0.88*	*0.88*	*0.81*	*0.85*	*0.81*	*0.81*
Transfer Learning	*0.68*	*0.52*	*0.55*	*0.53*	*0.45*	*0.45*	*0.38*	*0.37*
Gradient Boosting	*0.70*	*0.69*	*0.70*	*0.67*	*0.50*	*0.66*	*0.50*	*0.53*
Random Forest	*0.70*	*0.70*	*0.70*	*0.66*	*0.54*	*0.66*	*0.54*	*0.55*
Decision Tree	*0.59*	*0.58*	*0.59*	*0.57*	*0.39*	*0.45*	*0.39*	*0.40*
Logistic Regression	*0.62*	*0.63*	*0.62*	*0.60*	*0.51*	*0.68*	*0.51*	*0.55*

In addition to the tabulated results, we also generated confusion matrices for each country of origin within our independent dataset. This allowed us to assess how well our models perform on images from different sources, particularly in distinguishing between the German, and Belgium road sign images and the ones we captured in Melbourne to see if the model would generalise across the world. We found that our models performed overwhelmingly well against the Belgium, and German images with an overall performance of 88% for shape, and 81%. Further analysis of the problems indicated that there was a large amount of misclassification from our Australian Dataset indicating we need some further training data to be able to adapt to the completely different road signs in Melbourne.

We also made a few plots where we can view our misclassified images to get insights to what images were being misclassified as seen in Figure 16.

While accuracy remains our primary metric for quickly evaluating our model's performance, we will also consider F1 score when determining the "best model" as this takes into account other metrics like precision, and recall. F1-Score provides a more comprehensive understanding of our model's capabilities.

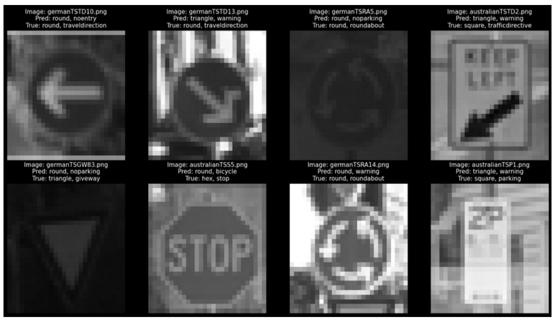


Figure 16: Independent Evaluation Misclassification

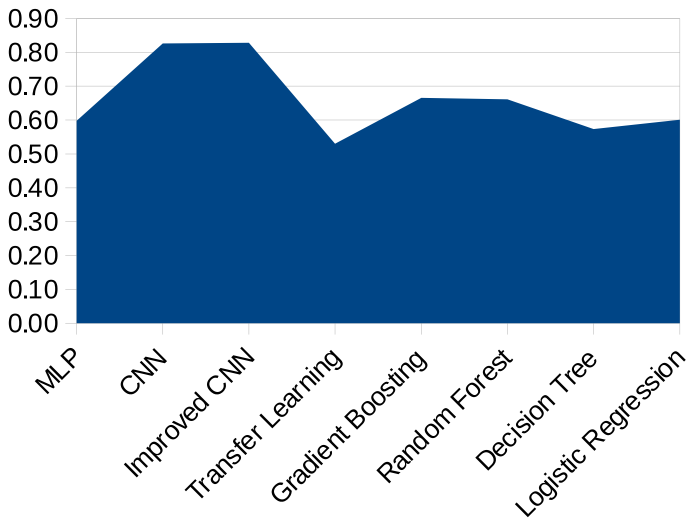


Figure 17: Independent Evaluation for Shape across all Models

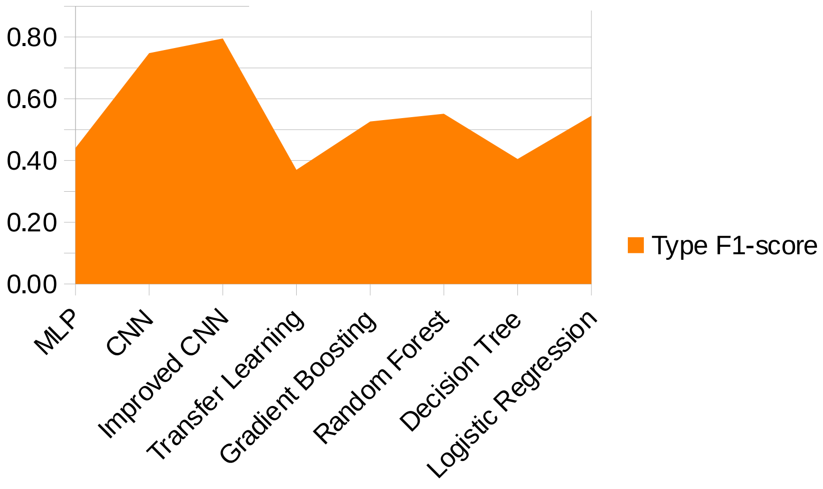


Figure 18: Independent Evaluation for Type across all Models

As seen in Figure 17 for shape classification independent evaluation our CNN models perform the best, while other tree-based models still perform impressively for their lower computational cost. Similarly, (shown in Figure 18) for type classification we find our CNN models performs optimally while MLPs, and all other models fall below 50%.

