Automatic Detection & Attribute Classification of Maltese Traffic Signs

ARI3129 — Advanced Computer Vision for AI

GitHub Link: https://github.com/liamkazzopardi/Advanced-CV-ARI3129

Daniel Simon Galea
Liam Azzopardi
Liam Jake Vella
Matthew Privitera

January 2026

Contents

Chapter 1

Introduction

1.1 Project Overview

This project focuses on the automatic detection and attribute classification of Maltese traffic signs using modern computer vision techniques. The system aims to detect traffic signs in real-world street images and classify one selected attribute per team member, supporting intelligent traffic sign monitoring and maintenance.

1.2 Scope and Objectives

The project addresses the detection of six Maltese traffic sign categories: Stop, No Entry, Pedestrian Crossing, Roundabout Ahead, No Through Road, and Blind-Spot Mirrors. Each detected sign is further analysed using one of the following attributes: viewing angle, mounting type, sign condition or sign shape.

1.3 Team Contributions

The dataset was collected and annotated collaboratively. Model implementation and attribute classification were carried out individually as follows:

- Matthew Privitera: RF-DETR and YOLOv12 — Viewing Angle
- Liam Jake Vella: YOLOv12 and Faster R-CNN — Sign Shape
- Daniel Simon Galea: YOLOv11 and FCOS — Mounting Type
- Liam Azzopardi: YOLOv8 and RetinaNet — Sign Condition

Chapter 2

Background on the Techniques Used

2.1 YOLO Family (v8, v11, v12)

2.1.1 YOLOv8

Introduced by Ultralytics in 2023 [1], YOLOv8 is a state-of-the-art object detection model in the YOLO family. YOLOv8 introduced several significant changes within it's architecture from its predecessor models including a *C2f* module replacing the *C3* module in YOLOv5 and an anchor-free detection head, and a decoupled head architecture [2] that separates classification and localisation tasks. As with the all other modern YOLO models, YOLOv8 offers five scaled variants, YOLOv8n (nano), YOLOv8s (small), YOLOv8m (medium), YOLOv8l (large) and YOLOv8x (extra-large) [1], allowing developers to balance detection accuracy against computational constraints and real-time performance requirements.

2.1.2 YOLOv11

Released in late 2024 by Ultralytics [3], YOLOv11 advances the YOLO family by refining convolution-centric design to achieve stronger accuracy–efficiency trade-offs while preserving real-time performance. Unlike earlier versions that focused primarily on scaling depth and width, YOLOv11 emphasises architectural optimisation through improved feature extraction and aggregation. The model introduces three key innovations: first, the C3k2 module, a lightweight Cross-Stage Partial block with smaller kernels that enhances gradient flow while reducing computational cost. Second, an enhanced Spatial Pyramid Pooling-Fast (SPPF) layer to better capture multi-scale contextual information and third, the C2PSA (Convolutional block with Parallel Spatial Attention) module, which strengthens spatial feature modelling without significantly increasing latency. Beyond object detection, YOLOv11 is designed as a unified framework supporting tasks such as instance segmentation, pose estimation, and oriented bounding box detection. Similar to previous releases, YOLOv11 is provided in multiple scaled variants which go from nano to extra-large making it suitable for deployment across a wide range of devices [3].

2.1.3 YOLOv12

Introduced by Tian et al. [4] in early 2025, YOLOv12 represents a paradigm shift in the YOLO architecture by prioritising attention-centric mechanisms while maintaining the real-time performance characteristic of the YOLO family. Unlike YOLOv11 and previous iterations which rely primarily on convolutional neural networks, YOLOv12 departs from traditional CNN-based

approaches by using attention mechanisms to improve feature representation and modelling capabilities. The architecture introduces three new innovations. First, the *Area Attention* mechanism processes large receptive fields while reducing computational cost compared to standard self-attention. Secondly, *Residual Efficient Layer Aggregation Networks (R-ELAN)* improves feature aggregation and addresses optimisation challenges in larger attention-centric models. Finally, the model implements an optimised attention architecture using FlashAttention for memory access overhead alongside the removal of positional encoding for a cleaner and faster model. Similar to the previous models, YOLOv12 offers five scaled variants: YOLOv12n (nano), YOLOv12s (small), YOLOv12m (medium), YOLOv12l (large), and YOLOv12x (extra-large).

2.2 RF-DETR

RF-DETR (Neural Architecture Search for Real-Time Detection Transformers) [5] is a recent object detection framework designed to bridge the gap between high-capacity transformer-based detectors and real-time specialist models. It addresses key limitations of both YOLO-style detectors, which rely on heuristic post-processing such as Non-Maximum Suppression (NMS), and standard DETR architectures, which often suffer from high computational overhead. As a result, RF-DETR is particularly well suited for real-time applications such as road sign detection.

Architecturally, RF-DETR adopts a streamlined end-to-end transformer design that eliminates hand-crafted components including anchor generation and NMS. RF-DETR builds upon LW-DETR [6] by integrating a self-supervised DINOv2 backbone. This backbone provides robust, high-quality visual representations that generalise better on small datasets. Moreover, unlike Deformable DETR, which uses a multi-scale self-attention mechanism, RF-DETR extracts features from a single-scale backbone. This reduces computational overhead while maintaining the spatial awareness necessary for detecting small objects like traffic signs. Finally, to achieve real-time speeds, the model utilises a lightweight decoder stack and an efficient multi-scale projector to bridge the pre-trained ViT backbone and the decoder stack.

To accommodate various hardware constraints, RF-DETR is released in a family of six sizes: Nano, Small, Medium, Large, XLarge, and 2XLarge. The Nano and Small variants are NAS-discovered configurations optimised for maximum speed (e.g., Nano achieves 2.3ms latency with 30.5M parameters). High-capacity variants like XLarge (126.4M parameters) and 2XLarge (126.9M parameters) prioritise maximum Average Precision, with RF-DETR 2XLarge being the first real-time detector to surpass 60 mAP on the COCO benchmark.

2.3 Faster R-CNN

Faster R-CNN is a two-stage object detection framework that prioritises detection accuracy and strong localisation over raw inference speed. The first stage effectively tells the network "where to look." The RPN slides a small window over the convolutional feature map generated by the backbone. At each sliding window location, it evaluates multiple Anchor Boxes of different scales and aspect ratios to predict whether a region contains an object (objectness score) and roughly refines its coordinates. This allows the model to propose potential Regions of Interest (RoIs) without examining the entire image pixel-by-pixel [7].

The second stage refines these proposals. It uses an RoI Pooling layer to extract fixed-size feature vectors from the variable-sized regions proposed by the RPN. These features are then passed through fully connected layers to predict the final specific class and further regress the bounding box coordinates for precise alignment [8].

2.4 RetinaNet

Introduced by Lin et al. [9] in 2017, RetinaNet is a single-stage object detector that addresses the class imbalance problem present in datasets. RetinaNet combines a Feature Pyramid Network (FPN) backbone with a focal loss function, achieving comparable performance with two-stage object detectors, while maintaining the computational efficiency of a one stage approach [9].

RetinaNet makes use of a ResNet-FPN backbone that constructs a multi-scale feature pyramid, allowing the detection of objects in different scales [9]. This contrasts with the earlier single-stage detectors that struggled with the extreme foreground-background class imbalance. RetinaNet's focal loss dynamically changes the contribution of easy and hard examples during training. The RetinaNet architecture is made up of three main components: a backbone network using ResNet with FPN for multi-scale feature extraction, a classification head that predicts object class probabilities at each spatial location, and a box regression component that predicts bounding box offsets from anchor boxes [9].

The focal loss function down-weights the contribution of easy negative examples and focuses on the harder examples. As shown in Equation 2.1, p_t is the model's estimated probability for the ground-truth class, α_t is a weighting factor for class imbalance, and γ is a focusing parameter that the developer controls, to change the rate at which easy examples are down-weighted. When $\gamma = 0$, focal loss is equivalent to standard cross-entropy loss. The authors found $\gamma = 2$ to work well in handling the easy examples [9]. This approach prevents the large number of easy negatives from overwhelming the training process, a critical issue in dense detection scenarios where thousands of anchor boxes are evaluated per image, with only a handful corresponding to actual objects.

With regards to our dataset, focal loss may come up useful when training RetinaNet with specific sign attributes, as in real-world scenarios some signs could be under-represented compared to others, in terms of their condition, sign type and mounting type.

$$FL(p_t) = -\alpha_t(1 - p_t)^\gamma \log(p_t) \tag{2.1}$$

2.5 FCOS

Fully Convolutional One-Stage (FCOS) is an anchor-free object detection framework that reformulates object detection as a per-pixel prediction task. Unlike anchor-based detectors, FCOS eliminates the need for predefined anchor boxes and heuristic matching rules by directly regressing bounding box coordinates at each spatial location. Specifically, for every location on a feature map, FCOS predicts a 4-dimensional vector (l, t, r, b), representing the distances from the location to the left, top, right, and bottom sides of the associated ground-truth bounding box [10]. This design simplifies the detection pipeline and allows all pixels within a bounding box to contribute as positive training samples. To effectively handle objects at different scales, FCOS is integrated with a Feature Pyramid Network (FPN), enabling multi-level feature representations while sharing prediction heads across pyramid levels. A key challenge in dense, per-pixel object detection is that locations far from the center of an object tend to produce low-quality bounding box predictions. To mitigate this issue, FCOS introduces a center-ness branch that estimates the proximity of a location to the center of its corresponding object. The center-ness target for a pixel with regression targets l^*, t^*, r^*, b^* is defined as

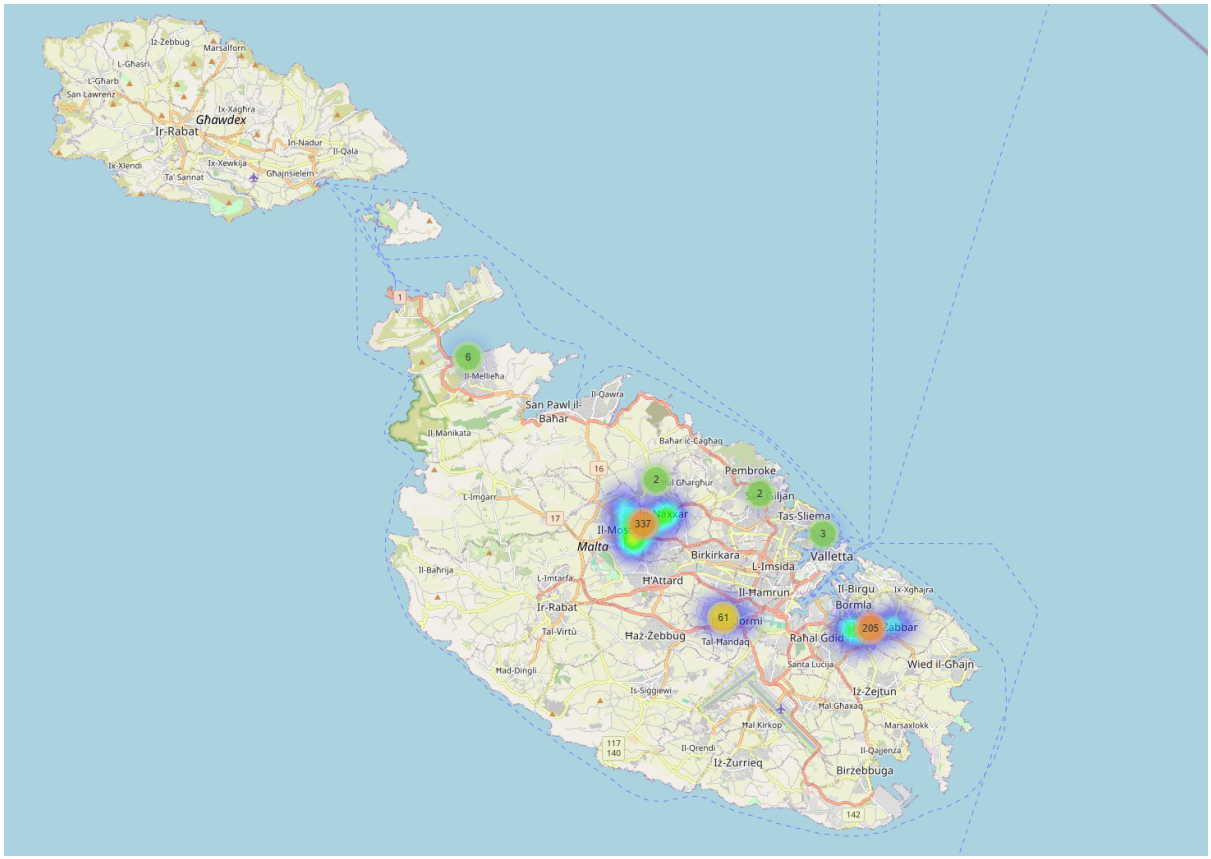
$$\text{centerness}^* = \sqrt{\frac{\min(l^*, r^*)}{\max(l^*, r^*)} \times \frac{\min(t^*, b^*)}{\max(t^*, b^*)}}. \tag{2.2}$$

This formulation produces values in the range $[0, 1]$, assigning higher scores to locations closer to the object center. During inference, the predicted center-ness score is multiplied with the classification confidence to suppress low-quality detections prior to non-maximum suppression, leading to improved localization accuracy and reduced false positives [10].

