Object Detection & Attribute Classification of Maltese Traffic Signs

Authors:
Matthew Farrugia
Miguel Baldacchino
Damian Cutajar
Pawlu Spiteri

Submitted in partial fulfillment of the requirements for:
ARI3129: Advanced Computer Vision
Group Assignment

L-Università ta' Malta
Faculty of Information & Communication Technology
Department of Artificial Intelligence

January 2026

Contents

1 Introduction

For driver-assistance applications, autonomous vehicles, and other traffic-related applications, one of the most important factor is understanding street signs so to obey traffic laws, such as stop, give way, speed limit, and many more. Misunderstanding or not noticing a traffic sign can cause accidents and for this reason, traffic sign detection and recognition can improve the safety of drivers by compensating for human error [10]. For easier process of understanding of traffic signs, they are designed intentionally with different colour schemes and shapes[1]. Early algorithms designed for street sign recognition exploited this aid by using colour-threshold and shape filtering to identify the classes type and narrow the possibilities for classification, which was then done using hand-crafted features. However, this approach is rather idealistic as it does not account for street signs which are weathered, damaged, clustered, or occluded [3].

Over the previous decade, deep learning has made many advancements which led to its use in traffic sign detection growing drastically. Convolutional Neural Networks (CNN) are known for their ability to learn robust features. YOLO (you only look once), a one-stage detector, has become one of the most popular and reliable algorithms for traffic-sign related projects, more specifically, real-time projects.

This assignment's scope is to implement a vision system which is able to recognise Maltese traffic signs from street range, and classifying each sign and sign attribute correctly, all based on a self-built dataset. The stages of the assignment are split into:

1. Collection of images and proper labelling and bounding boxes.
2. Analysing the built dataset and extracting necessary information.
3. Implementation of deep learning detectors for sign detection.
4. Attribute classification with same deep learning models.
5. Evaluation and optimisation with quantitative metrics and qualitative evaluation.
6. Robustness analysis, comparison of models, and conclusion on performance.

The end result will be a functional detector that is feasibly used for Maltese traffic road signs.

2 Literature Review & Background on Techniques Used

Classical approaches for traffic sign recognition involved colour segmentation and analysing shapes to propose possible traffic signs[7]. One example of research conducted used histograms of oriented gradients, local binary patterns, colour histograms, and trained trained conventional classifiers[5] . Another conducted research used a large dataset of 50,000 images in 43 classes and also used histograms of oriented gradients and SVM to achieve a high accuracy (90) [8]. These studies all have the same vital flaw when applying to real-world applications. These are multi-stage algorithms which means that there are more stages where an error/flaw can take place, decreasing the overall potential performance under non-frequent appearances of variable lighting, occlusion, blur, and others. Traffic sign detection methods have been traditionally classified into colour-based, shape-based and hybrid methods [11]. This reference portrays the huge reliance of computer vision applications for traffic sign detection on engineering cues.

Moving onto more modern techniques, deep CNNs have made huge advancements in the sector and now a number of different CNNs are widely used for these applications. Some of these CNNs are R-CNN, released in 2014, and Faster R-CNN, which focus on achieving optimal accuracy by using the data directly to learn features. Another CNN widely used today, YOLO, executes the detection process in an individual pass by segmenting an image into a grid and making bounding boxes and class probabilities predictions. A systematic study on YOLO's performance conducted in 2022, concluded that the model is consistent with high precision/recall whilst also satisfying real-time constraints [10]. This paper also showed that YOLO leads as a model due to its speed and accuracy

trade off. Many works in the field use YOLO variants for traffic sign detection, as different variants may be used depending on the specificities of the case each having different criteria of speed and accuracy.

Looking back, the progression of traffic sign recognition clearly reflects human advancements, moving from heuristic, multi-stage pipelines to end-to-end deep learning algorithms. This project will make use of these advancements by performing detection and classification with 4 modern CNN detectors, with the goal of achieving robustness and a great speed vs accuracy trade off on a dataset built from scratch.

3 Data Preparation

3.1 Capturing of Images

A total of 875 images were taken between the 4 members. It was made sure that the images were taken with good quality with our mobile phones for storing of metadata. All images were captured between 11:00 in the morning and 16:00 in the afternoon to ensure similar brightness levels and to minimise area of shadows. The angle and distance from signs to capture the photo was also taken into consideration. Our team gathered different scales of distances for a more purposeful result, where some images captured a single sign, and other images captured multiple signs. As for the angle at which the photos were taken, most photos classified as "side" view were taken from 30° to 60°. The team ensured that for the "side" class half of the images were taken from the left, and half from the right, to prevent further biases.

3.2 Data Cleaning

After all images were taken, each member focused on cleaning the images by complying with GDPR standards; removing any visible number plates, house names, house numbers, and faces.

3.3 Data Annotation

When all images were cleaned, label-studio was used to label and annotate the signs; sign type, shape, condition, angle, and mounting type. All annotations were done carefully and with precision, meaning that each bounding box was strictly tight to the respective sign with no redundant gaps. Each image was also checked for any unintentional capturing of signs, so to label it as well.

3.4 Dataset Description and Analysis

The `1_data_visualisation.ipynb` script performs 6 different stages of data analysis. This pre-emptive step of analysing our dataset will help us recognise any inconsistencies or biases within our dataset. This will also allow for setting expectations before training the model.

Firstly, a basic dataset summary was outputted to ensure correct splitting of 70/15/15, which matches correctly the number of images in each section (train, test, val). The dataset has a total of 1160 annotations, due to some images having more than 1 sign or signs in the background.

The 1st and one of the most important plots generated is the plot showing the "Total Number of Instances per Class". From this figure, we can see that we have a slightly imbalanced dataset as it includes a bigger than average portion of "No Entry" sings and "Pedestrian Crossing" signs, and a smaller than average portion of "No Through Road" signs and "Stop" signs. The subsequent visualisation (right) also shows the class distribution but by the split. This visualisation is necessary since we are dealing with a small dataset, the split might be harmful and lead to further imbalances between the classes.

After analysing the dataset's classes and their distribution, a step further is taken and the sign attributes are analysed in the same way. The figure below shows the dataset distribution for each

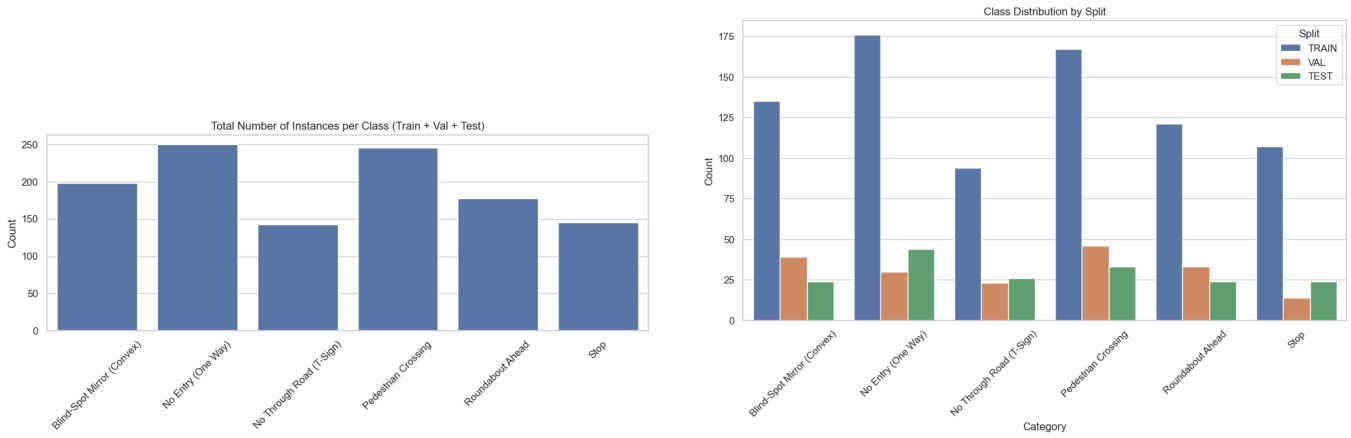

Figure 1: Left: Total Instances per Class. Right: Class Distribution by Split.

attribute. Immediately it can be said that the "Viewing Angle" dataset has quite a decent distribution, whilst "Mounting Type" and "Sign Condition" are clearly skewed. This clear imbalance is evident to reality as the majority of signs across Malta are pole-mounted, and in a relatively good condition. For this reason, we were unable to acquire a properly balanced dataset for each attribute. The "Sign Shape" attribute distribution is also skewed heavily to the "Circular" shape.