Ultimate Judgement

Now that we have completed our Independent Evaluation, we can provide our ultimate judgement on the optimal model choice. Based on the raw results, we can conclude that for the Shape classification task, the best model choice is the CNN architecture. For the Type classification task, CNNs were the only models that consistently performed above 50% accuracy, with a score of roughly 80%, making them the sole recommended choice. Considering the simplicity of having a single model for both tasks, the best overall choice is the optimised CNN architecture. It demonstrates exceptional performance in both Shape and Type classification, making it the most viable option for real-world deployment.

If computational resources are limited and a "Budget Model" is required, we recommend a simple logistic regression model. Although it may not achieve the highest accuracy, it still performs reasonably well and could be further optimised with additional tuning.

Our independent evaluation highlights the difficulty in creating a model that is generalisable to a country or scenario where it has no training data. Despite this challenge, the trained CNN model proves its viability in real-world practice, effectively classifying road signs based on their shape and type when applied to a diverse and independent dataset. This exceptional performance justifies the use of CNNs for this task and highlights their potential for real-world deployment, even in the presence of limited training data from certain regions.

Problems

During the model development process, we encountered several challenges. We experimented with various techniques such as Leaky ReLU activation, Stochastic Gradient Descent (SGD) optimizer, and batch size tuning to improve model performance. When exploring different batch sizes, we found that using batch sizes larger than 32 led to all 16GB of RAM being consumed, while smaller batch sizes resulted in poor performance. This highlighted the importance of finding the right balance between computational resources and model efficiency. Additionally, we faced issues with generators feeding file paths instead of actual pixel data, which required modifications to our data pipeline.

Another significant challenge was the presence of bad training data and edge cases. The small number of samples in certain classes limited the ability of Neural Networks to abstract and learn features effectively, hindering their generalization capability. The training data was also isolated, containing only a single country and a handful of shapes and types from the original dataset. Furthermore, the absence of color information, which is a distinctive feature of some road signs, made classification more challenging.

We also encountered difficulties with encoders losing the actual label for each class, making it harder to interpret the results when dealing with numeric labels (0-16) instead of descriptive names.

Moreover, when attempting to apply transfer learning techniques, we faced challenges due to the minimum size requirements of pre-trained models, which led to significant image distortion when resizing our smaller 28×28 images.

Conclusion

In conclusion, our traffic sign classification model development project has been an overall success. Through the evaluation of various architectures and techniques, we have identified CNNs as the optimal choice for both Shape and Type classification tasks. The trained CNN model demonstrates exceptional performance, even when applied to a diverse and independent dataset, proving its viability for real-world deployment.

Key findings and outcomes:

- ***CNNs consistently outperformed other architectures in both Shape and Type classification tasks***
- ***The trained CNN model achieves a classification accuracy of roughly 90% for Shape, and 80% for Type***
- ***The model's performance remains robust even when applied to a diverse and independent dataset***
- ***The project highlights the challenges of creating a generalisable model when training data is limited for certain regions or scenarios***

Potential next steps and improvements:

- ***Increase the size and diversity of the training dataset, including images from multiple countries and a wider range of shapes and types***
- ***Incorporate colour information in the training data to enhance the model's ability to distinguish between different road signs***
- ***Explore techniques for handling imbalanced datasets and improving the model's performance on underrepresented classes***
- ***Investigate the application of transfer learning to leverage pre-trained models and reduce the need for large amounts of labelled data***
- ***Optimise the model architecture and hyperparameters to further improve performance and efficiency***

Given the success of this project and the valuable insights gained, we are confident that future iterations and improvements will lead to an even more robust and generalisable traffic sign classification system.

Appendix

Appendix A: s3952320_s3900481.ipynb

Appendix B: Independent_dataset

Appendix C: trafficsigns_dataset

Appendix D: rawResults.xlsx

Appendix E: model/

Appendix F: model.png

References

1. Abdulla et al, W 2016, *Traffic Sign Recognition with TensorFlow*, Medium, accessed 20 April 2024, https://medium.com/@waleedka/traffic-sign-recognition-with-tensorflow-629dffc391a6.

2. Alavi, A, Sanjeevani, P & Chu, R 2024, *CANVAS RMIT for Machine Learning COSC2673 Particularly Labs, Lectures, and Tutorials. RMIT University* accessed 2024.

3. Deisenroth, MP, Faisal, AA & Ong, CS 2020, *Mathematics for Machine Learning*, Cambridge University Press, accessed 18 April 2024, https://mml-book.github.io/book/mml-book.pdf.

4. Google, *Images denoted with Belgium at the start*, Google Maps, accessed 5 May 2024, https://www.google.com/maps/place/Belgium/.