Chapter 3

Data Preparation

3.1 Dataset Planning



Spatial distribution and density of captured traffic sign images across Malta.

The spatial distribution of the captured images is visualised above using a geographic heatmap. The map illustrates that data collection was concentrated in both urban and suburban regions, with higher densities observed in central and densely populated areas and surrounding localities. Additional samples were collected in coastal and less urbanised areas to improve geographic coverage and reduce location bias.

This distribution strategy ensures that the dataset captures a wide range of environmental conditions, including variations in road layout, background clutter, lighting conditions, and sign placement. By spreading data collection across multiple regions rather than a single locality,

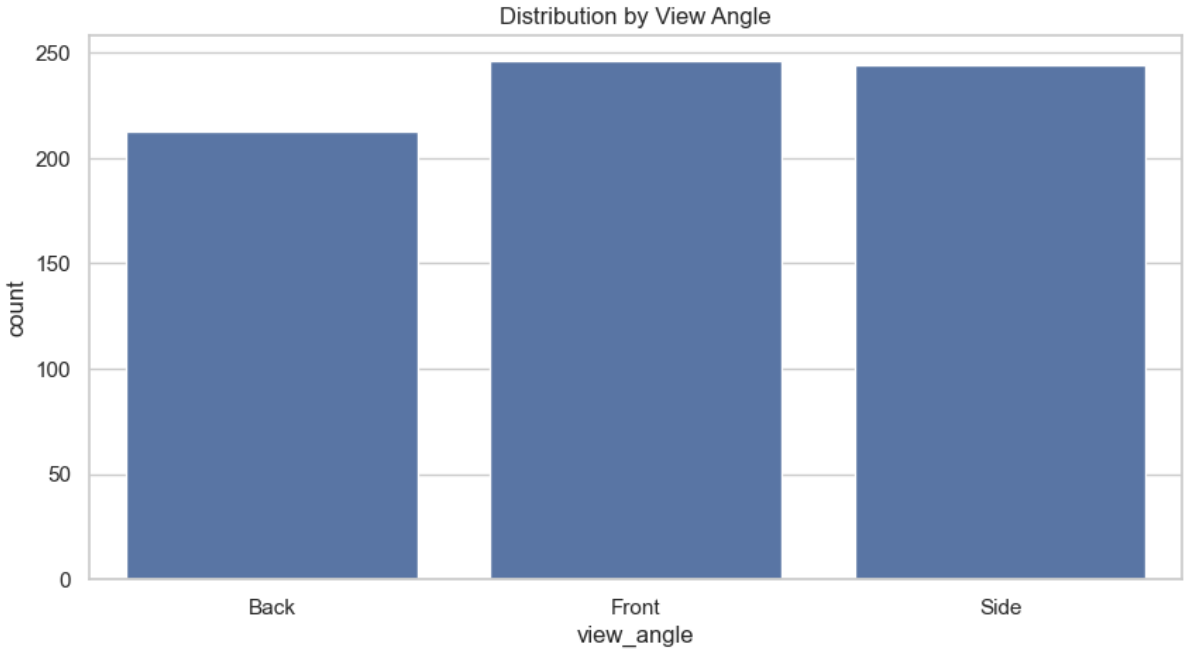
the dataset improves robustness and generalisation potential for real-world deployment. The heatmap confirms that while image density varies by location, coverage spans the majority of the island, supporting the goal of creating a geographically diverse and representative traffic sign dataset.

3.2 Sign Classes and Attributes

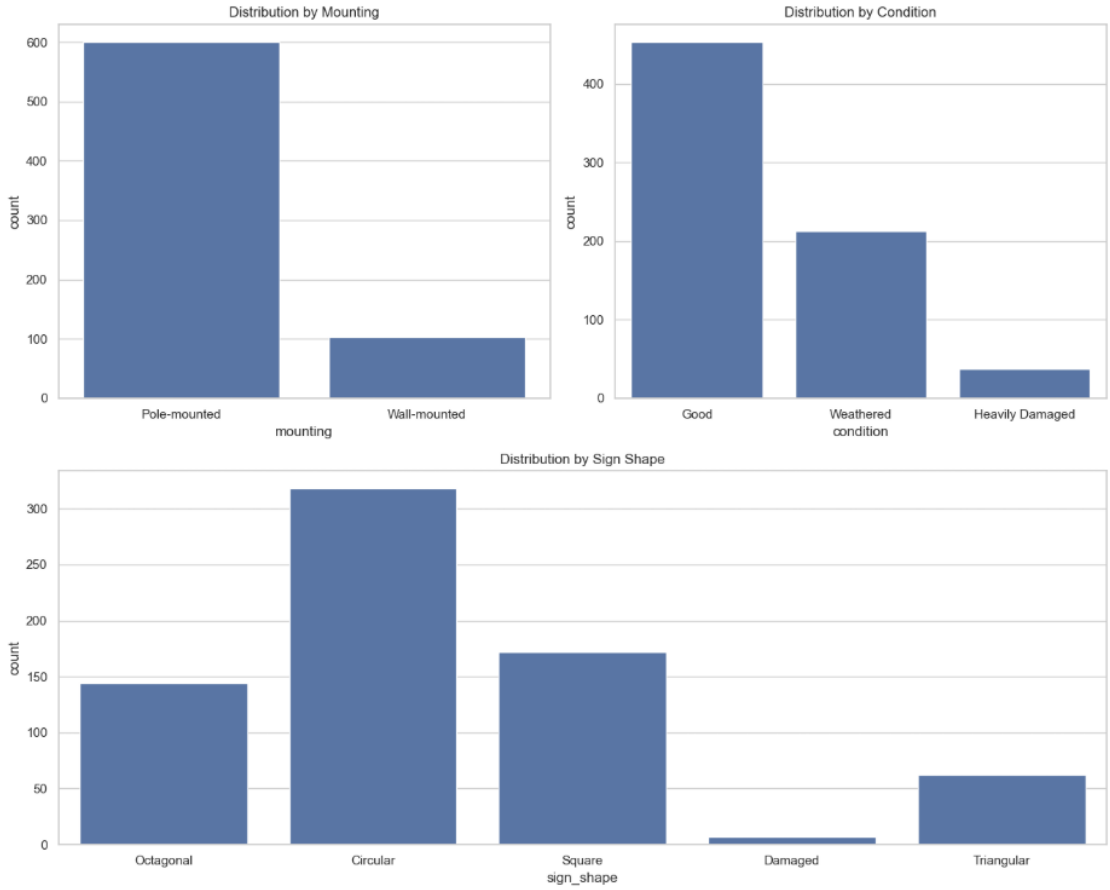
3.2.1 Traffic Sign Classes

For this assignment, we had to take images of 6 types of signs. These are Stop, No Entry, Pedestrian Crossing, Roundabout Ahead, No Through Road and Blind-Spot Mirror road signs. For each sign, there are multiple annotations that had to be labelled for each sign. These were:

- Viewing Angle: Front, Side, Back
- Mounting Type: Wall-mounted, Pole-mounted
- Sign Condition: Good, Weathered, Heavily Damaged
- Sign Shape: Circular, Square, Triangular, Octagonal, Damaged



Distribution of view angles of sign types.



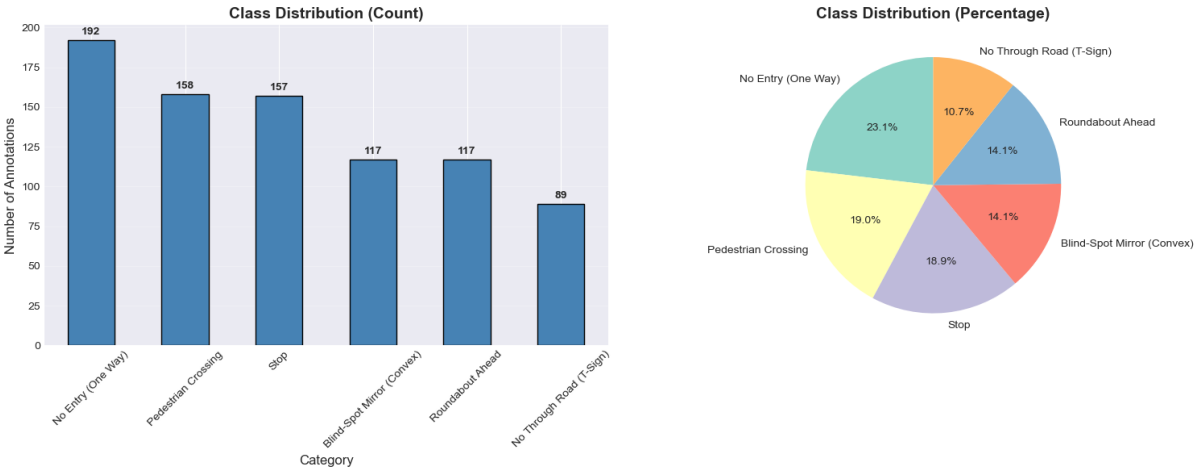
Spatial distribution and density of captured traffic sign images across Malta.

The dataset exhibits imbalances and sparse representation across several attributes. In particular, wall-mounted signs and heavily damaged signs are under-represented compared to pole-mounted and good-condition signs. Similarly, certain sign shapes, such as damaged and triangular, appear far less frequently than circular or square signs. This imbalance will affect the models which will train these particular attributes, as it will limit their ability to generalise for these categories and this will further be discussed when analysing the results of the respective models.

3.3 Annotation Process

Label Studio was used to annotate all images with bounding boxes and attribute labels. Annotations were exported in JSON format and converted into COCO format. Personal identifiers such as faces and vehicle number plates were removed or masked using blurring or generative inpainting to ensure GDPR compliance.

3.4 Dataset Statistics and Balance



Class distribution of traffic sign annotations shown as absolute counts (left) and relative percentages (right).

The distribution of sign classes in the dataset is visualised above using both absolute counts and relative percentages. While the dataset is not perfectly balanced, the class frequencies are reasonably well distributed, with no single category overwhelmingly dominating the annotations. The *No Entry (One Way)* class is the most frequent, comprising 192 instances (approximately 23.1%), whereas the *No Through Road (T-Sign)* class is the least represented with 89 instances (approximately 10.7%). The remaining classes, including *Pedestrian Crossing*, *Stop*, *Blind-Spot Mirror (Convex)*, and *Roundabout Ahead*, each account for roughly 14% to 19% of the dataset. Although some level of class imbalance is present, the variation is moderate and reflects realistic road scene distributions rather than artificial uniformity.

3.5 Dataset Split and Augmentation

The dataset was split into training, validation, and test sets, ensuring no overlap of physical sign instances. Data augmentation techniques such as rotation, scaling, and colour jitter were applied, depending on the object detection technique used.

Chapter 4

Implementation of the Object Detectors

4.1 Matthew: RF-DETR and YOLOv12 (Viewing Angle)

4.1.1 RF-DETR Implementation

The implementation of RF-DETR for sign detection was developed using the official Roboflow documentation [11] and the reference notebook example [12]. Training was conducted on Google Colab due to the lack of local access to CUDA-enabled GPUs, specifically targeting T4 runtime.