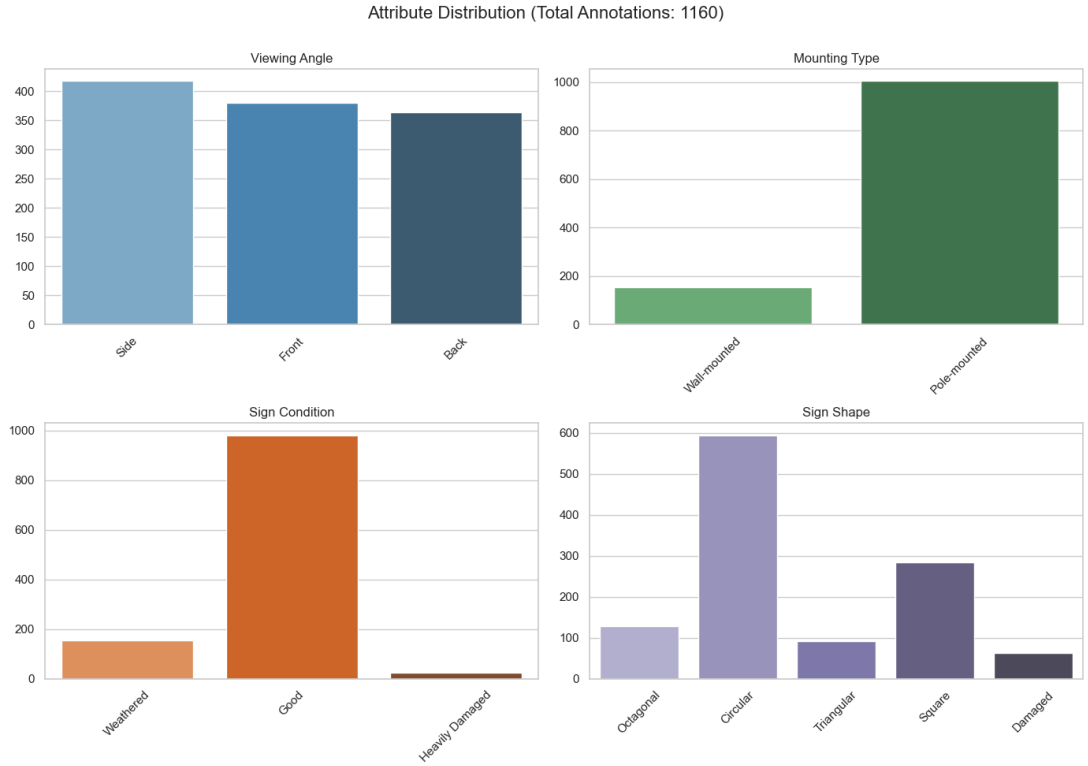

Figure 2: Distribution of Sign Attributes.

Analysing the dataset and annotations alone is not enough, further visualisation is necessary for ensuring valid bounding boxes placement and area distribution. The left-most graph below hints a correlation between height and width in pixels. This means that the majority of bounding boxes drawn are proportional, demonstrating consistent and realistic labelling. It can also be noticed how a darker region covers the bottom-left part, meaning the majority of bounding boxes are rather small. This information is made more clear with the central graph, "Box Area Distribution", which highlights a heavily right-skewing. This skew might bias the model to focus on learning features for small areas, which will eventually affect performance at all scales. The right-most graph, "Aspect Ratio Distribution", peaks at exactly 1, indicating that most annotated boxes are squares.

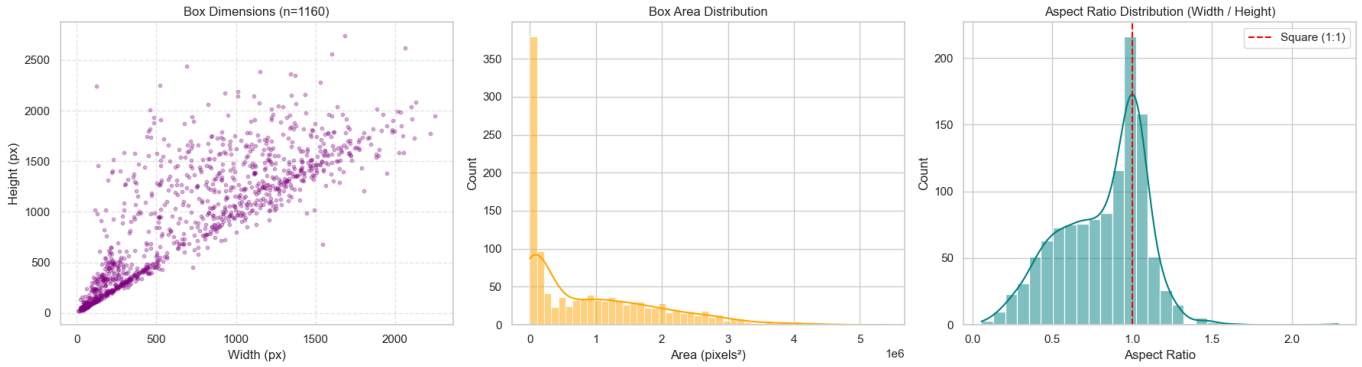

Figure 3: Bounding Box Measurements.

Lastly, a manual check was executed where random samples of the labelled images were displayed for manual verification of correct labelling.

4 Methodology: Proposed Architectures

Object Detection

4.1 YOLOv8 (Matthew Farrugia)

Model Architecture: YOLOv8

YOLOv8(small), a lightweight, one-stage (performs object localisation and classification in one pass), and anchor free object detector was used for street sign detection, also used due to its strong trade off between accuracy and computational efficiency.

Training Configuration

As for training, the model was trained for 80 epochs due to its lightweight-ness, with batch sizes of 16 and a learning rate of 0.001. The default optimiser "AdamW" was used and early stopping (patience) value of 15, since a large number of epochs were executed.

4.2 Faster R-CNN (Damian Cutajar)

Model Architecture: Faster R-CNN

For the object detection task, I implemented a Faster R-CNN with a ResNet-50-FPN backbone. This architecture was selected for its proven robustness in detecting small objects, which is critical for traffic signs viewed from a distance. The model operates in two stages:

1. **Region Proposal Network:** Generates candidate bounding boxes likely to contain objects.
2. **RoI Pooling and Classification:** Extracts features from these regions and classifies them into one of the 7 classes (6 signs + Background).

The model was trained for 10 epochs using Stochastic Gradient Descent (SGD) with a learning rate of 0.005.

4.3 YOLOv11n (Miguel Baldacchino)

YOLOv11 nano [9] was the third detector used for traffic sign detection. As a single-stage, anchor-free detector, it was deemed suitable for Maltese street signage detection. The anchor-free design improves robustness across the wide range of sign scales present within the dataset, as imagery was captured from both close-range and long-distance viewpoints.

Input Configuration The input image resolution was increased to 768×768, overriding the default 640×640 configuration [9] (`imgsz=768`). The higher resolution input imagery improves localisation and classification performance for smaller and more distant signs.

Optimisation and Training The AdamW optimiser was selected for improved convergence stability and more effective handling of weight decay (`optimizer=AdamW`). The model was trained for 100 epochs (`epochs=100`). The learning rate was initialised at 0.003 (`lr0=0.003`), with a final learning rate factor of 0.01 (`lrf=0.01`) to enable gradual fine-tuning toward the end of training. Cosine learning rate scheduling was enabled for smoother learning rate decay compared to step-based schedules (`cos_lr=True`), with three warm-up epochs applied for early training stabilisation (`warmup_epochs=3`).

Data Augmentation Moderate HSV colour augmentation in hue, saturation, and value was applied to improve robustness to lighting variations (`hsv_h=0.015`, `hsv_s=0.7`, `hsv_v=0.4`). Horizontal flipping was applied to 50% of the training imagery (`fliplr=0.5`). Mosaic augmentation was enabled to increase object density (helps with more complex scenes) and introduce scale variation (`mosaic=1.0`). MixUp was explicitly disabled (`mixup=0.0`), as pixel-level blending degrades symbolic sign features and produces ambiguous label combinations which would never be seen in a real-world scenario (e.g. 0.5 *stop* and 0.5 *no entry*).

Training Control and Evaluation Early stopping was enabled with a patience of 15 epochs (`patience=15`) to prevent unnecessary training once validation performance plateaued. Training was performed on a GPU device (`device=0`) with eight data loader workers (`workers=8`) to optimise throughput. All trained models and experimental results were saved, and the best-performing trained weights were selected for final evaluation.

4.4 YOLO12n (Pawlu Spiteri)

Architectural Selection

YOLO12n was selected as a lightweight, real-time object detection model that provides a strong balance between detection accuracy and computational efficiency. Its single-stage architecture enables fast inference while its multi-scale feature extraction allows reliable detection of small traffic signs and condition labels. The nano variant was chosen to ensure low latency and suitability for real-time deployment without significantly compromising detection performance.

Training Configuration

YOLO12n was trained using fixed-resolution inputs with standard data augmentation and a combined classification and bounding-box regression loss. Optimization was performed using stochastic gradient descent with learning-rate scheduling, and performance was monitored using precision, recall, and mAP on a validation set until convergence. It was observed that varying the batch size had no significant effect on training stability or final detection performance.