5. GTSDB September 2010, *Images denoted with German at the start*, GTSDB, accessed 18 April 2024, https://benchmark.ini.rub.de/gtsdb_dataset.html.

6. Kalahiki, C 2017, *CNN-for-BelgiumTS-and-MNIST*, GitHub, accessed 19 May 2024, https://github.com/ChrisKalahiki/CNN-for-BelgiumTS-and-MNIST/tree/master.

7. Karlijn Willems. 2017, *TensorFlow Tutorial For Beginners*, Hacker Noon, accessed 5 May 2024, https://hackernoon.com/tensorflow-tutorial-for-beginners-69358e73dee7.

8. Karlijn Willems. 2019, *TensorFlow Tutorial: A Guide to Building Deep Learning Models*, DataCamp, accessed 28 April 2024, https://www.datacamp.com/tutorial/tensorflow-tutorial.

9. Katiyar, P 2022, *Traffic Sign Recognition with TensorFlow*, Kaggle, accessed 19 May 2024, https://www.kaggle.com/code/parthkatiyar/traffic-sign-recognition-with-tensorflow.

10. Kelleher, JD, Mac Namee, B & D'Arcy, A 2015, *Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies*, MIT Press, accessed 7 May 2024, https://ebookcentral.proquest.com/lib/rmit/detail.action?docID=6246595&pq-origsite=primo.

11. Khan et al. 2022, *A Lightweight Convolutional Neural Network (CNN) Architecture for Traffic Sign Recognition in Urban Road Networks*, ResearchGate, accessed 20 April 2024, https://www.researchgate.net/publication/369959504_A_Lightweight_Convolutional_Neural_Network_CNN_Architecture_for_Traffic_Sign_Recognition_in_Urban_Road_Networks.

12. Khan et al. 2023, *SmartTutor: A Vision-Based Interactive Augmented Reality Educational Application for Kids Using Real-Time Object Detection and Deep Learning*, MDPI, accessed 3 May 2024, https://www.mdpi.com/2079-9292/12/8/1802.

13. Kherraki et al. 2022, *YOLOv5: An Efficient Object Detection Algorithm for Real-Time Applications*, IEEE Xplore, accessed 7 May 2024, https://ieeexplore.ieee.org/document/9806122.

14. Konar, M 2018, *BelgiumTS Dataset*, Kaggle, accessed 28 April 2024, https://www.kaggle.com/datasets/mahadevkonar/belgiumts-dataset/code.

15. Mathias, M, Timofte, R, Benenson, R & Van Gool, L 2013, 'Traffic Sign Recognition – How far are we from the solution?', *2013 International Joint Conference on Neural Networks (IJCNN)*, pp. 1-8, accessed 19 May 2024, https://btsd.ethz.ch/shareddata/publications/Mathias-IJCNN-2013.pdf?ref=hackernoon.com.

16. *Matplotlib Documentation*, accessed 2024, https://matplotlib.org/.

17. Mens Health Staff 2021, *Australian Road Signs for Dummies | Men's Health Magazine Australia*, Men's Health website, accessed 28 April 2024, https://menshealth.com.au/australian-road-signs-for-dummies/.

18. Michaels, P 2023, *iPhone 13 vs. iPhone 12: Which should you buy? | Tom's Guide*, Tom's Guide website, accessed 20 April 2024, https://www.tomsguide.com/face-off/iphone-13-vs-iphone-12.

19. Oisin & Vince, *Images denoted with Australian at the start*, RMIT University

20. Oisin Aeonn, 2024, *Topic: Lecture 1*, RMIT Canvas website, 2024, https://rmit.instructure.com/courses/125015/discussion_topics/2248913.

21. Romachub 2020, *Traffic Signs Recognition Test*, Kaggle, accessed 18 April 2024, https://www.kaggle.com/code/romachub/traffic-signs-recognition-test.

22. Sharma, T 2019, *Belgian Traffic Dataset*, Kaggle, accessed 6 May 2024, https://www.kaggle.com/code/tusharsharma118/belgian-traffic-dataset.

23. *Sklearn Documentation*, accessed 2024, https://scikit-learn.org/stable/.

24. Stallkamp et al. 2011, "The German Traffic Sign Recognition Benchmark: A multi-class classification competition", accessed 7 May 2024, https://ieeexplore.ieee.org/document/6033395.

25. Tang et al. 2018, *How to Classify MNIST Images with Convolutional Neural Network*, Diving into Genetics and Genomics, accessed 3 May 2024, https://divingintogeneticsandgenomics.com/post/how-to-classify-mnist-images-with-convolutional-neural-network/.

26. Temel et al. 2019, *Traffic Sign Detection and Recognition based on Hierarchical Deep CNN*, arXiv.org, accessed 12 May 2024, https://arxiv.org/pdf/1908.11262.

27. *TensorFlow Documentation*, accessed 20 April 2024, https://www.tensorflow.org/learn