RF-DETR expects a specific COCO-format dataset structure where annotations and images are organised into corresponding train, validation, and test directories. The dataset structure was configured as follows:

Listing 4.1: Directory Structure for RF-DETR

```
dataset/
  train/
    _annotations.coco.json
    [images]
  valid/
    _annotations.coco.json
    [images]
  test/
    _annotations.coco.json
    [images]
```

Two modifications were required to ensure compatibility with RF-DETR's annotation format expectations. First, RF-DETR requires a `supercategory` field for each category in the COCO annotations. Since the original dataset lacked this field, it was added programmatically to all category definitions. Second, the original category IDs ranged from 1-6, whereas RF-DETR expects zero-indexed category IDs (0-5). The annotations were preprocessed to decrement all `category_id` values by 1 to align with model's class indexing scheme.

The RF-DETR Nano model was selected for its optimal balance between detection accuracy, inference speed, and hardware limitations. The complete training configuration is shown in Listing 4.2

Listing 4.2: RF-DETR Training Configuration

```
model.train(
    dataset_dir = "/content/dataset/",
    epochs = 50,
    batch_size=4,
    grad_accum_steps = 4,
    lr=1e-4,
    lr_scheduler='cosine',
    lr_min_factor=0.01,
    early_stopping=True,
    early_stopping_patience=10,
    early_stopping_min_delta=0.001,
    checkpoint_interval=5,
    output_dir = output_path,
    val_interval = 1,
    weight_decay=0.0001,
    clip_max_norm=0.1
)
```

The training configuration was designed over 50 epochs. A cosine annealing learning rate scheduler was used, starting at 1×10^{-4} and decaying to 1×10^{-6} (minimum factor 0.01). This was done to provide a smooth learning rate reduction to prevent overfitting in later epochs. Early stopping was also implemented with a patience of 10 epochs and minimum improvement threshold of 0.001 (0.1% mAP) to automatically terminate training when validation performance plateaued. As recommended by the Roboflow guide, gradient accumulation with 4 steps and batch size of 4 was used to train effectively on the T4 GPU. Regularisation techniques included weight decay to prevent overfitting and gradient clipping to stabilise training. Model checkpoints were saved every 5 epochs for backups and analysis after training.

4.1.2 YOLOv12 Implementation

The YOLOv12 Nano model was trained locally using the Ultralytics framework. Unlike RF-DETR which required cloud-based GPU resources, YOLOv12's Metal Performance Shaders (MPS) support enabled efficient training on Apple Silicon hardware.

YOLOv12 uses the standard YOLO annotation format, where each image has a corresponding text file containing normalised bounding box coordinates and class labels. The dataset was organised into train, validation, and test splits, with a `data.yaml` file specifying the dataset paths and the three class names (Back, Front, Side).

The model was initialised with YOLOv12 Nano pre-trained weights and fine-tuned on the viewing angle dataset. Training was done up to 100 epochs using the AdamW optimiser with an initial learning rate of 1×10^{-3}, a batch size of 32, and image resolution of 640x640 pixels to preserve spatial details. Furthermore, augmentation was used to simulate varied environmental conditions and perspectives found in road scenes. Finally, to prevent the model from overfitting on the training distribution, early stopping with a patience of 20 epochs was added. The complete training configuration is shown in Listing 4.3.

Listing 4.3: YOLOv12 Training Configuration

```
results = model.train(
    data=yaml_path,
    epochs=100,
    imgsz=640,
```

```
    batch=32,
    device='mps',
    workers = 4,
    patience = 20,
    project = 'runs/viewing_angle',
    name='Maltese_Signs_ViewingAngle',
    optimizer = "AdamW",
    lr0=0.001,
    save_period = 10,
    augment=True,
    plots=True,
    verbose=True
)
```

4.2 Liam Jake: YOLOv12 and Faster R-CNN (Sign Shape)

4.2.1 YOLOv12 Implementation

The YOLOv12 Nano (yolov12n.pt) model was selected for the sign shape classification task. Training was done using the Ultralytics framework, initialised with pre-trained weights to leverage transfer learning. The implementation details are shown in Listing 4.4. The model was trained for a maximum of 100 epochs with an image resolution of 640×640 pixels. A dynamic batch size (batch=-1) was utilised, allowing the framework to automatically determine the optimal batch size based on the available GPU memory.

Listing 4.4: YOLOv12n Training Configuration

```
model.train(
    data='/content/data.yaml',
    epochs=100,
    imgsz=640,
    batch=-1,
    device=0,

    patience=20,

    mosaic=1.0,
    close_mosaic=10,

    degrees=15,
    scale=0.5,
    fliplr=0.0,

    optimizer='auto',

    project='runs/detect',
    name='traffic_signs_yolov12n2',
    exist_ok=True
)
```

To ensure training stability and prevent overfitting, specific hyperparameters were configured.

Early stopping was set to 20 epochs (patience=20). A specific augmentation strategy was tailored for traffic signs: geometric augmentations included a rotation range of $\pm 15°$ and a scaling factor of 0.5 to simulate varying distances. Horizontal flipping (fliplr) was explicitly disabled (0.0). This is important for traffic signs, as they are direction-specific and maintaining orientation is necessary for accurate shape classification. Mosaic augmentation was enabled (mosaic=1.0) to improve small object detection but disabled for the final 10 epochs to fine-tune on natural images.

4.2.2 Faster R-CNN Implementation

Faster R-CNN was implemented using the Detectron2 framework, utilising a ResNet-50 backbone combined with a Feature Pyramid Network (FPN). The training configuration is detailed in Listing 4.5.

Listing 4.5: Faster R-CNN Training Configuration

```
register_coco_instances("sign_train", {}, "/content/COCO-
    based_COCO_sign_shape/annotations/train.json", "/content/COCO-
    based_COCO_sign_shape/images/train")
register_coco_instances("sign_val", {}, "/content/COCO-
    based_COCO_sign_shape/annotations/val.json", "/content/COCO-
    based_COCO_sign_shape/images/val")

cfg = get_cfg()

cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/
    faster_rcnn_R_50_FPN_3x.yaml"))

cfg.DATASETS.TRAIN = ("sign_train",)
cfg.DATASETS.TEST = ("sign_val",)

cfg.DATALOADER.NUM_WORKERS = 2

cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/
    faster_rcnn_R_50_FPN_3x.yaml")

cfg.SOLVER.IMS_PER_BATCH = 6
cfg.SOLVER.BASE_LR = 0.00025
cfg.SOLVER.MAX_ITER = 2000
cfg.SOLVER.STEPS = (1500,)
cfg.SOLVER.GAMMA = 0.1

cfg.MODEL.ROI_HEADS.NUM_CLASSES = 5

cfg.OUTPUT_DIR = "/content/output"
)
```

Unlike YOLO epoch-based training, this model was trained for 2,000 iterations (`MAX_ITER`). A batch size of 6 images (`IMS_PER_BATCH`) was selected to accommodate the memory constraints of the T4 GPU on Google Colab. The base learning rate was set at 2.5×10^{-4} with a scheduler that decays the rate by a factor of 0.1 at 1,500 iterations. This allows the model to fine-tune its parameters more precisely towards the end of training. The ROI heads were configured

for 5 classes (`NUM_CLASSES = 5`), corresponding to the specific sign shapes (Circular, Square, Triangular, Octagonal, Damaged) present in the dataset.

4.3 Daniel: YOLOv11 and FCOS (Mounting Type)

4.3.1 YOLOv11 Implementation

The YOLOv11 model was trained using the following hyperparameters.

Listing 4.6: YOLOv11 Training Configuration

```
results = model.train(
    data='YOLO_COCO/data.yaml',
    epochs=100,
    imgsz=640,
    batch=5,
    name='yolo11m_100epochs',
    device=0,  # GPU
    val=True,
    save=True,
    plots=True,
    show=True,
    patience = 10
)
```

Training is set for 100 epochs to ensure that there is a convergence while avoiding overfitting. An input resolution of 640×640 was used as it preserves enough spatial detail for small objects while maintaining reasonable GPU memory usage and inference speed. A batch size of 5 was used to ensure that the GPU does not run of of memory and that the model maintains a stable training rate. Patience was set to 10 so that if there is no further after 10 epochs, training automatically stops. Performance metrics and the model itself were saved to be used for model comparison and visualisation in the later notebooks and sections.

4.3.2 FCOS Implementation

An anchor-free FCOS detector was implemented using a ResNet-50 backbone with a Feature Pyramid Network (FPN). The default FCOS detection head was replaced with a custom head configured to predict two classes corresponding to sign mounting types: pole-mounted and wall-mounted. Training was performed for up to 50 epochs with a batch size of 4 images. A relatively small batch size was selected to accommodate GPU memory limitations while maintaining stable training performance. The model was optimised using the AdamW optimiser with a learning rate of 3×10^{-4} and a weight decay of 0.0001 to support stable convergence while fine-tuning pretrained backbone weights. To mitigate class imbalance, a weighted random sampling strategy was applied at the image level, increasing the sampling probability of images containing wall-mounted signs. Data loading was configured with zero worker processes to avoid multiprocessing issues during training. A multi-step learning rate scheduler was employed, reducing the learning rate by a factor of 0.1 at epochs 30 and 40 to encourage optimisation during early training followed by finer parameter adjustments as convergence approached. The model achieving the lowest validation loss was saved for final evaluation, with periodic checkpoints stored every ten epochs.

4.4 Liam Azzopardi: YOLOv8 and RetinaNet (Condition)

4.4.1 YOLOv8 Implementation

For this assignment, it was decided to use YOLOv8-n for computational efficiency reasons. The YOLOv8-n model was trained using the configuration shown in Listing 4.7, attention was given to reproducibility and training stability by providing a fixed seed value, eliminating randomness from data augmentation and weight initialising.

Listing 4.7: YOLOv8 Training Configuration

```
results = model.train(
    data='data.yaml',
    epochs=100,
    imgsz=640,
    batch=8,
    patience=25,
    save=True,
    device='0',
    workers=8,
    project='traffic_signs',
    name='yolov8_final_run',
    exist_ok=True,
    optimizer='auto',
    verbose=True,
    seed=42,
    deterministic=True,
    plots=True,
    save_period=1,
    warmup_epochs=3.0,
    cos_lr=True
)
```

A batch size of 8 was chosen mainly due to GPU memory limitations while training at 640×640 resolution. This resolution proves to provide a good balance between computational efficiency and detection accuracy. Since the optimizer was set to auto-selection mode (defaulted to AdamW with its associated default learning rate) the 3 epoch warm-up period prevents training instability by starting with a small fraction of the default learning rate and linearly increasing it over the first 3 epochs. This prevents the large, unpredictable gradients that can occur in early training from destabilizing the optimization process. Cosine annealing was used as the learning rate scheduler to smoothly decerease the learning rate over the remaining training period. The model was implemented to perform a maximum of 100 epochs, with a patience of 25 epochs to prevent overfitting and reduce unnecessary computation. Training would stop prematurely if validation set performance did not improve for 25 consecutive epochs, automatically selecting the best checkpoint (best model saved if performance improves every epoch) based on validation metrics.

4.4.2 RetinaNet Implementation

RetinaNet was implemented using a ResNet-50-FPN backbone. The classification head of RetinaNet was modified to output three classes corresponding to sign conditions (Good, Weathered and Heavily Damaged). Training was conducted over a maximum of 15 epochs with a batch size

of 4 images. A small batch size number was used as training was starting to take a lot of time to finish one epoch and a batch size of 4 provided faster training with the GPU memory constraints. The AdamW optimizer was used with a learning rate of 0.0001 and a weight decay of 0.0001 was selected to enable stable convergence while fine-tuning the pretrained ResNet-50 weights. The number of data loading workers was set to 0, this is to avoid having multiprocess complications during training. Patience was set to 5 to prevent overfitting and reduce unnecessary computation. Training terminated if the validation loss failed to improve for 5 consecutive epochs, with the best-performing checkpoint with the lowest validation loss preserved for final evaluation. Learning rate scheduling was also used using a factor of 0.5 when performance plateaued for 3 consecutive epochs to attempt enabling aggressive optimisation during early training when the gradients remained large, then automatically transitions to detailed parameter adjustments as the model approaches convergence.