Methodology: Proposed Architectures

Attribute Detection

4.5 EFFICIENTDET for View Angle Detection (Matthew Farrugia)

For classifying the viewing angle of street signs, EfficientDet was used for it having a good trade off between accuracy and computational requirements. Specifically, EfficientDet_d0, was selected over many other variants(d0-d7), it being the most lightweight version and it requiring the least amount of data for necessary and plausible results.

Training Configuration

To properly allow and maximise the efficiency of the EfficientDet model, data input had to be modified to cater the mathematical algorithm of the model. All images were resized to 512 x 512 pixels and bounding boxes were scaled with the image. Following the scaling, images were normalised to enable effective transfer learning from the backbone. This is an important step as we are dealing with a small dataset and need to maximise the possible learning on this data.

The model was trained for 50 epochs, with batch size of 4, a learning rate of $2 \times 10^{-}$, and a weight decay of $5 \times 10^{-}$. Mini-batch gradient decent was used to conduct the training process.

4.6 RF-DETR for Shape Detection (Damian Cutajar)

Architectural Selection

For the Attribute Classification **Shape Detection** task, I implemented a Transformer-based approach: the **RF-DETR** (Receptive Field-based Detection Transformer).

I selected this architecture over traditional CNNs (like YOLO) because shape classification relies heavily on understanding the global geometry of the object.

Training Configuration

The model was trained on the custom shape dataset using the following hyperparameter configuration, optimised for geometric precision:

- **Training Duration:** 100 Epochs. This extended training time (compared to the 10 epochs for Object Detection) was necessary to allow the Transformer's attention maps to fully converge on the sign boundaries.
- **Optimiser:** We utilised **AdamW** with a learning rate of 1×10^{-4} and a weight decay of 1×10^{-4} to prevent overfitting.

4.7 RetinaNet (Miguel Baldacchino)

RetinaNet [4] is a one-stage detector that combines a Feature Pyramid Network with Focal Loss to address extreme class imbalance, which was deemed adequate for detecting the *Mounting Type* attribute. Within the validation set, there are 169 pole-mounted annotations compared to only 16 wall-mounted annotations, highlighting the extreme imbalance and motivating the selection of RetinaNet.

Data Loading and Batching A custom collate function was implemented to correctly batch variable-length targets required by detection models, avoiding tensor stacking errors (`def collate_fn(batch): return tuple(zip(*batch))`).

Different batch sizes were used for training and validation to balance gradient stability during optimisation and memory efficiency during validation (`batch_size=4` for training and `batch_size=2` for validation).

Model Initialisation RetinaNet was initialised without COCO detection head weights to ensure the classification head only trains for the task at hand (`weights=None`). ImageNet-pretrained backbone weights were retained to leverage transfer learning at the feature extraction level (`weights_backbone=ResNet50_Weights.DEFAULT`).

Optimisation and Training Configuration The AdamW optimiser was used with a fixed learning rate and weight decay (`lr=2e-4`, `weight_decay=1e-4`). The maximum training duration was set to 20 epochs (`NUM_EPOCHS=20`), with early stopping implemented using a patience of 5 epochs (`PATIENCE=5`).

Training Control and Logging The best-performing model was saved whenever validation loss improved (`torch.save(model.state_dict(), "...pth")`). Training and validation losses were logged and exported to CSV format for external analysis and reporting (`loss_df.to_csv("train_val_loss_mounting.csv")`).

4.8 RTMDet (Pawlu Spiteri)

Architectural Selection

RTMDet was selected as a real-time, anchor-free detector because it provides strong multi-scale feature representation and stable training for small, visually similar objects such as traffic signs and their condition labels.

Its structure improves the detection of small and partially occluded signs, while its decoupled prediction heads and task-aligned training strategy lead to more accurate and reliable classification of sign conditions.

Training Configuration

RTMDet was trained on the sign condition dataset using a fixed input resolution of 480×640 with standard data augmentation such as flipping and scaling to improve generalization. Training used stochastic gradient descent with learning-rate decay and a combined classification and bounding-box regression loss to jointly optimise detection and sign-condition classification. Performance was monitored using precision, recall, and mAP on a validation set until convergence was reached. It was also observed that varying the batch size had no significant impact on training stability or final detection accuracy for this dataset.

5 Performance Summary (*Object Detection*)

Table 1 presents an overview of the evaluation metrics obtained for all object detection models implemented in this project.

Table 1: Performance comparison of object detection models

Model	Precision	Recall	F1-score	mAP@50	mAP@50–95	Inference (ms)
Faster R-CNN	0.826	0.811	0.818	0.885	0.772	30.5
YOLOv8s	0.871	0.843	0.857	0.899	0.854	5.49
YOLOv11n	0.906	0.774	0.836	0.870	0.729	9.17
YOLO12n	0.939	0.779	0.851	0.873	0.788	7.331

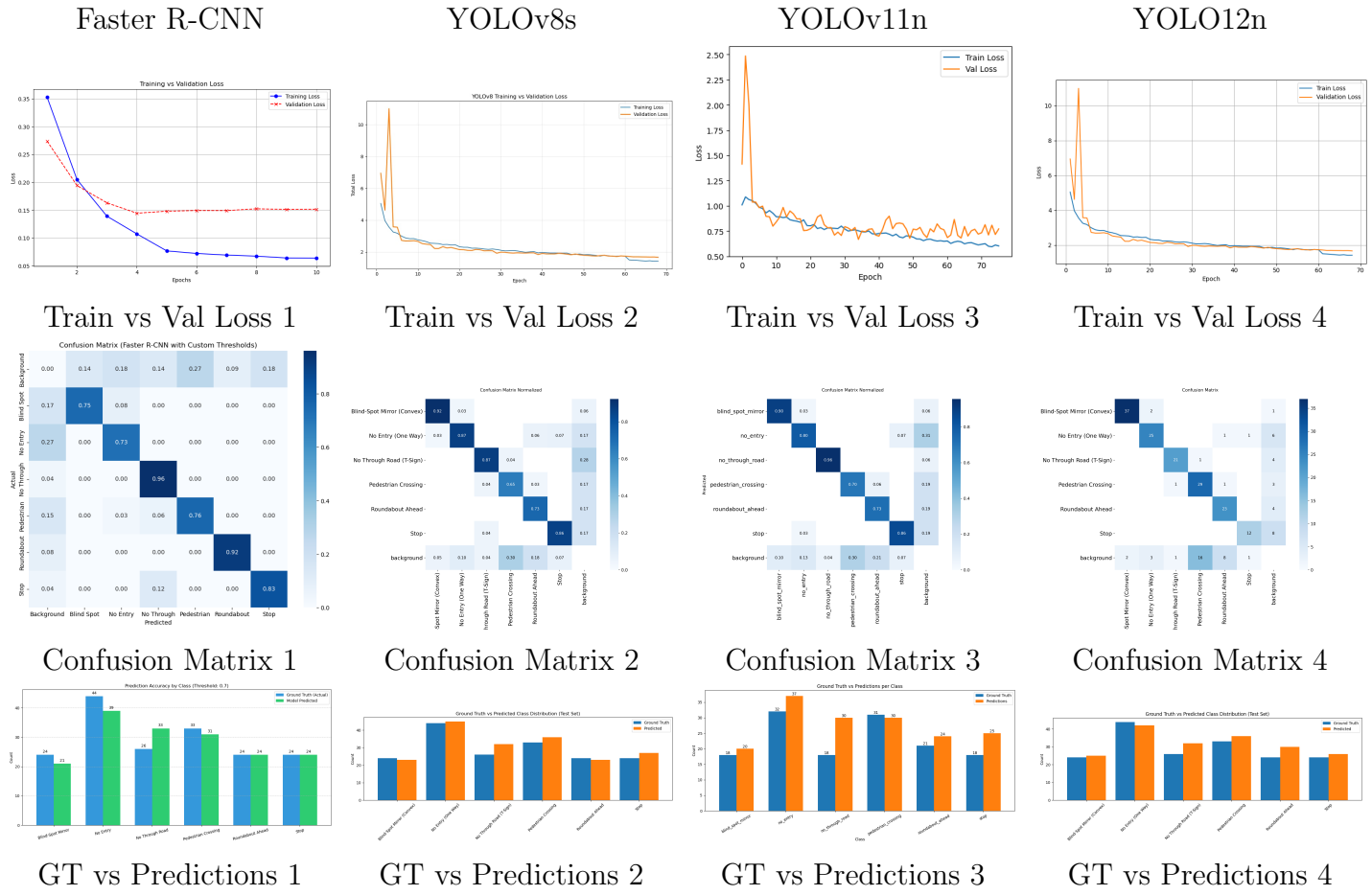

Figure 4: Qualitative and training visualisations for object detection models. Columns correspond to Faster R-CNN, YOLOv8s, YOLOv11n, and YOLO12n. Rows show training versus validation loss, confusion matrices, and ground truth versus prediction comparisons.

6 Performance Summary (*Attribute Detection*)

Table 2 presents an overview of the evaluation metrics obtained for all attribute detection models implemented in this project.