Chapter 5

Evaluation of the Object Detectors

5.1 All: Sign Type Results

5.1.1 YOLOv8-n

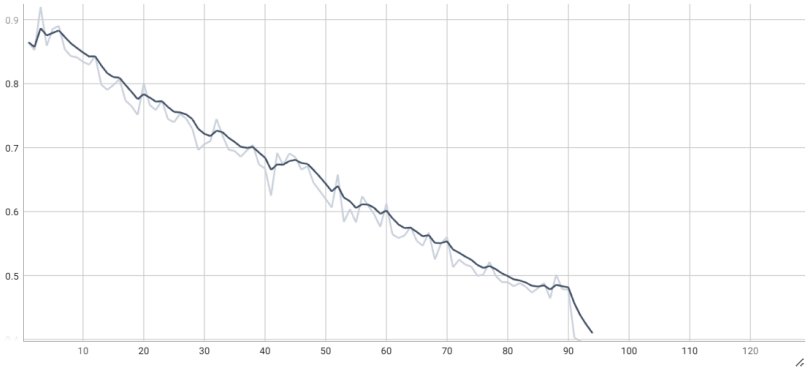


Figure 5.1: YOLOv8 *box_loss*

Looking at the *box_loss* (figure 5.1) for YOLOv8-n, it is evident that the model demonstrated smooth and consistent convergence throughout training. The loss decreases from 0.8647 to 0.4101, indicating effective learning of bounding box localisation. However, the plot shows that there is a continued downward trend with no clear plateau, indicating that there is potential for further improvement if the model trained with more epochs or learning rate scheduling.

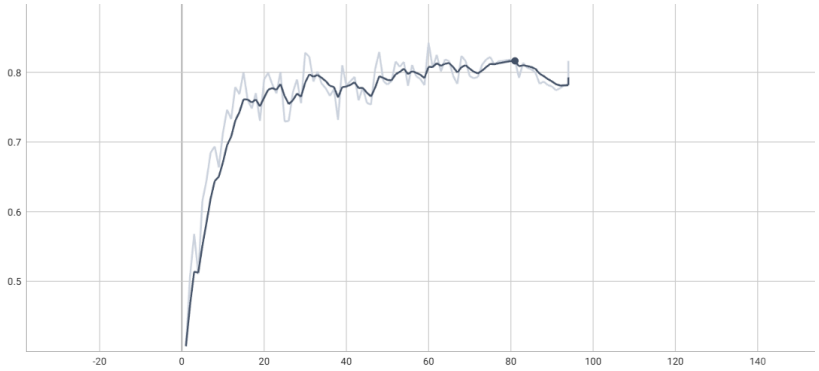


Figure 5.2: YOLOv8 - mAP@0.5

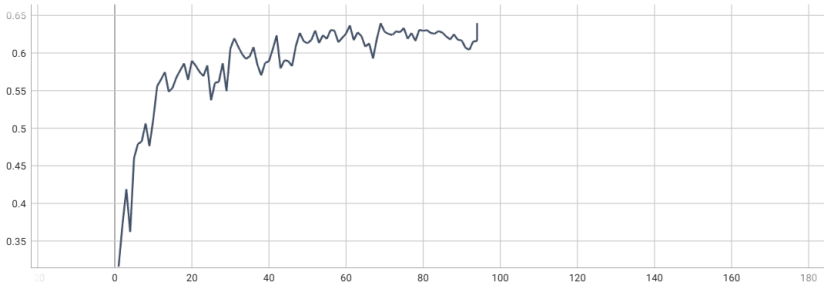


Figure 5.3: YOLOv8 - mAP@0.5-0.9

For detection performance, looking at figure 5.2, *mAP@0.5* for YOLOv8 improved significantly throughout the early epochs, going from 0.407 to 0.713 in the first 10 epochs. At around the 20th epoch, the *mAP* seems to stabilise and plateau with oscillations. Towards the end of training, the mAP seems to start declining slightly, which could be caused by potential overfitting between the training set and the validation set. *mAP@0.5-0.9* plot in figure 5.3 indicates that there is a similar trajectory between *mAP@0.5-0.9* and *mAP@0.5*, which is expected given the stricter *IoU* thresholds required across the range. *mAP@0.5-0.9* peaks at epoch 69, while *mAP@0.5* peaks at epoch 60, this late peak suggests that the model requires additional training to maximise performance across stricter *IoU* thresholds, even after detection at *IoU=0.5* has reached its optimum. The best model is saved based on *mAP@0.5:0.95* performance, the best model prioritises localisation precision across the stricter IoU thresholds rather than maximising detection at *IoU=0.5*.

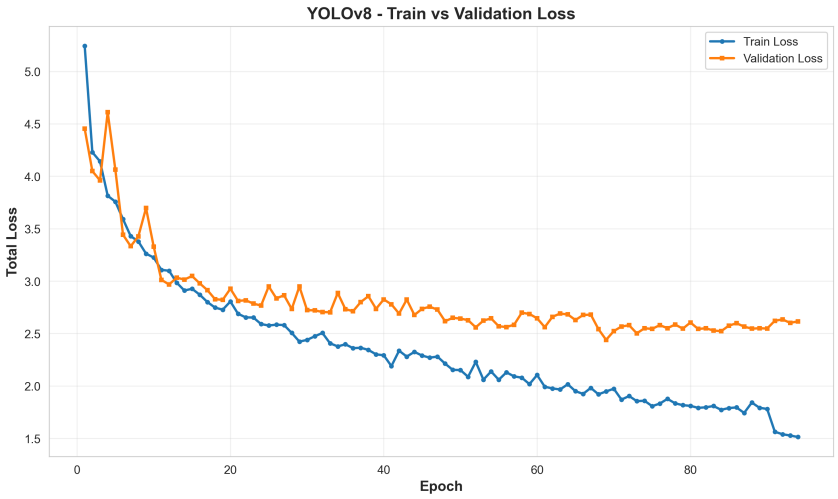


Figure 5.4: Loss - Train vs Validation - YOLOv8

Looking at the specific train versus validation loss plot in figure 5.4, both losses decrease significantly during the early stages of training, falling from approximately 5.3 and 4.5 to around 3.0 within approximately the first 15 epochs. However, a concerning pattern starts to form after epoch 20, the training loss curve continues to decline throughout training, reaching approximately 1.5 by epoch 94, while the validation loss plateaus around 2.6-2.8 after epoch 20 and shows no further improvement, this demonstrates significant overfitting. Even after this clear divergence between the training loss and validation loss, *mAP@0.5* and *mAP@0.5:0.95* continued improving until epochs 60 and 69 respectively, demonstrating that the composite loss function and detection metrics measure different aspects of performance. This justifies why using *mAP@0.5:0.95* for best checkpoint selection rather than using validation loss.

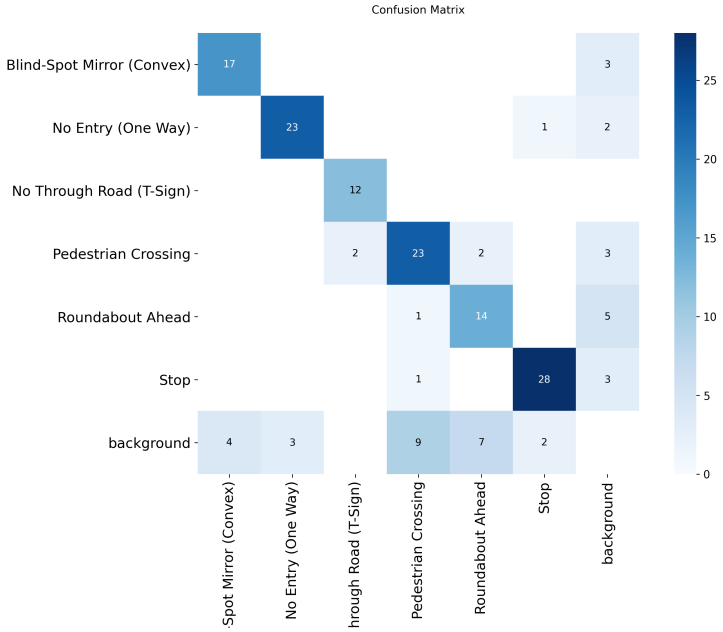


Figure 5.5: Confusion Matrix on the Test Set Using YOLOv8

The best model that was saved for YOLOv8-n achieved a final *mAP@0.5* of 0.8191 and *mAP@0.5:0.95* of 0.6409 on the validation set. When the best model was used to evaluate the test set, the *mAP@0.5* to 0.8552 while the *mAP@0.5:0.95* improved to 0.7225, indicating good generalisation on unseen examples. This also indicates that the validation set may have had much more difficult examples than the test set.

Table 5.1: Per-class detection performance for YOLOv8-n on test set

Class	**Images**	**Instances**	**P**	**R**	**mAP@0.5**	**mAP@0.5:0.95**
Overall	122	149	0.922	0.772	0.855	0.723
Stop	31	31	0.929	0.903	0.928	0.832
No Through Road	14	14	0.980	0.857	0.912	0.769
No Entry	24	26	0.915	0.885	0.901	0.745
Blind-Spot Mirror	21	21	0.966	0.810	0.893	0.801
Pedestrian Crossing	30	34	0.878	0.638	0.816	0.670
Roundabout Ahead	19	23	0.861	0.539	0.682	0.517

Looking at Table 5.1, we can see some performance patterns, including an overall high precision and moderate recall, indicating that it misses the detection of signs at time but does not falsely detect signs. Stop signs performed the best with superior performance compared to signs like 'Roundabout Ahead' which performed the worst, with almost half of the 'Roundabout Ahead' signs being completely missed. This poor performance could be due to the fact that there is a mix of circular and triangular sign types of 'Roundabout Ahead' in the dataset which could contribute to this poor performance. Signs with distinctive features, such as Stop signs being the only octagonal sign shape, could reflect this superior performance, along with it being one of the most populated sign types in the dataset.

The confusion matrix (Figure 5.5) reveals the error patterns underlying YOLOv8-n's detection performance on the test set. Confusion with the background class represents the false negatives where ground truth signs were not detected. Pedestrian Crossing had 9 missed detections, Roundabout Ahead had 7 , and Blind-Spot Mirror had 4. This pattern directly explains the

reduced recall observed in Table 5.1, particularly for 'Roundabout Ahead' that resulted in a Recall of 0.539 and 'Pedestrian Crossing' that resulted in a recall of 0.638.

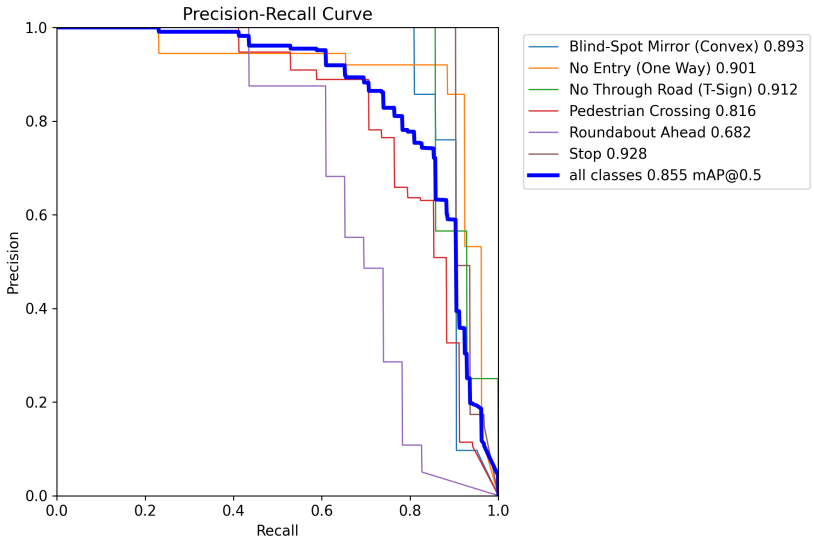


Figure 5.6: Precision-Recall Curve for YOLOv8

The precision-recall curves (Figure 5.6) corroborate the performance patterns observed in the confusion matrix and per-class metrics. Stop, No Through Road, and No Entry maintain high precision across wide recall ranges, consistent with their minimal background confusion seen in Figure 5.5 and strong recall values in Table 5.1. The sign type 'Roundabout Ahead' Precision-Recall curve validates the confusion matrix findings, showing an early precision decline at recall of only 0.55-0.6. This confirms that the class's poor performance originates from fundamental detection difficulty rather than suboptimal confidence threshold tuning. The 7 missed detections and 5 false positives observed in the confusion matrix reflect systematic challenges in identifying this sign type using YOLOv8-n. Pedestrian Crossing occupies the middle ground, with its steeper precision drop beginning around recall of 0.75, directly corresponding to the 9 background confusions observed in the confusion matrix. The PR curve reveals that improving recall for this class necessarily introduces more false positives, explaining the precision-recall trade-off present in its moderate performance.

5.1.2 YOLOv11-m

When looking at the results for YOLOv11-m, it can be clearly seen that there is a stable convergence across all of the metrics which were monitored. The training *box_loss* decreases from about 0.91 in the first epochs to around 0.64 by the final epoch, which indicates that there is increasing bounding-box accuracy as the model trains on further epochs. There is a similar trend in the classification loss where values reduce from 2.7 to 0.5. Validation losses also follow a similar trend with validation box loss decreasing to about 0.8. The close alignment of validation and training loss shows that there is limited overfitting and shows that the model generalises well to unseen data.

When it comes to detection performance, the model achieves a final *mAP@0.5* score of 0.80. The steady improvement of *mAP@0.5* throughout training demonstrates effective learning across increasing localisation strictness. Precision and recall values converge to approximately 0.89 and 0.74 respectively, reflecting a balanced trade-off between false positives and false negatives. These results shows that the trained model achieves reliable localisation and classification per-

formance. In addition to strong performance at *mAP@0.5*, the model achieves a *mAP@0.5–0.95* score of approximately 0.62. The relatively smaller gap between *mAP@0.5* and *mAP@0.5–0.95* indicates that the model maintains robust performance even under stricter localisation requirements which suggests that the network is capable of precise object localisation in addition to accurate classification.

When analysing YOLOv11-m in the train versus validation plot, it can be seen that training loss decreases steadily through the learning process. Validation loss initially spikes early, but then after about 10 epochs, this stabilises and follows a downward trend, which is very similar to the training loss. The persistent but controlled gap between training and validation loss indicates a healthy bias-variance trade-off, with the model continuing to learn meaningful representations without memorising the training data.

Table 5.2: YOLOv11 Detection Performance per Class

Class	**Images**	**Instances**	**P**	**R**	**mAP@0.5**	**mAP@0.5:0.95**
Overall	122	149	0.878	0.840	0.888	0.746
Stop	31	31	0.903	0.899	0.937	0.848
No Through Road	14	14	0.932	0.980	0.990	0.856
No Entry	24	26	0.788	0.923	0.881	0.737
Blind-Spot Mirror	21	21	0.949	0.893	0.959	0.864
Pedestrian Crossing	30	34	0.885	0.735	0.845	0.637
Roundabout Ahead	19	23	0.813	0.609	0.716	0.531

When looking at this table, it can be seen that the model achieves high recall and precision scores for all classes. The class which has the worst performance in this case is roundabout ahead which has the lowest recall score, meaning that the model is struggling to detect this type of sign. This is further supported by the low *mAP@0.5–0.95* score which means that the model is struggling to pinpoint the exact location of the sign.

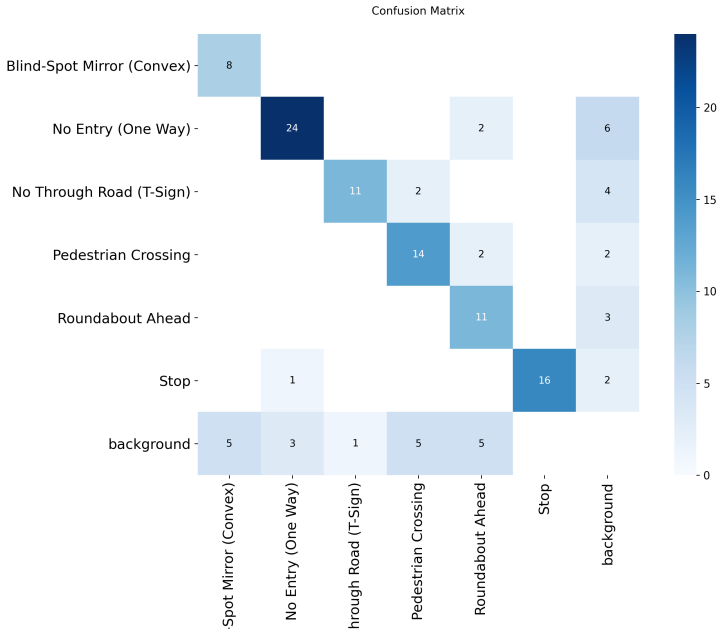


Figure 5.7: Confusion Matrix on the Test Set Using YOLOv11

When looking at the confusion matrix, it can be seen that the model performs the best on the No Entry and No Through Road class types. The model struggled to detect about 40% of the Blind-Spot Mirror and misclassified a significant number of Roundabout Ahead signs, supporting the previous table which shows that the model performed the weakest on this class.

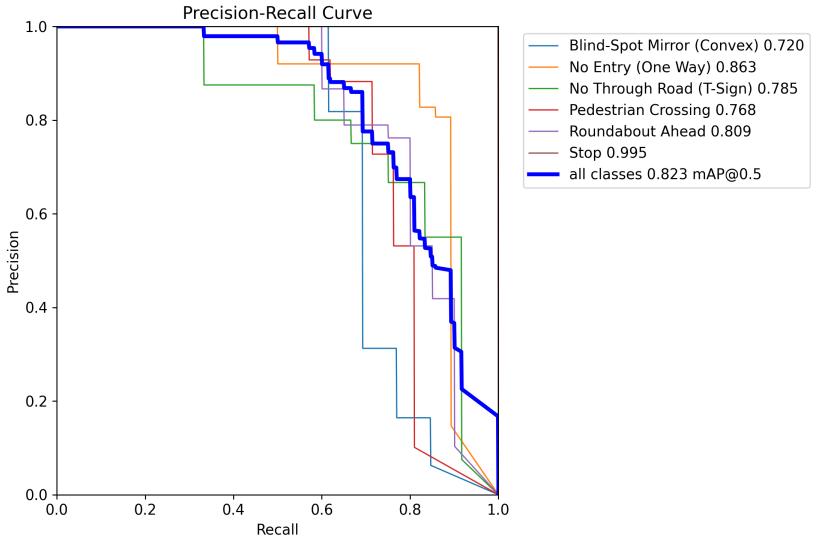


Figure 5.8: Precision-Recall Curve for YOLOv11

This precision–recall curve shows overall strong detection performance, with an *mAP@0.5* of 0.823 across all classes, indicating a good balance between precision and recall. The Stop sign class performs exceptionally well, maintaining high precision even at high recall, which suggests the model detects it very reliably. Classes like No Entry, Roundabout Ahead, and No Through Road also show solid performance, while Blind-Spot Mirror and Pedestrian Crossing degrade more quickly as recall increases, indicating more false positives at higher sensitivity.

5.1.3 YOLOv12-n

The YOLOv12-n model demonstrated solid detection performance in the test set, achieving an overall mAP@0.5 of 0.802 in all classes. This indicates a reliable classification of sign types under standard IoU thresholds, though with room for improvement in stricter localisation scenarios.

Table 5.3: Per-class Performance for YOLOv12-n

Class	**Instances**	**P**	**R**	**mAP@0.5**
Overall	110	0.799	0.773	0.802
Blind-Spot Mirror	13	0.750	0.692	0.605
No Entry	28	0.774	0.857	0.879
No Through Road	12	0.800	0.667	0.810
Pedestrian Crossing	21	0.875	0.667	0.741
Roundabout Ahead	20	0.833	0.750	0.781
Stop	16	0.762	1.000	0.995

Looking at Table 5.3, the model shows high precision in most classes, indicating few false positives, but recall varies significantly. The Stop sign achieves perfect recall and the highest mAP@0.5, likely due to its distinctive octagonal shape. In contrast, the Blind-Spot Mirror class

has the lowest recall and mAP@0.5, suggesting challenges in detecting convex mirrors amid cluttered backgrounds. No Entry and No Through Road perform strongly, while Pedestrian Crossing and Roundabout Ahead exhibit moderate results, possibly influenced by variations in sign shapes (triangular vs. circular for Roundabout Ahead).

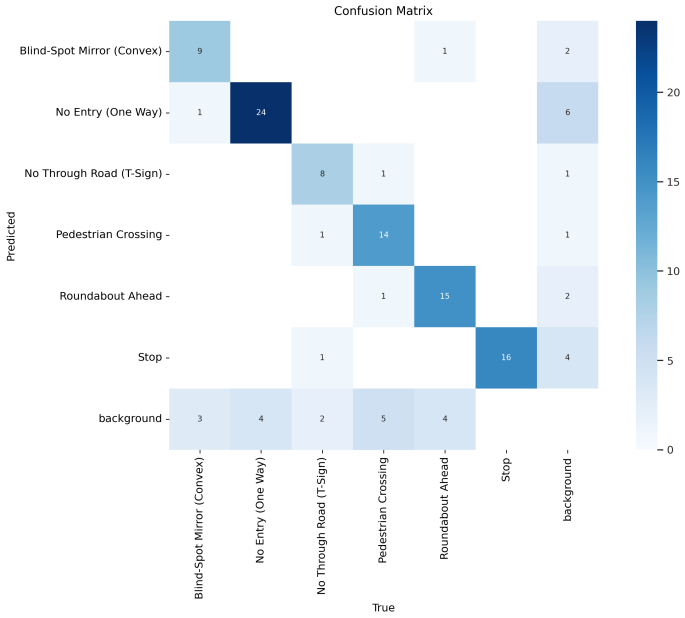


Figure 5.9: Confusion Matrix on the Test Set Using YOLOv12n

The confusion matrix 5.9 highlights the model's classification patterns. Most confusions occur with the background class, representing missed detections (false negatives). For instance, Pedestrian Crossing has 5 missed detections, Roundabout Ahead has 4, and Blind-Spot Mirror has 3, which aligns with their lower recall values. There are minor misclassification between classes, such as 1 Blind-Spot Mirror misclassified as No Entry and 1 as Stop, indicating occasional feature overlap. Notably, the model introduces some false positives from background, particularly for No Entry (6) and Stop (4), contributing to slightly reduced precision in those classes.

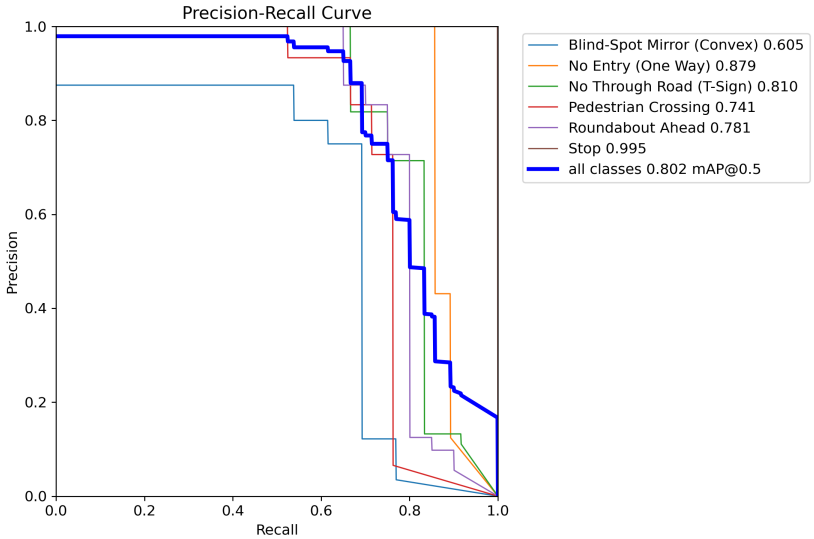


Figure 5.10: PR Curve on the Test Set Using YOLOv12n

The precision-recall curves 5.10 further confirm these observations. Stop maintains near-perfect

precision across all recall levels, consistent with its zero missed detections. No Entry and No Through Road show gradual precision drops at higher recalls, reflecting robust performance. However, Blind-Spot Mirror experiences a sharp decline in precision after recall of approximately 0.6, confirming detection difficulties. Pedestrian Crossing and Roundabout Ahead display intermediate curves, with precision dropping around recall 0.7-0.8, suggesting a trade-off where pushing for higher recall introduces more errors. Overall, the curves demonstrate that the model's strengths lie in distinctive sign types, while challenges with less salient or variable signs contribute to the average mAP.