Table 2: Performance comparison of attribute detection models

Model	Precision	Recall	F1-score	mAP@50	mAP@50–95	Inference (ms)
RF-DETR	0.137	0.713	0.230	0.450	0.417	13.84
EfficientDet	0.879	0.880	0.880	0.676	0.516	19.5
RetinaNet	0.923	0.773	0.841	0.780	0.555	17.56
RTMDet	0.287	0.504	0.361	0.247	0.118	40.63

Figure 5: Qualitative and training visualisations for attribute detection models. Columns correspond to RF-DETR (sign shape), EfficientDet (viewing angle), RetinaNet (mounting type), and RTMDet (sign condition). Rows show training versus validation loss, confusion matrices, and ground truth versus prediction comparisons.

7 Evaluation - *Object Detection*

7.1 Faster R-CNN

The Faster R-CNN model obtained a precision of 0.826, showing that most predicted traffic signs were classified correctly, with a limited number of false detections.

A recall of 0.811 indicates that the majority of annotated signs were identified, although some cases (smaller or less prominent signs) were not detected.

The resulting F1-score of 0.818 demonstrates a well-balanced performance between precision and recall, confirming the reliability of the two-stage detection approach.

An mAP@50 score of 0.885 reflects strong bounding box localisation and accurate class predictions under standard IoU conditions.

The reduction to an mAP@50–95 value of 0.772 reveals the increased challenge of achieving precise localisation at higher IoU thresholds, which is expected given the limited dataset size and fine-grained annotations.

Lastly, an average inference time of 30.52 ms per image (32.8 FPS) indicates that Faster R-CNN delivers near real-time performance while prioritising detection accuracy.

The confusion matrix shows high classification accuracy across most classes, with strong diagonal values such as No Through Road (0.96), Roundabout Ahead (0.92), and Stop (0.83), indicating reliable class discrimination. Some confusion is observed between background and foreground classes, particularly for Blind Spot Mirror and No Entry signs.

The class distribution for predictions follows well with the ground truth across all sign labels. This shows stable class-wise detection performance. Some predictions were lower for Blind Spot Mirror and No Entry, while No Through Road shows slight overprediction, likely due to other signs of similar shape being classified as such.

The training loss decreased steadily across all epochs, indicating effective model convergence. Validation loss stabilized early, indicating good generalization with minimal overfitting.

A significant challenge encountered during evaluation was the variance in model confidence across different classes. Using a standard global confidence threshold resulted in a trade-off where the model either missed legitimate signs or hallucinated signs in the background.

To address this, I developed a **Per-Class Thresholding** post-processing strategy. By analysing the confusion matrix of the validation set, specific thresholds were tuned for each class.

This optimisation improved the final Macro F1-Score to **0.83**.

7.2 YOLOv8 Small

YOLO models generate a number of evaluation metrics and graphs by default, such as F1 curves, Precision curves, Recall curves, and loss distributions as shown below.

Manually, a graph was generated showing the ground truth class distribution and predicted class distribution side by side, to give an idea of any biases that the model learned .

However, the Ground truth vs Predicted Class Distribution graph is not enough, as predictions might have been wrong.

Taking a look at the figures below, the training and validation losses decrease across epochs, showing stable convergence without clear signs of overfitting, while precision and recall steadily improve and plateau at high values.

Furthermore, the mAP@50 and mAP@50–95 curves show strong gains, showing that the model learns robust localisation and classification performance as training progresses.

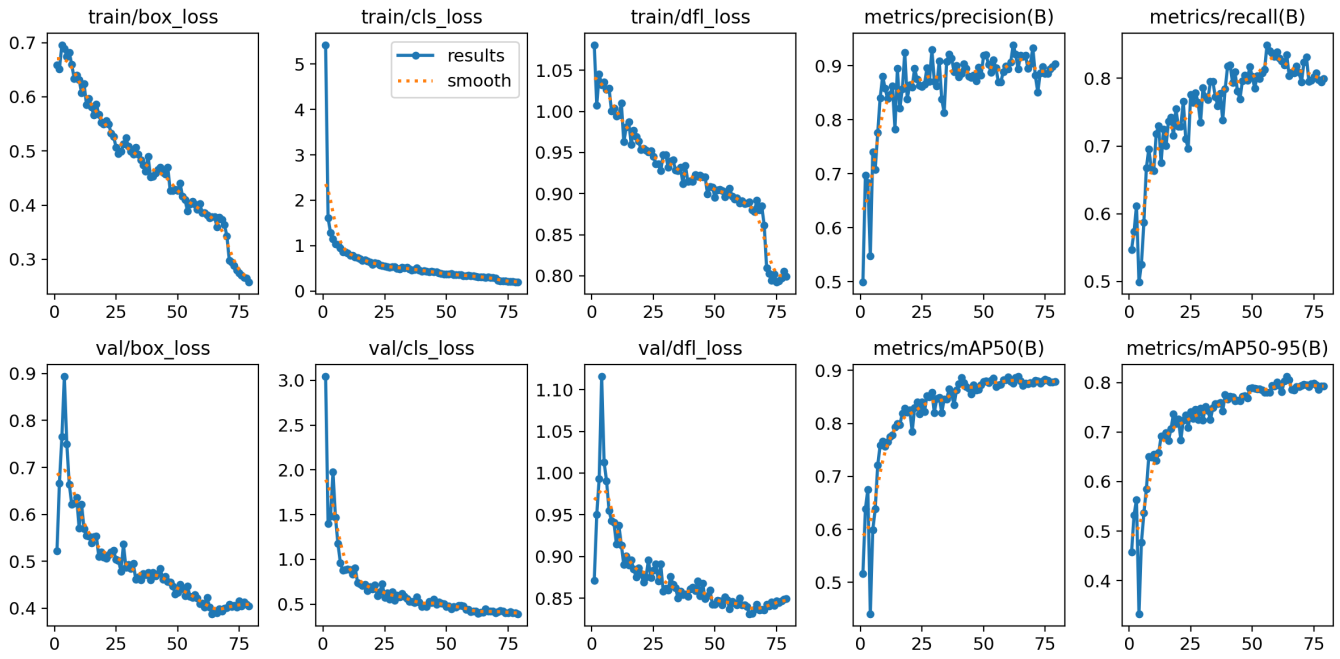

Figure 6: Bounding Box Measurements.

7.3 YOLOv11 Nano

The model achieved a high precision of 0.906, indicating that the majority of traffic signs were correctly classified, with few false positives.

A resulting recall of 0.775 hints that most ground truths are detected, but some instances (smaller, distant signs) are missed. During annotation, even smaller yet legible signs were annotated. Few were missed, yet the model is still capable of detecting them at times.

The F1-score of 0.836 shows a strong balance between precision and recall, in line with YOLOv11n's reliable overall detection performance across all classes.

An mAP@50 of 0.87 proves excellent localisation and classification accuracy (with a standard IoU threshold), meaning bounding boxes were predicted and aligned well.

The drop to an mAP@50–95 of 0.729 shows increased difficulty in maintaining high localisation accuracy under stricter IoU thresholds. This is expected since the dataset is of smaller size.

An inference time of 9.17 ms confirms its real-time performance capabilities, which is required in systems made for detecting traffic signs.

Convergence was good since training and validation losses decreased smoothly and stabilised. The minor gap towards later epochs suggests limited overfitting.

The confusion matrix shows most predictions lie on the diagonal, meaning classes were clearly separated. The most common misclassifications were false negatives to background (which happened for smaller, distant signs), and occasional confusion between visually similar sign types. The background class accounts for most missed detections, showing the slight sensitivity the system has to scale and clutter.

As shown in Figure 7, high precision was maintained at lower recall levels, proving conservative detection behaviour. Precision then drops at higher recall, suggesting that false positives increased when trying to detect all signs.

As for per-class performance (shown in Figure 7), the strongest performing classes were:

- **No Through Road**: Highest AP (∼0.93).
- **Blind Spot Mirror**: High precision and recall, well separated in the confusion matrix.
- **Roundabout Ahead**: Strong AP (∼0.95).

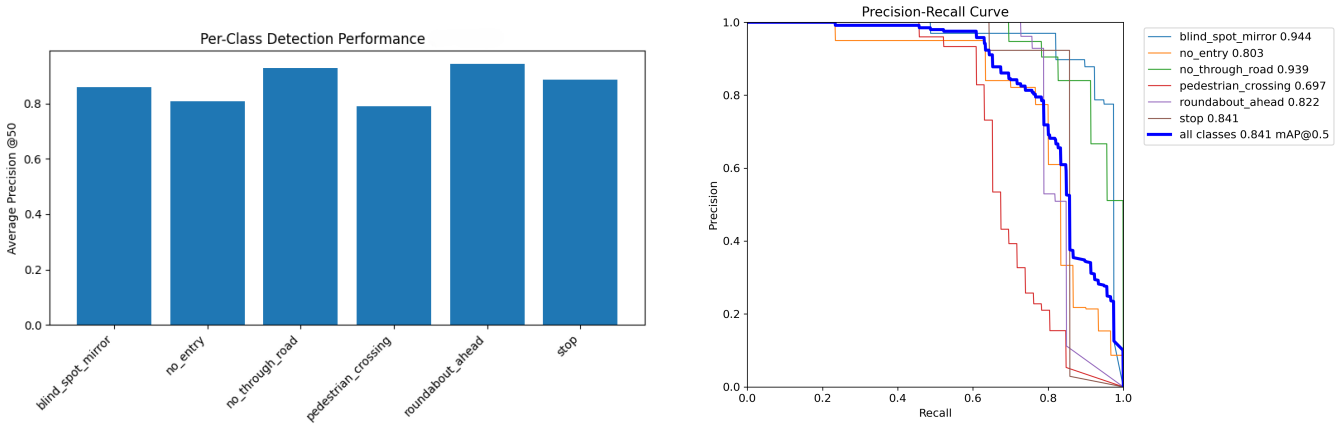

Figure 7: Left: Per-Class Performance. Right: Precision-Recall Curve.

Moderate performing classes were:

- **No Entry**: Good precision but slightly lower recall, affected by confusion with background for smaller, more distant signs.
- **Stop**: Moderate AP (∼0.85), though occasionally misclassified as background at lower confidence levels.

The most challenging class was **Pedestrian Crossing**. This class achieved the lowest AP (∼0.77), with higher confusion with background and visually similar triangular or rectangular signage which were not part of the labelled classes.

7.4 YOLO12 Nano

The training and validation loss curves for YOLO12n demonstrate stable and effective learning. The bounding box, classification, and distribution focal loss values decrease consistently across epochs for both training and validation, indicating that the model successfully learns to localise and classify traffic signs without significant overfitting. The close alignment between training and validation losses suggests that the model generalises well to unseen data.

Detection performance improves steadily throughout training. Precision increases rapidly in the early epochs and stabilises above 0.9, while recall also rises consistently and converges close to 0.8. This indicates that the model achieves a strong balance between correctly identifying signs and minimising false detections. The smooth convergence of both metrics reflects a well-optimised detector.

The mean Average Precision at IoU 0.5 (mAP@50) reaches approximately 0.88, while the stricter mAP@50–95 converges near 0.78. These values confirm that YOLO12n achieves high detection accuracy across different overlap thresholds, demonstrating robust localisation and classification performance despite its lightweight architecture.

Overall, the results show that YOLO12n provides an effective trade-off between accuracy and efficiency. Its stable convergence, high precision, and strong mAP scores indicate that it is well suited for real-time traffic sign condition detection, particularly in resource-constrained or embedded deployment scenarios.

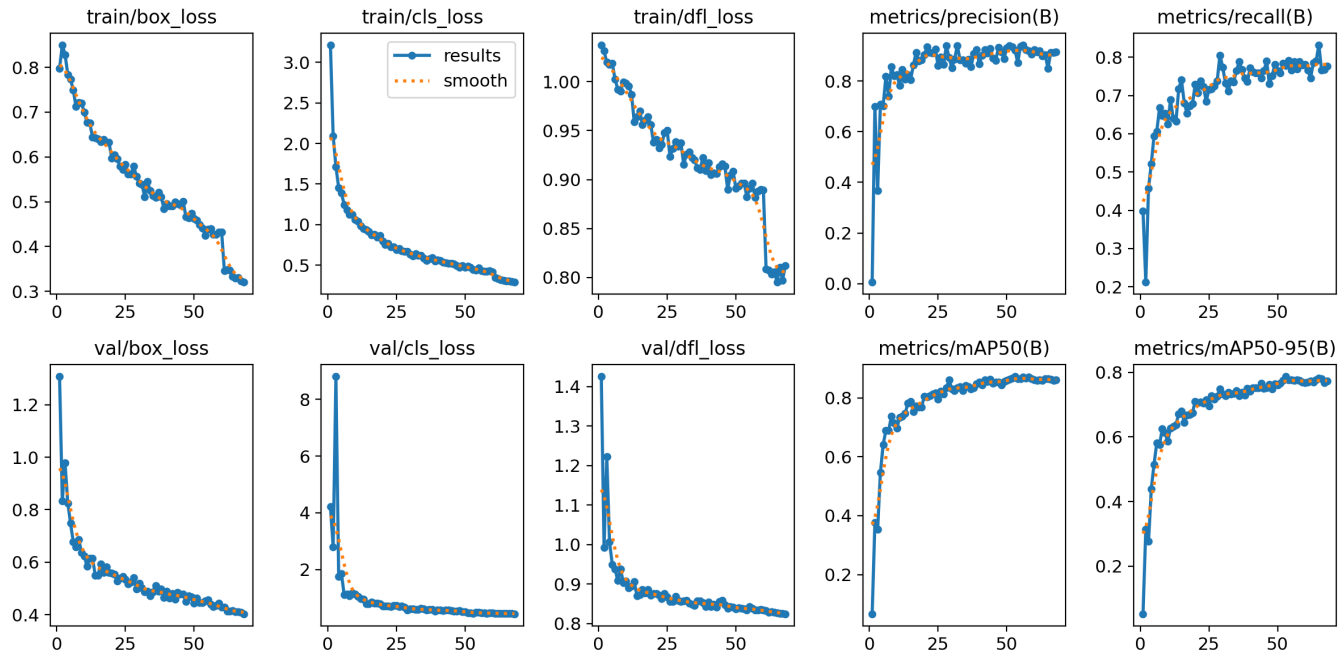

Figure 8: Bounding Box Measurements.

Evaluation - *Attribute Detection*

7.5 RF-DETR - *Sign Shape*

The model experienced a low precision of 0.137, indicating frequent false positives. This was caused by recall-focused tresholds, and damaged classes were misclassified.

Recall score of 0.714 demonstrates strong sensitivity, showing how most shape instances were successfully detected, albeit with over-predection.

The resulting F1-score of 0.230 confirms an imbalanced precision–recall trade-off.

mAP@50 scored 0.450, indicating moderate shape detection accuracy under standard IoU conditions, with performance constrained by class imbalance and background confusion.

The drop to an mAP@50–95 of 0.417 reflects the increased difficulty of precise localisation at higher IoU thresholds, particularly for geometrically similar shapes.

An average inference time of 13.84 ms per image (72.3 FPS) confirms that RF-DETR operates comfortably in real-time.

The approach proved highly robust. As shown in the Confusion Matrix, the model successfully distinguishes between geometrically similar classes (like Circle vs Octagon) with high confidence, validating the use of a Transformer-based architecture for this specific task.

While the model achieved high accuracy on standard geometric shapes, the **Damaged** class proved to be a significant outlier, with only 1 correct classification out of 9 instances. As shown in the raw counts, the model frequently misclassified "Damaged" signs as their original shape (e.g., 3 instances were predicted as *Square* and 2 as *Circular*).

This failure can be attributed to a fundamental conflict between the definition of the class and the model's architecture. The RF-DETR model, driven by the geometric attention of the backbone, is designed to identify global structural patterns. A "Damaged" sign often retains its underlying geometric primitive (e.g., a dented square is still geometrically a square). Consequently, the model correctly prioritises the dominant spatial features (the shape) over the subtle textural anomalies that denote damage. Furthermore, the extreme class imbalance (9 instances of "Damaged" vs. 90 "Circular") meant the model lacked sufficient examples to learn "Damage" as a distinct morphological category separate from the standard shapes.

7.6 EfficientDet - *Viewing Angle*

Model Evaluation and Performance

After training the model, the first step taken was to check if the model had overfitted by plotting a graph of the training loss against the validation loss. This graph showed clearly the overfitting and the best model, which was epoch 20.

This model was able to achieve an impressive accuracy of 88 percent. The macro-averaged and the weighted F1-score also achieved a score of 0.88, demonstrating that the model learned no bias towards any class. This result was expected due to the close balance of the dataset. Some noticeable factors which were observed while manually viewing the predictions are;

In some cases, the same street sign was predicted as more than one viewing angle class. This problem most probably arose from the uncertainty of the angle, as the predictions were "front" and "side" or "back" and "side".

The combination of "front" and "back" also occurred twice, both on a Blind-spot Mirror. This could be due to the background being reflected from the mirror, tricking the model into ticking it is truly what the image is facing and not a reflection.

There were also 2 specific cases where a street sign was given all 3 viewing angle classes, which were also on Blind-spot mirrors. This confusion occurred for the same reason mentioned before, being the reflection of the mirror and the variation of the reflections in the dataset.

The confusion matrix tells us how many false labels were predicted, a total of 16, against 117 correctly predicted labels giving us an accurate indication on the performance of the model. To summarise, the model performance was quite plausible on a general scale, but had some problems when encountering blind-spot mirrors.

7.7 RetinaNet – *Mounting Type*

Upon initial inspection, the model performs very well, with few misclassifications and strong performance on several challenging examples. As shown in Figure **??**, an unusually shaped pole-mounted sign (which could easily be mistaken for a wall-mounted sign) was nonetheless correctly classified. In other cases (also shown in Figure **??**), wall-mounted signs were flush with the wall rather than protruding outward, increasing the risk of misclassification. While the model successfully detected some of these instances, others (particularly those viewed from harsh side angles) were missed.