5.1.4 RF-DETR

Figure 5.11 shows the model's training progression over 38 epochs. The training loss decreased from 6.5 to 2.6, showing that the model learned effectively. However, the validation loss plateaued at around 4.5 after epoch 10, which creates a gap between the training and validation performance. This gap means that some overfitting is present, where the model memorised training data rather than learning general patterns.

Despite this overfitting, the validation metrics continued improving throughout training. Average precision at IoU=0.5 reached approximately 75% and stabilised at around epoch 20. Similarly, Average precision at IoU=0.50:0.95 reached 60% by epoch 25, with minimal improvement afterward. Average recall stabilised at 70%, which means that the model shows consistent detection capability across epochs.

The model was quickly learning in the first 10 epochs, after which improvements became gradual. Both the base model and EMA (Exponential Moving Average) model tracked closely, with the EMA model providing slightly smoother and more stable metrics. Training effectively converged by epoch 25 due to the early stopping hyperparameter (patience of 10).

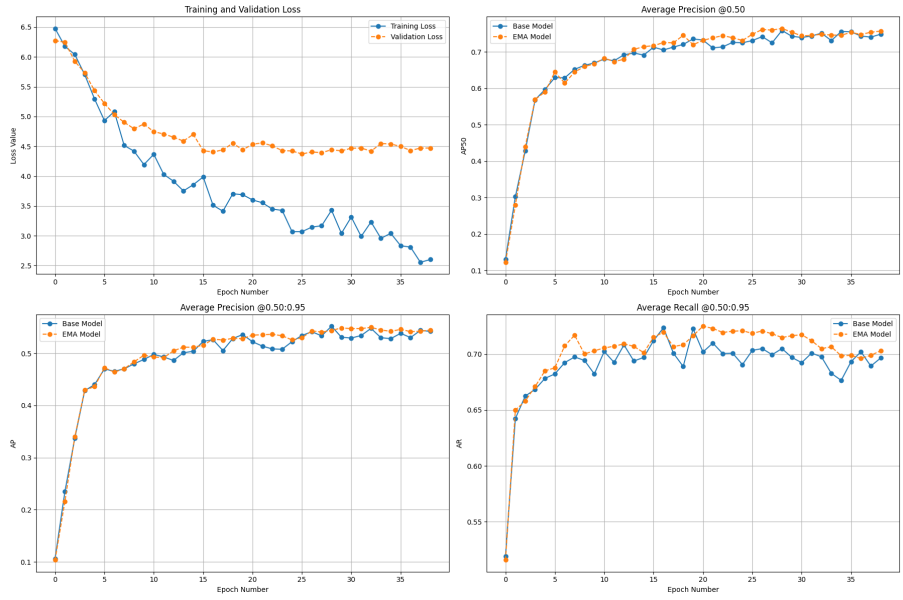


Figure 5.11: Comparison of Base and EMA model performance across Loss, Average Precision (AP), and Average Recall (AR) metrics.

Table 5.4 shows the final detection performance on the 122-image test set containing 149 traffic sign instances. Overall, the model achieved 60.4% mAP@50:95, which serves as the primary evaluation metric following COCO detcetion standards. With a 78.4% mAP at the 0.50 IoU

threshold, the model showcases effective localisation performance.

With a precision score of 81.5%, the model correctly identifies four out of five predictions correctly. This high precision indicates that the model is reliable when it makes predictions. However, recall measured 67.0%, meaning that the model detected only two-thirds of the traffic signs in the test set. This precision-recall trade-off reflects a conservative detection strategy where the model prioritises accuracy over coverage. For traffic sign detection applications, high precision is preferable to high recall, as false positives (incorrect detections) could mislead drivers or autonomous systems more severely than missed detections.

Performance varies across the six traffic sign classes. Blind-Spot Mirror achieved the highest mAP@50:95 of 76.9%, followed by No Through Road at 68.6%. These classes also have perfect precision, meaning the model never incorrectly predicted these sign types when they were absent. On the other end, Roundabout Ahead struggled with only 50.4% mAP@50:95, making it the most challenging class.

While the overall recall measured 67.0%, the identical values across all classes in Table 5.4 reflect the model's performance at a fixed confidence threshold determined during the final test evaluation. This indicates a consistent baseline detection capability, even though individual class sensitivity likely varies at different threshold points.

Table 5.4: Per-class detection performance for RF-DETR Nano on test set

Class	**Images**	**Instances**	**P**	**R**	**mAP@0.5**	**mAP@0.5:0.95**
Overall	122	149	0.815	0.670	0.784	0.604
Blind-Spot Mirror	21	21	1.000	0.670	0.905	0.769
No Through Road	14	14	1.000	0.670	0.853	0.686
No Entry	24	26	0.818	0.670	0.790	0.575
Stop	31	31	0.583	0.670	0.716	0.555
Pedestrian Crossing	30	34	0.793	0.670	0.726	0.535
Roundabout Ahead	19	23	0.696	0.670	0.716	0.504

Figure 5.12 displays the confusion matrix showing prediction outcomes. The diagonal values indicate correct classifications, with most confusion occurring in the rightmost False Negative Column and bottom False Positive row rather than between classes.

The model produced minimal confusion between classes. For example, only 11 instances were misclassified as the wrong sign type across the entire test set. The primary failure was detecting the signs themselves rather than classifications since when signs were detected, they were almost always classified correctly.

There were 37 False Negatives (missed detections) and 36 False Positives (incorrect detections) recorded in the matrix. It is important to note that this confusion matrix was generated using a raw confidence threshold of 0.01 to visualise the model's full range of detection tendencies. This low threshold explains the high count of false positives compared to the finalised metrics in Table 5.4, which utilise an optimised, more conservative threshold. Under these raw conditions, the 'No Entry' class generated the most hallucinations, likely due to the model's sensitivity to red circular objects in complex road scenes. However, at the optimised threshold used for the final results, many of these low-confidence detections are filtered out, leaving the 'Stop' sign as the class with the most significant impact on precision.

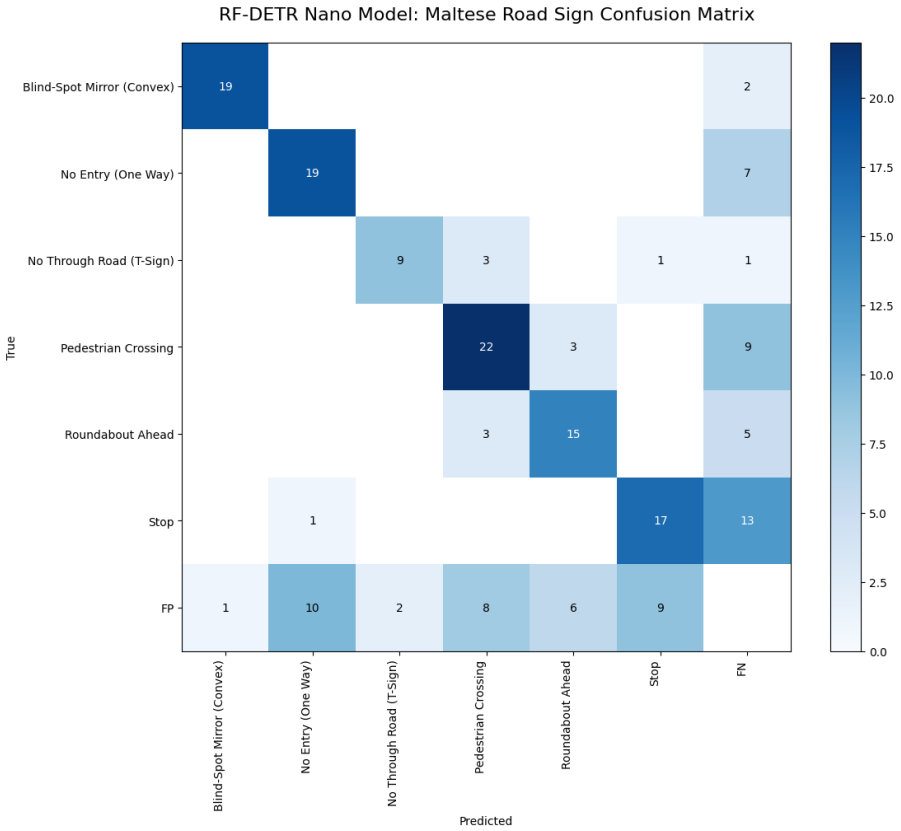


Figure 5.12: Confusion Matrix on the Test Set Using RF-DETR

The overall performance of the RF-DETR Nano is heavily influenced by its input resolution of 384×384. Evaluation logs show a significant disparity in performance based on object scale: the model achieved a high Average Recall (AR) of 80.3% for large objects, but dropped to 19.3% for medium objects and failed entirely (0.0%) for small objects. This suggests that the 384×384 resolution is sufficient for detecting traffic signs that occupy a large portion of the frame but lacks the pixel density required to preserve features for distant or smaller signs. Figure 5.13 illustrates this limitation, where the model successfully detects a nearby 'Pedestrian Crossing' sign but fails to localise smaller signs further down the road. Future improvements could involve increasing the input resolution or utilising a larger RF-DETR variant with more parameters to handle this limitation.

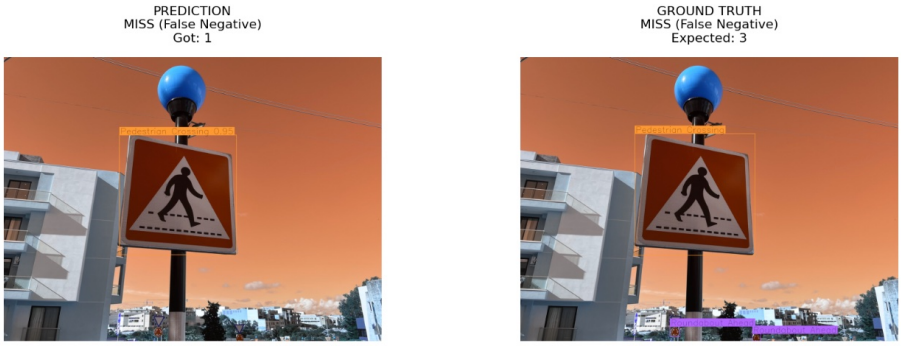


Figure 5.13: Example of missed sign detections further down the road

5.2 Matthew: Viewing Angle Results

The YOLOv12 Nano model was trained locally for 100 epochs with early stopping (patience=20). Training converged by epoch 40, with validation metrics stabilising at the final performance levels. The model was evaluated on a 122-image test set, containing 149 traffic sign instances labelled with viewing angles (Back, Front, or Side).

Table 5.5 presents the detection performance for viewing angle classification. The model achieved 72.0% mAP@50:95, significantly outperforming the RF-DETR traffic sign detector. This improvement is mostly due to the simpler nature of the task, since here, only three viewing angles are being classified instead of six different sign types. At the more tolerant IoU threshold of 0.50, the model achieved 88.7% mAP. Furthermore, the model achieved 91.7% precision and a recall of 83.2%.

Table 5.5: Per-class detection performance for YOLOv12 Nano on test set (viewing angle)

Class	**Images**	**Instances**	**P**	**R**	**mAP@0.5**	**mAP@0.5:0.95**
Overall	122	149	0.917	0.832	0.887	0.720
Back	34	37	0.891	0.946	0.959	0.799
Front	55	61	0.886	0.836	0.884	0.762
Side	47	51	0.973	0.714	0.817	0.597

There was some variation in performance across the three viewing angles. Back view achieved the highest mAP@50:95 of 79.9% with a recall of 94.6%, making it the easiest angle to detect. This is likely because back-facing signs have a uniform appearance across different sign types so factors such as colour did not matter as much. Front view performed moderately well with 76.2% mAP@50:95 and balanced precision (88.6%) and recall (83.6%). Side view showed the weakest performance with 59.7% mAP@50:95 and the lowest recall (71.4%). However, it achieved the highest precision (97.3%), meaning that when the model predicted a side view, it was almost always correct. The lower recall suggests side views are harder to detect, likely because signs photographed from the side present less distinctive visual features and are often in the background of other centralised signs.