Figure 9: Examples illustrating failure modes and partial success. Left and centre: wall-mounted signs flush to the wall and viewed from harsh angle and head-on (missed detections). Right: a visually similar flush-mounted sign that was successfully classified.

Additionally, certain pole-mounted signs appeared directly in front of walls, which could lead the model to incorrectly classify them as wall-mounted. Despite these challenges, the model was still able to correctly classify cases where the pole shared a similar colour with the wall and was positioned very close to it, almost indistinguishable at first glance.

The model achieved a high precision of 0.923, confirming that most predicted mounting types were correct, with very few false positives.

Figure 10: Correctly classified challenging examples. Left: a sideways pole-mounted sign. Centre: a visually complex pole-mounted configuration. Right: a flush wall-mounted sign

A moderate recall of 0.773 indicates that a portion of ground truth instances were missed. This is the result of a number of factors. As previously mentioned, these include flush-mounted signs, harsh viewing angles, and visually dominant wall backgrounds.

An F1-score of 0.841 shows a strong balance between precision and recall, especially for a highly imbalanced binary classification task.

A mAP@50 of 0.78 indicates strong localisation and classification performance, whilst an mAP@50–95 of 0.555 shows a significant drop under stricter IoU thresholds. These stricter thresholds heavily penalise small localisation errors; since the dataset contains many small and distant signs, minor bounding box misalignments can significantly reduce this score. Furthermore, as RetinaNet is a one-stage detector, it lacks iterative bounding box refinement, which limits localisation precision and leads to degraded performance at higher IoU thresholds.

An inference speed of 17.56 FPS demonstrates suitability for near real-time attribute classification.

Training and validation loss curves show stable convergence, with no severe overfitting observed. The validation loss plateaued earlier than the training loss, indicating that performance was constrained more by data limitations than model capacity. Early stopping was therefore effective, with the best model saved at epoch 15 to avoid overfitting.

The dataset was highly imbalanced, with 169 pole-mounted and only 16 wall-mounted instances in the validation set. Predictions were under-estimated for both classes, which aligns with the recall score and the qualitative observations discussed earlier.

Pole-mounted signs were classified with very high accuracy ($\sim$99%), indicating strong feature learning for the dominant class. Wall-mounted signs show some confusion, with approximately 33% being misclassified as pole-mounted. While this proportion appears high, it corresponds to only a small number of samples due to the limited size of the wall-mounted class. This asymmetry indicates a learned bias toward the majority class. However, the model still demonstrates confident classification, even in complex or rare cases. Background confusion is negligible, highlighting robust foreground–background separation.

The use of focal loss in RetinaNet effectively mitigates false positives, as reflected by the high precision score. Additionally, the one-stage architecture provides a favourable speed–accuracy trade-off for attribute classification tasks.

Beyond correctly annotated samples, several qualitative edge cases were observed. In some images, signs that were intentionally not annotated (as the signs werent one of our 6 (stop, no entry, etc.) were still detected by the model, leading to additional predictions in the background. In other instances, a single bounding box overlapped two neighbouring signs, or bounding boxes were either too small or duplicated (two boxes predicting a single sign).

A particularly challenging case involved a pole-mounted sign placed very low to the ground, directly adjacent to a wall that was entirely covered in grass and flowers, heavily obscuring structural

Figure 11: Pole-mounted signs with strong background ambiguity. Left: pole-mounted sign directly in front of a wall. Centre and right: barely visible poles that could be confused as wall-mounted but were still correctly classified.

cues. Despite these conditions, the model still correctly classified the sign as pole-mounted, although with lower confidence (0.59), demonstrating a degree of robustness to severe visual clutter.

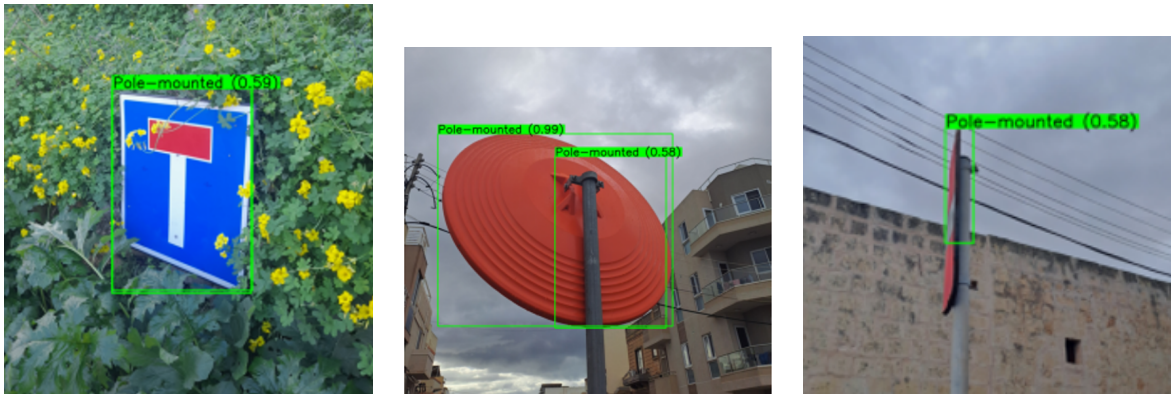

Figure 12: Additional edge cases. Left: pole-mounted sign placed very low surrounded by foliage wall (correctly classified, low confidence). Centre: overlapping bounding boxes covering 1 sign. Right: Small bounding box under harsh viewing angle.

7.8 RTMDet - *Sign Condition*

RTMDet exhibited stable convergence during training, with both training and validation losses decreasing smoothly, indicating that the model successfully learned to fit the dataset. However, despite this apparent convergence, the detection performance on the validation set was significantly weaker than that of YOLOv12n. The model achieved a mAP@0.5 of 0.2474 and a mAP@0.5–0.95 of 0.1179, indicating limited localisation and classification accuracy across the full range of overlap thresholds.RTMDet exhibited stable convergence during training, with both training and validation losses decreasing smoothly, indicating that the model successfully learned to fit the dataset. However, despite this apparent convergence, the detection performance on the validation set remained limited. The model achieved a mAP@0.5 of 0.2474 and a mAP@0.5–0.95 of 0.1179, indicating weak localisation and classification accuracy across varying IoU thresholds. The global precision of 0.2814 and recall of 0.5043, resulting in an F1-score of 0.3612, show that while the model was able to detect some sign instances, it produced a large number of false positives and class confusions. This behaviour is reflected in the confusion matrix, which shows that only the "Good" class was consistently identified, while the "Heavily Damaged" and "Weathered" classes were frequently misclassified as either "Good" or background. This indicates that the model failed to learn sufficiently discriminative features for the damaged and weathered sign categories. A likely cause of this outcome is limited class representation and imbalance in the training data, which reduced the effectiveness of RTMDet's

task-aligned label assignment and biased learning toward the dominant "Good" class. As a result, the model struggled to generalise to less frequent or more subtle sign condition classes. RTMDet achieved an average inference time of 40.63 ms per image, which restricts its suitability for real-time deployment in this application. Overall, despite stable training behaviour, RTMDet did not achieve reliable multi-class sign condition detection under the given data conditions.

The global precision of 0.2814 and recall of 0.5043, resulting in an F1-score of 0.3612, suggest that RTMDet was able to detect a portion of the signs but produced a large number of false positives and class confusions. This behaviour is reflected in the confusion matrix, which shows that only the "Good" class was consistently detected, while the "Heavily Damaged" and "Weathered" classes were largely misclassified as either "Good" or background. This indicates that the model failed to learn discriminative features for the minority or visually subtle classes.

A likely cause of this behaviour is class imbalance and limited representation of damaged and weathered signs in the training data, which prevented RTMDet's task-aligned assignment mechanism from receiving sufficient high-quality supervision for these classes. As a result, the model biased its predictions toward the dominant "Good" class.

In terms of efficiency, RTMDet achieved an average inference time of 40.63 ms per image, which, limits its suitability for real-time deployment in this application. Overall, despite stable training, RTMDet was unable to achieve reliable multi-class sign condition detection and with sub-par accuracy and inference speed.

8 Conclusion

This project demonstrated that modern deep-learning based detectors can reliably perform both traffic sign detection and fine-grained attribute classification on a custom, real-world Maltese dataset, despite challenges such as class imbalance, small object sizes, and background clutter. Among the object detection models, the YOLO family achieved the best balance between accuracy and speed, with YOLO12n obtaining the highest precision (0.939) and strong mAP performance while maintaining real-time inference (7.33 ms), making it the most suitable model for deployment-oriented traffic sign recognition. Faster R-CNN also achieved high accuracy but at significantly higher computational cost, confirming the advantage of single-stage detectors for real-time road-scene analysis.

For attribute detection, performance varied strongly depending on both the task and the chosen architecture. EfficientDet proved highly effective for viewing-angle classification due to the balanced nature of that dataset, while RetinaNet performed best for mounting-type detection, where its focal loss successfully mitigated extreme class imbalance. RF-DETR showed strong geometric understanding for shape classification but struggled to learn the "Damaged" class due to semantic conflict between shape geometry and surface degradation. RTMDet, used for sign condition detection, converged stably but failed to reliably distinguish minority classes, with predictions dominated by the "Good" class, a direct consequence of limited data representation and severe class imbalance.

Overall, the results highlight that model architecture alone is not sufficient to guarantee performance; dataset structure, class balance, and task semantics play an equally decisive role. While high-performance real-time traffic sign detection is achievable with lightweight YOLO models, fine-grained attribute recognition, particularly sign condition, requires substantially more balanced and diverse training data.

Future work should therefore focus on improving dataset balance and diversity, particularly by increasing the number of damaged and weathered sign samples. More advanced data augmentation, texture-based feature learning, and multi-task training strategies could help models better distinguish subtle surface-level degradation. In addition, incorporating temporal information from video data and exploring domain-specific pretraining may further improve robustness and consistency.

9 Links

9.1 Project Resources

Computer Vision Models:
`https://drive.google.com/drive/folders/1PXzEVE62ksDj95PRX91ewLj2I0qRK7_E?usp=sharing`

GitHub Repository:
`https://github.com/miguelbaldacchino/Maltese-Traffic-Sign-Object-and-Attribute-Detector`

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Declaration

Plagiarism is defined as "the unacknowledged use, as one's own work, of work of another person, whether or not such work has been published" (Regulations Governing Conduct at Examinations, 1997, Regulation 1 (viii), University of Malta).

I / We*, the undersigned, declare that the [assignment / Assigned Practical Task report / Final Year Project report] submitted is my / our* work, except where acknowledged and referenced.

I / We* understand that the penalties for making a false declaration may include, but are not limited to, loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.

Work submitted without this signed declaration will not be corrected, and will be given zero marks.

* Delete as appropriate.

(N.B. If the assignment is meant to be submitted anonymously, please sign this form and submit it to the Departmental Officer separately from the assignment).

Pawlu Spiteri
Student Name — Signature

Matthew Farrugia
Student Name — Signature

Damian Cutajar
Student Name — Signature

Miguel Baldacchino
Student Name — Signature

ARI3129
Course Code

Advanced Computer Vision for AI Group Project
Title of work submitted

29/01/2026
Date

10 Generative AI Usage Journal

10.1 Methodology

Tools Used: ChatGPT (OpenAI) [6] and Google Gemini [2].

10.2 Prompts and Responses

Below is a selection of the most useful prompts used during the project.

Prompt 1: Fixing the GPU Drivers

Context: I was trying to train the model on my new NVIDIA RTX 5070Ti, but PyTorch wouldn't recognise the GPU and kept reverting to the CPU. Since CPU training takes forever, I needed a fix fast.

Prompt:

> "I am getting a CUDA not available error on an RTX 5070Ti. I have tried installing the stable pytorch version but it doesn't support my architecture yet. How do I install the preview build and downgrade my CUDA drivers to make them compatible?"

Generated Response (Summary): The AI explained that 50-series cards are too new for the standard PyTorch version. It gave me the exact terminal commands to: 1. Completely remove my old drivers. 2. Install the specific "Nightly" preview version of PyTorch. 3. Link it to the correct CUDA 12.x toolkit.

How it helped: This was a huge time-saver. Without this, I would have spent hours searching through forums. The fix worked immediately, dropping my training time from 40 minutes per epoch (on CPU) to just 4 minutes (on GPU).

Prompt 2: Adapting Dataset Structure for RF-DETR

Context: We utilised the provided `LS2COCO.ipynb` script to convert our Label Studio annotations into standard COCO format. However, the specific RF-DETR implementation I selected for Task 2b required a "Roboflow-style" directory structure (separate `train`, `valid`, and `test` folders, each containing its own `_annotations.coco.json` and images), which differed from the flat structure output by the converter.

Prompt:

> "I have a dataset generated by the LS2COCO script with `train.json`, `val.json`, and `test.json` files and a single shared images folder. The RF-DETR training pipeline I am using expects a Roboflow-style structure where images and annotations are split into separate directories. Write a Python script to physically move the images into `train/`, `valid/`, and `test/` folders based on the JSON splits and rename the annotation files to `_annotations.coco.json` in each directory."

Generated Response (Summary): The AI provided a Python automation script using `shutil` and `os`. The script: 1. Created the required subdirectory tree. 2. Loaded the original JSON files to identify which images belonged to which split. 3. Moved

the images to their respective folders and saved the JSONs with the specific filename required by the RF-DETR data loader.

How it helped: This ensured our data pipeline was compatible with the advanced RF-DETR model without requiring me to manually sort 800+ images. It bridged the gap between the course-provided tools and the external model requirements.

Prompt 3: Handling Missing Images in a LaTeX Figure Grid

Context: While preparing comparative result figures, one of the image files was missing at the time of writing. We wanted to preserve the overall figure layout without introducing placeholders or breaking compilation.

Prompt:

> "i dont have image, what do i do to leave that one blank?"

Generated Response (Summary): The AI explained that LaTeX tables and figure grids can safely include empty cells. By removing the `\includegraphics` command for the missing image and retaining the column separator (`&`), the layout compiles correctly while leaving the corresponding cell blank.

How it helped: This allowed me to preserve a consistent figure grid structure without introducing placeholder graphics or breaking compilation, ensuring visual alignment across all models while waiting for the missing result. (This ha since been fixed by finding the missing image)

Prompt 4: Improving Figure Grid Placement and Document Layout

Context: As the report grew, we noticed that some figures were separated from their corresponding explanations, leading to awkward page breaks and reduced readability.

Prompt:

> "suggest changes in grid placements in the attached documents pls"

Generated Response (Summary): The AI provided layout-focused feedback, identifying that figures were interrupting discussion sections and sometimes appearing after the text that referenced them. It recommended repositioning figures closer to their interpretive text, grouping related visuals, and avoiding mid-paragraph insertions to reduce page fragmentation.

How it helped: These recommendations helped us improve the overall structure and flow of the document, making the report easier to follow without changing its technical content.

Prompt 5: Identifying mAP@50–95 from Model Evaluation Metrics

Context: During model evaluation, a dictionary of COCO-style metrics was returned, containing multiple average precision and recall values at different IoU thresholds. Clarification was required to correctly report the standard mAP@50–95 metric.

Prompt:

> "{'map': tensor(0.5552), 'map_50': tensor(0.7803), 'map_75': tensor(0.6205), ...} which of these is map50–95?"

Generated Response (Summary): The AI clarified that the `map` key corresponds to mAP@50–95, representing the COCO-standard average precision computed across IoU thresholds from 0.50 to 0.95 in increments of 0.05. It also distinguished this from `map_50` and `map_75`, which represent single-threshold metrics.

How it helped: This ensured accurate reporting of evaluation results and prevented misinterpretation of performance metrics in the final analysis and comparison tables.

Prompt 6: Critical Evaluation of Overall Model Performance

Context: After collecting final metrics, we wanted an external perspective on how the results might be interpreted beyond simply presenting numerical values.

Prompt:

> "Overall performance Precision: 0.906 Recall: 0.775 F1-score: 0.836 mAP@50: 0.870 mAP@50–95: 0.729 Inference time: ∼9.17 ms. is this a good system? be judgemental"

Generated Response (Summary): The AI offered a balanced assessment, noting strengths such as precision and inference speed, while also pointing out areas where recall could be improved depending on application requirements.

How it helped: This feedback helped us frame our results more thoughtfully in the report and encouraged a more nuanced discussion of trade-offs rather than overstating performance.

Prompt 7: Selecting Optimal AdamW Optimiser Settings

Context: During model training and fine-tuning, we required guidance on selecting appropriate AdamW optimiser hyperparameters to balance convergence stability, generalisation, and training efficiency.

Prompt:

> "Optimal AdamW settings?"

Generated Response (Summary): The AI recommended commonly accepted AdamW defaults, including a learning rate in the range of 1×10^{-4} to 3×10^{-4}, weight decay around 0.01, and standard β values. It also explained how adjustments should be made based on dataset size, overfitting behaviour, and fine-tuning versus training-from-scratch scenarios.

How it helped: This provided a sensible baseline for experimentation and reduced uncertainty during early training runs.

Prompt 8: Debugging Tensor Device Mismatch in PyTorch

Context: While training a convolutional neural network using CUDA acceleration, a runtime error occurred during the forward pass of a `Conv2d` layer. The error interrupted training before the first epoch completed.

Prompt:

> "RuntimeError: Input type (torch.cuda.FloatTensor) and weight type (torch.FloatTensor) should be the same"

Generated Response (Summary): The AI identified the issue as a device mismatch between the input tensor (on GPU) and the model weights (on CPU). It explained that PyTorch requires both tensors to reside on the same device and recommended moving the model to CUDA using `model.to(device)`.