Figure 5.14 displays the confusion matrix showing prediction outcomes. The strong diagonal values indicate correct classifications, with minimal confusion between viewing angles. Only 6 instances were misclassified as the wrong viewing angle across the entire test set, mainly between Front and Side. Similar to RF-DETR, the primary errors were false negatives (23 missed detections) and false positives (10 incorrect detections) rather than confusion between angle types.

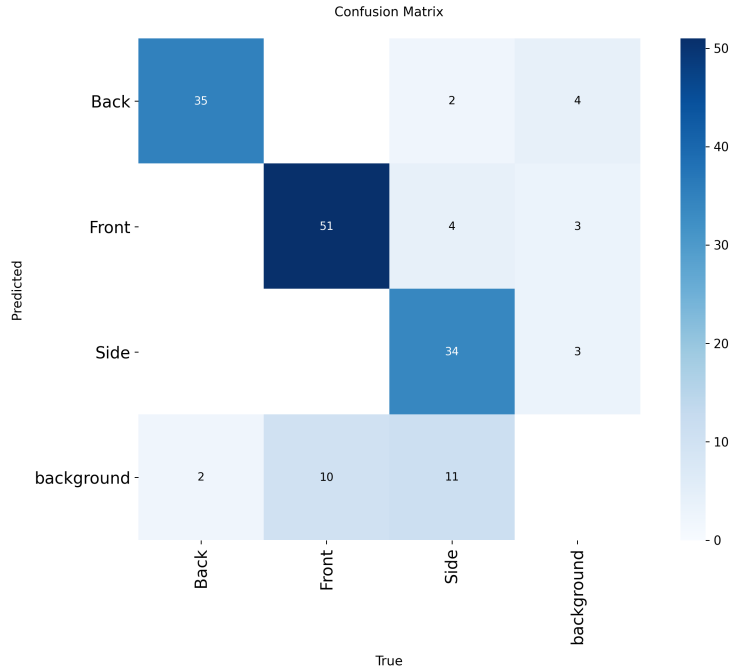


Figure 5.14: Confusion Matrix on the Viewing Angle Test Set Using YOLOv12n

5.3 Liam Jake: Sign Shape Results

The Faster R-CNN detector demonstrated effective convergence throughout the training cycle. The total loss exhibited a steep decline from an initial 34.187 at iteration 19 to 0.224 by iteration 1979.

Specific loss components highlight the model's learning trajectory:

- **Classification Accuracy:** The `fast_rcnn/cls_accuracy` remained high throughout, consistently staying above **98%** across most iterations.
- **Localization Loss:** The `loss_box_reg` decreased from **0.468** at the start to **0.064** at the end of training, indicating significant improvement in bounding box precision.
- **RPN Performance:** The Region Proposal Network (RPN) effectively learned to propose potential sign locations, with `loss_rpn_cls` dropping from **1.213** to **0.067**.

Individual shape performance varied:

- **Octagonal signs** (primarily Stop signs) showed the highest performance with an AP of **0.418**.
- **Circular signs** reached an AP of **0.123**.
- **Square signs** achieved an AP of **0.073**.
- **Triangular signs** remained the most difficult to classify, consistently reporting an AP of **0.0** in the sampled evaluation periods, likely due to their lower frequency in the dataset.

The results indicate that while the Faster R-CNN is a robust detector for general sign presence, specific shape classification accuracy depends heavily on the geometry and frequency of the class, with distinctive shapes like octagons being identified most reliably.

5.4 Daniel: Mounting Type Results

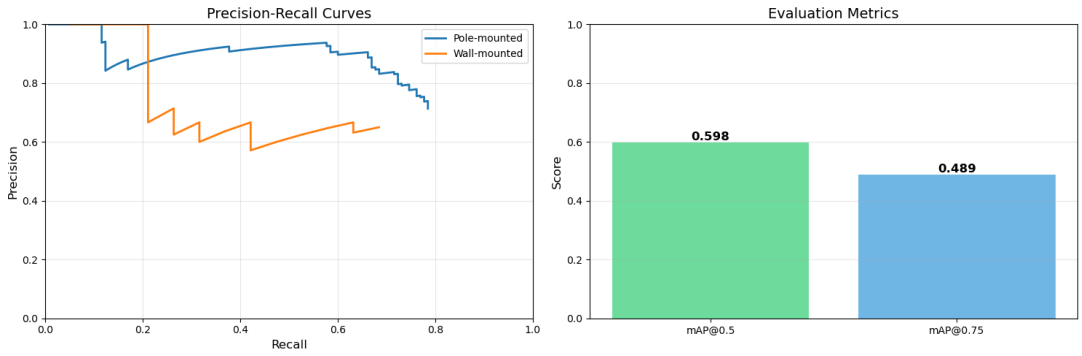


Figure 5.15: FCOS Precision–Recall Curve and mAP Score

The FCOS model shows much stronger results for pole-mounted signs than wall-mounted. This is due to the fact that there is a heavy class imbalance and even with oversampling, this effect cannot be diminished entirely. In addition, there was certain mislabelling in the data which affects the model's performance. In the precision–recall curves, the pole-mounted class maintains high precision across a broad recall range, indicating that the model is both confident and consistent when detecting this category. An *mAP@0.5* of 0.598 suggests reasonable detection quality at a standard IoU threshold, while the drop to 0.489 at *mAP@0.75* highlights some sensitivity to stricter localization requirements. This gap indicates that while the model often finds the correct object and class, bounding box alignment can still be improved.

Figure 5.16: Example of incorrect labelling

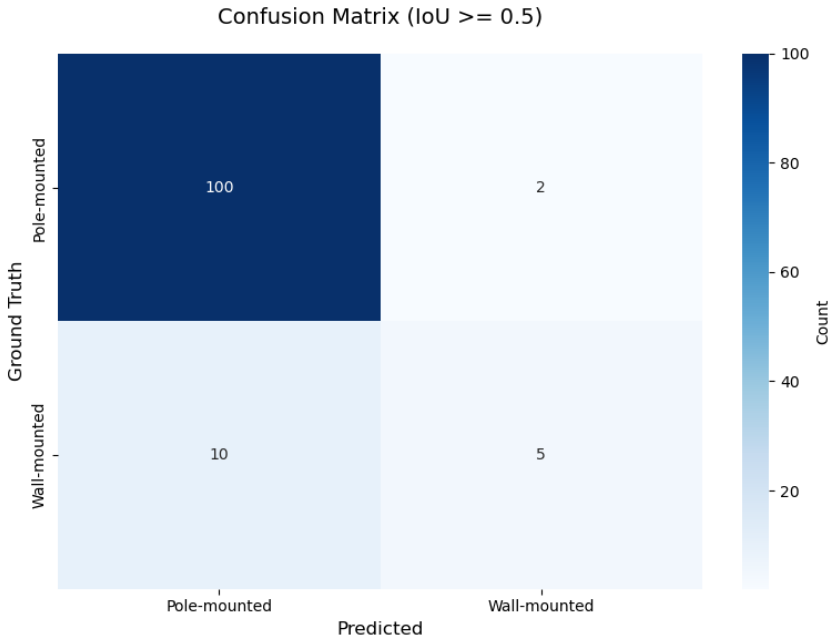


Figure 5.17: Confusion matrix for FCOS mounting-type classification

The confusion matrix further clarifies the model's behaviour. Pole-mounted signs are classified very accurately, with 100 true positives and only 2 misclassifications, demonstrating strong class separability and robust feature learning for this mounting type. In contrast, wall-mounted signs are more challenging: a noticeable portion is misclassified as pole-mounted, and only 5 are correctly identified. Qualitative examples reinforce this pattern, showing that wall-mounted signs are often detected but confused when visual context such as building edges or proximity to poles resembles pole-mounted setups. Overall, the model is reliable for pole-mounted detection, but would benefit from additional data diversity or targeted augmentation to better distinguish wall-mounted signs and improve bounding box localisation.

5.5 Liam Azzopardi: Sign Condition Results

To detect sign conditions, a confidence threshold of 0.4 was used as it balanced to filter predicted bounding boxes. This confidence threshold was selected as it provided a balance between detection sensitivity and false positive suppression; detecting less traffic signage but also less background elements. When using lower threshold values, it was evident that there were excessive false detections on background elements such as walls, buildings, and non-traffic signage, significantly degrading precision significantly despite higher recall.

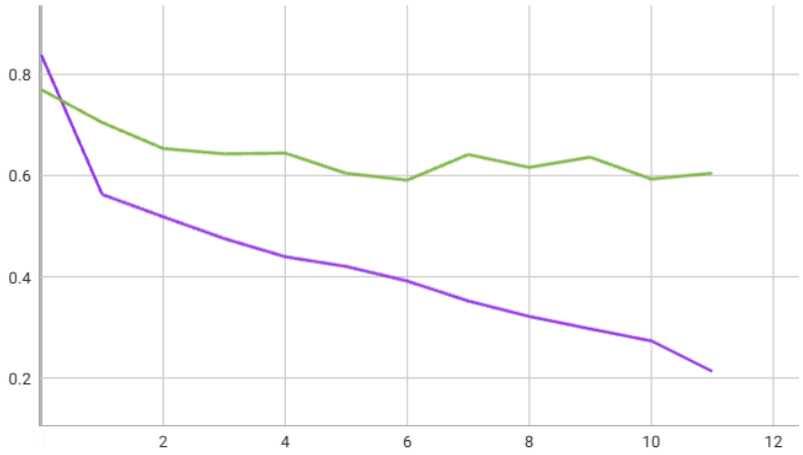


Figure 5.18: Total Loss - Train vs Validation Loss

RetinaNet managed to finish its training after finishing epoch 12, with epoch 7 having the least validation loss and saving the best model up to that point. Looking at the total loss between train and validation loss in figure 5.18, the green line reflects the validation loss across 12 epochs, while the purple line reflects the training loss across 12 epochs. The y-axis reflects the loss value while the x-axis reflects the number of epochs (step 0 = epoch 1). It is clear to see that there is significant overfitting which happens during the first epochs. We also see the point where the model reaches the best validation loss seen in these 12 epochs, seen in step 6 (epoch 7).

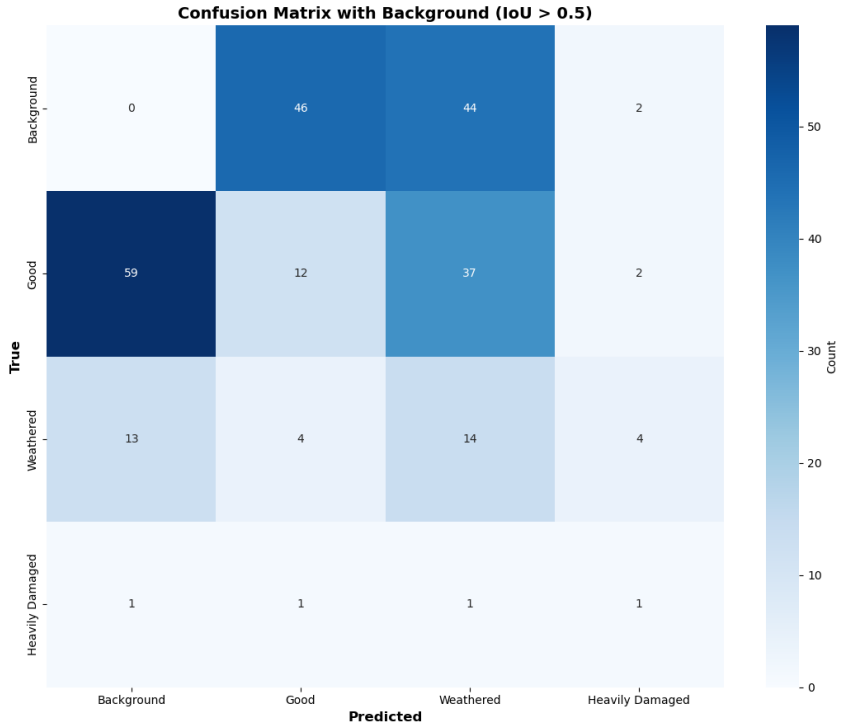


Figure 5.19: Confusion Matrix for Sign Conditions, Predictions with confidence scores below 0.4 were filtered out, and spatial matches required IoU $\geq$ 0.5.