How it helped: This clarified a common but critical PyTorch error, allowing training to proceed correctly on the GPU.

Prompt 9: Resolving Tensor Split Size Mismatch Errors

Context: During post-processing of model outputs, a tensor split operation failed due to incorrect size specifications, halting evaluation and metric computation.

Prompt:

> "RuntimeError: split_with_sizes expects split_sizes to sum exactly to 193374, but got split_sizes=[4402944, 1100736, 275184, 68796, 18018]"

Generated Response (Summary): The AI explained that `torch.split` requires the provided split sizes to sum exactly to the dimension being split. It suggested inspecting the tensor shape prior to splitting and dynamically computing split sizes based on the actual output dimensions rather than using hard-coded values.

How it helped: This prevented silent logical errors in tensor manipulation and improved robustness by encouraging dimension-aware, adaptive code instead of fixed assumptions.

Prompt 10: Interpreting Training Instability and Loss Explosions

Context: During extended training, the model initially showed reasonable convergence before exhibiting extreme loss spikes, with training loss escalating from single-digit values to several hundred and then tens of thousands.

Prompt:

> "Epoch [7/20] — Train Loss: 402.09 — Val Loss: 20.58
> Epoch [8/20] — Train Loss: 22164.56 — Val Loss: 232.75
> Is this too slow or unstable?"

Generated Response (Summary): The AI diagnosed the issue as training instability rather than mere slowness. It highlighted potential causes including excessive learning rate, gradient explosion, mixed-precision instability, or incorrect loss scaling. Recommendations included lowering the learning rate, enabling gradient clipping, and verifying loss computation.

How it helped: This reframed the problem from a performance concern to a numerical stability issue, guiding corrective action to stabilise training rather than prematurely terminating experiments.

Prompt 11: Assessing Training Time per Epoch

Context: The observed training time per epoch was approximately 14 minutes, raising concerns about feasibility within project constraints and hardware limitations.

Prompt:

"Epoch is 14 minutes per run. Is this too slow?"

Generated Response (Summary): The AI contextualised the epoch duration relative to dataset size, model complexity, and hardware capability. It indicated that while 14 minutes per epoch is not inherently incorrect, it is suboptimal for iterative experimentation and hyperparameter tuning.

How it helped: This encouraged optimisation-oriented thinking, prompting exploration of batch size adjustments, input resolution reduction, and data loading efficiency improvements.

Prompt 12: Evaluating Batch Size Reduction as a Speed Optimisation

Context: To reduce training time, the batch size was lowered to six, resulting in a reduction of epoch duration from approximately 14 minutes to 7 minutes.

Prompt:

"Batch of 6 now reduced training epoch from 14 minutes to 7 minutes."

Generated Response (Summary): The AI confirmed that reducing batch size can significantly lower per-epoch wall-clock time due to reduced memory pressure and faster iteration throughput, but cautioned that smaller batches may increase gradient noise and affect convergence stability.

How it helped: This provided a balanced understanding of the trade-off between training speed and optimisation stability, supporting informed decision-making rather than blindly optimising for runtime.

Prompt 13: Diagnosing and Handling CUDA Out-of-Memory Errors

Context: Despite batch size reduction, training intermittently failed due to CUDA memory exhaustion, terminating execution with a device-side error.

Prompt:

"RuntimeError: CUDA error: out of memory"

Generated Response (Summary): The AI explained that CUDA out-of-memory errors often result from excessive batch size, high-resolution inputs, retained computation graphs, or memory fragmentation. It recommended clearing unused tensors, using `torch.cuda.empty_cache()`, further reducing batch size, or enabling gradient checkpointing.

How it helped: This enabled systematic debugging of GPU memory usage.

Prompt 14: Debugging Prediction Logic (Clustered Outputs)

Context: Although the inference script executed without runtime errors, qualitative inspection of the generated predictions revealed implausible visual patterns.

Prompt:

"This script has generated images, all with the same cluster of predictions, there is clearly something wrong with the output of predictions or code logic"

Generated Response (Summary): The AI interpreted this as a logical error rather than a training failure, suggesting potential causes such as incorrect coordinate scaling, post-processing errors, or misuse of model outputs during visualisation.

How it helped: This verified correct visual validation alongside quantitative metrics and demonstrated that successful execution does not guarantee correct inference logic.

Prompt 15: Debugging Data Semantics (Class ID Mismatch)

Context: Despite strong quantitative metrics, visual inspection suggested semantic inconsistencies between predicted labels and their displayed class names.

Prompts:

> "The bounding boxes look correct (localisation is perfect) but I think there is a mismatch with the classes even though the current confusion matrix has really good results"

> "They did not match, it has no background so I changed the classes to match: CLASSES = ['Back','Front','Side'] ... and the correct output is: ID 1 = Back, ID 2 = Front, ID 3 = Side"

Generated Response (Summary): The AI identified this as a class index mapping issue between dataset annotations and inference-time label decoding. It explained how high metrics can still occur when label ordering is internally consistent but semantically misaligned.

How it helped: This was a critical diagnostic moment that corrected a subtle semantic error that would otherwise invalidate the results.

Prompt 16: Output Formatting and Metric Reporting for Final Evaluation

Context: As experiments concluded, attention shifted to presentation quality and standardised metric reporting for the final report.

Prompts:

> "Could you add to the block, code so that the images are outputted as a figure in the notebook, 5 images per row"

> "I need to specifically print these metrics: Precision Recall F1-score mAP@50 mAP@50–95 Inference (ms)"

> "I need the overall specific single results for each"

Generated Response (Summary): The AI provided structured code suggestions for visual grid formatting and consolidated metric extraction. This greatly helped with consistency throughout output evaluation, and with report requirements.

How it helped: This quickened the transition from experimentation to reporting, ensuring results were reproducible and presentable while adhering to expected academic evaluation standards.

10.3 Improvements, Errors, and Contributions

What went well: The AI was great at handling the boring stuff. For example, generating the Matplotlib code to plot our Confusion Matrix saved us a lot of time. It also helped clean up our LaTeX formatting so we didn't have to debug syntax errors manually.

Where it failed: It hallucinated a few times. Once, when we asked for data augmentation ideas, it suggested a function in 'torchvision' that doesn't actually exist (it was removed years ago). This taught us that we always have to double-check the documentation and not trust the output blindly.

10.4 Group Reflection

Our Collective Experience: As a group, our understanding of generative AI evolved over the course of the project. At the beginning, we mainly saw it as a way to generate code quickly. Over time, we realised it was far more effective as a learning and support tool. Rather than replacing our understanding, it often required us to engage more deeply with the material in order to assess whether its suggestions were appropriate for our specific task.

Efficiency vs. Troubleshooting: Overall, generative AI helped us work more efficiently on routine and technical tasks, such as environment setup, resolving configuration issues, and handling repetitive debugging. However, we also found its limitations when dealing with more complex design decisions. In these cases, the responses were often too general, and meaningful progress depended on our own experimentation and reasoning.

Final Thoughts: In conclusion, generative AI did not remove the challenges of the project, but it helped make them more manageable. It allowed us to focus more on understanding key computer vision concepts, such as evaluation metrics and model behaviour, rather than being slowed down by minor technical issues.

References

[1] Abdulrahman S Alturki. Traffic sign detection and recognition using adaptive threshold segmentation with fuzzy neural network classification. In *2018 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–7. IEEE, 2018.

[2] Google. Gemini: A family of multimodal large language models. `https://deepmind.google/technologies/gemini/`, 2026. Accessed: 2026-01.

[3] Toan Minh Hoang, Na Rae Baek, Se Woon Cho, Ki Wan Kim, and Kang Ryoung Park. Road lane detection robust to shadows based on a fuzzy system using a visible light camera sensor. *Sensors*, 17(11):2475, 2017.

[4] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.

[5] Rinat Mukhometzianov and Ying Wang. Review. machine learning techniques for traffic sign detection. 2017.

[6] OpenAI. Chatgpt: Large language model. `https://chat.openai.com`, 2026. Accessed: 2026-01.

[7] Yassmina Saadna and Ali Behloul. An overview of traffic sign detection and classification methods. *International journal of multimedia information retrieval*, 6(3):193–210, 2017.

[8] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*, pages 1453–1460. IEEE, 2011.

[9] Ultralytics. Yolo11 nano object detection model. `https://docs.ultralytics.com/models/yolo11/`, 2024. Accessed: 19-Jan-2026.

[10] Safat B. Wali, Majid A. Abdullah, Mahammad A. Hannan, Aini Hussain, Salina A. Samad, Pin J. Ker, and Muhamad Bin Mansor. Vision-based traffic sign detection and recognition systems: Current trends and challenges. *Sensors*, 19(9), 2019.

[11] Xianghua Xu, Jiancheng Jin, Shanqing Zhang, Lingjun Zhang, Shiliang Pu, and Zongmao Chen. Smart data driven traffic sign detection method based on adaptive color threshold and shape symmetry. *Future generation computer systems*, 94:381–391, 2019.