Table 5.6: RetinaNet Performance Metrics

Class	Precision	Recall	F1-Score	Avg IoU	TP	FP	FN
Good	0.1905	0.1091	0.1387	0.8441	12	51	98
Weathered	0.1458	0.4000	0.2137	0.9179	14	82	21
Heavily Damaged	0.1111	0.2500	0.1538	0.9486	1	8	3
Overall	**0.1607**	**0.1812**	**0.1703**	–	**27**	**141**	**122**

The confusion matrix in figure 5.19 reveals several patterns. Despite using a confidence threshold of 0.4, there are still a large amount of signs that are not classified and are confused with background elements or different signage, especially with the classes "Good" and "Weathered". Despite the poor classification, the model achieved surprisingly high Average IoU values between 0.84 and 0.95, with the lowest-performing 'Heavily Damaged' class showing the best localisation at 0.9486. The high IoU across all classes confirms that spatial localisation is not the limiting factor. The failure lies in the classification head's ability to distinguish between sign conditions given severe class imbalance and the model's tendency toward cross-class confusion. The confusion matrix reveals that there is also common confusion between instances of "Good" and "Weathered" where 37 instances with a "Good" ground truth where misclassified as "Weathered". The 0.4 confidence threshold, while filtering some low-confidence predictions, proved insufficient to address these fundamental classification failures. The overall F1-score of 0.1703 with 141 false positives versus only 27 true positives demonstrates that RetinaNet, although it handles class imbalance through Focal Loss, RetinaNet requires substantially more balanced training data or alternative approaches to achieve reliable condition classification performance.

Figure 5.20: Sign Condition Misclassification

.

Bibliography

[1] Ultralytics, *Explore Ultralytics YOLOv8*, en. Accessed: Jan. 31, 2026. [Online]. Available: https://docs.ultralytics.com/models/yolov8/.

[2] Ultralytics, *YOLOv8 vs. YOLOv5: A Comprehensive Technical Comparison*, en. Accessed: Jan. 31, 2026. [Online]. Available: https://docs.ultralytics.com/compare/yolov8-vs-yolov5/.

[3] R. Khanam and M. Hussain, "Yolov11: An overview of the key architectural enhancements," *arXiv preprint arXiv:2410.17725*, 2024, arXiv:2410.17725 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2410.17725.

[4] Y. Tian, Q. Ye, and D. Doermann, *YOLOv12: Attention-Centric Real-Time Object Detectors*, Feb. 2025. DOI: 10.48550/arXiv.2502.12524. arXiv: 2502.12524 [cs]. Accessed: Jan. 29, 2026.

[5] I. Robinson, P. Robicheaux, M. Popov, D. Ramanan, and N. Peri, *RF-DETR: Neural Architecture Search for Real-Time Detection Transformers*, Nov. 2025. DOI: 10.48550/arXiv.2511.09554. arXiv: 2511.09554 [cs]. Accessed: Jan. 29, 2026.

[6] Q. Chen et al., *LW-DETR: A Transformer Replacement to YOLO for Real-Time Detection*, Jun. 2024. DOI: 10.48550/arXiv.2406.03459. arXiv: 2406.03459 [cs]. Accessed: Jan. 29, 2026.

[7] S. Ren, K. He, R. Girshick, and J. Sun, *Faster r-cnn: Towards real-time object detection with region proposal networks*, 2016. arXiv: 1506.01497 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1506.01497.

[8] R. Girshick, *Fast r-cnn*, 2015. arXiv: 1504.08083 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1504.08083.

[9] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, *Focal Loss for Dense Object Detection*, arXiv:1708.02002 [cs], Feb. 2018. DOI: 10.48550/arXiv.1708.02002. Accessed: Jan. 31, 2026. [Online]. Available: http://arxiv.org/abs/1708.02002.

[10] Z. Tian, C. Shen, H. Chen, and T. He, "Fcos: Fully convolutional one-stage object detection," *arXiv preprint arXiv:1904.01355*, 2019, arXiv:1904.01355 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1904.01355.

[11] *Train an RF-DETR Model - RF-DETR*, https://rfdetr.roboflow.com/learn/train/. Accessed: Jan. 29, 2026.

[12] *Google Colab*, https://colab.research.google.com/github/roboflow-ai/notebooks/blob/main/notebooks/ho to-finetune-rf-detr-on-detection-dataset.ipynb. Accessed: Jan. 29, 2026.

Generative AI Usage Journal - ARI3129

Daniel Simon Galea, Liam Azzopardi, Liam Jake Vella, Matthew Privitera

January 31, 2026

Contents

1 Introduction

This project made use of several generative AI models, namely ChatGPT 5.2, Google Gemini 3, Claude Sonnet 4.5 and NotebookLM. These models were selected due to their reliability, robustness, and versatility across a wide range of tasks relevant to the project. Claude Sonnet 4.5 was used during the programming and development stages as it is particularly-suited for coding-related tasks such as debugging, code optimisation and explaining and outputting complex programming concepts. The ability to generate context-aware code makes it a valuable tool especially in the technical side of this project.

Google Gemini and ChatGPT 5.2 were used more broadly throughout the project. Google Gemini 3 was used due to the fact that the pro version is offered for free to students by the University of Malta directly making it a readily available powerful tool. ChatGPT proved useful due to its flexibility and ability to be used for many different tasks.

The use of generative AI models allowed for an approach where each model could be used in a way that would exploit it's strength and would overall increases the efficiency and quality of the project.

2 Ethical Considerations

Although generative AI tools were used to assist us in this project, their use raises important ethical considerations that must be addressed regarding data bias, privacy, academic integrity and transparency.

2.1 Data Bias and Fairness

When using GenAI tools to help us analyse experimental results, there is a risk that the AI's interpretations may reflect biases present in its training data. For example, when asked to suggest which important plots should be included in the evaluation section, the AI initially recommended a comprehensive six-plot layout typical for research papers. However, this recommendation may have reflected a bias toward more complex projects that would have been inappropriate for the scope of this one.

2.2 Privacy and Data Protection

The use of GenAI tools requires careful consideration of what information is shared. While data such as training logs and evaluation data is not personally sensitive, it is important to remember that GenAI tools retain conversation history, meaning project details may persist in external systems. To address these concerns, no personally identifiable information beyond standard result metrics were shared. The traffic sign images themselves were never uploaded in raw form. Furthermore, when seeking help with code, complete datasets were not provided, but instead only a small sample was given so that we can have a better understanding of how to evaluate the results.

2.3 Academic Integrity

The most significant ethical consideration involves maintaining academic integrity when AI tools are able to assist with technical writing and analysis. In this project, GenAI tools were used as an assistive tool rather than a content generator. Specifically, they were used to help improve text clarity, debug code errors, explain technical concepts and suggest plot layouts. However, all AI generated text was critically reviewed, edited, and adapted to reflect independent understanding of the project. Apart from that, AI provided explanations of concepts or models were cross-referenced with official documentation and research papers. AI suggestions were treated as a starting point for exploration rather than final answers. This was done to make sure that the final work reflects genuine understanding rather than unverified AI output.

2.4 Transparency and Accountability

Transparency and accountability of AI usage will be highlighted across the next sections, where we will describe the methodology, prompts and responses, improvements, errors and contributions made, and an individual reflection for each group member.

3 Methodology

Throughout the assignment, different generative AI models were used as collaborative tools to provide assistance throughout the different stages of the project, including code debugging, understanding concepts and model architectures and implementation guidance.

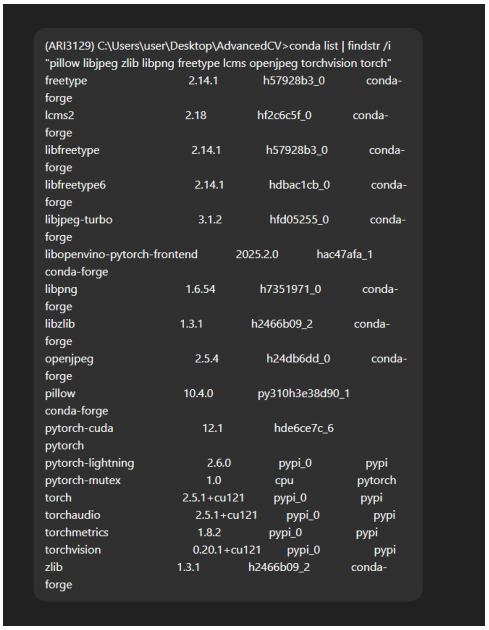
3.1 Environment Setup

Claude Sonnet 4.5 and ChatGPT-5.2 were particularly used to aid in the environment setup and help manage any version dependencies for the different python libraries that were needed to train the different models. Since there were some issues with the initial environment setup, Claude Sonnet 4.5 provided an identification of any required libraries and their appropriate versions to replace any library installation errors in the original *Dependencies.yml*. ChatGPT-5.2 helped in troubleshooting installation errors related to CUDA and Claude Sonnet 4.5 provided a new *Dependencies.yml* that successfully helped us setup the conda environment.

3.2 Implementation Guidance

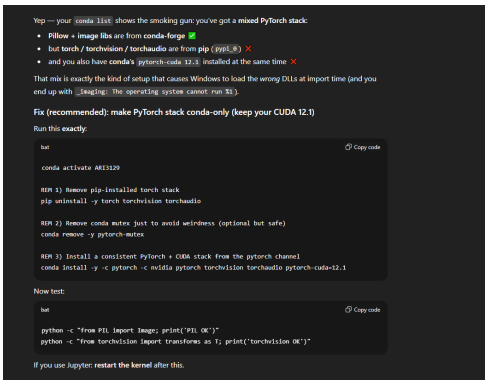
For the implementation process, Claude Sonnet 4.5 was selected based on its specific strengths to optimise different aspects of the code and to understand the effect different hyperparameter configurations have on the overall performance of the different models. Claude Sonnet 4.5 suggested the removal and addition of some hyperparameters, suggesting the inclusion of *warmup_epoch* and using *optimizer=auto* for YOLOv8, for example. As part of the implementation, Claude Sonnet 4.5 and NotebookLM helped us understanding the architectural concepts of each model when needed for example Focal Loss in RetinaNet and the ViT backbone architecture in RF-DETR. Specifically, NotebookLM was used by uploading research papers of different models to summarise different concepts and different architectures used in the assignment.

4 Prompts and Responses



```
(ARI3129) C:\Users\user\Desktop\AdvancedCV>conda list | findstr /i
"pillow libjpeg zlib libpng freetype lcms openjpeg torchvision torch"
freetype                  2.14.1          h57928b3_0          conda-
forge
lcms2                     2.18            hf2c6c5f_0          conda-
forge
libfreetype               2.14.1          h57928b3_0          conda-
forge
libfreetype6              2.14.1          hdbac1cb_0          conda-
forge
libjpeg-turbo             3.1.2           hfd05255_0          conda-
forge
libopenvino-pytorch-frontend  2025.2.0        hac47afa_1
conda-forge
libpng                    1.6.54          h7351971_0          conda-
forge
libzlib                   1.3.1           h2466b09_2          conda-
forge
openjpeg                  2.5.4           h24db6dd_0          conda-
forge
pillow                    10.4.0          py310h3e38d90_1
conda-forge
pytorch-cuda              12.1            hde6ce7c_6
pytorch
pytorch-lightning         2.6.0           pypi_0              pypi
pytorch-mutex             1.0             cpu                 pytorch
torch                     2.5.1+cu121     pypi_0              pypi
torchaudio                2.5.1+cu121     pypi_0              pypi
torchmetrics              1.8.2           pypi_0              pypi
torchvision               0.20.1+cu121    pypi_0              pypi
zlib                      1.3.1           h2466b09_2          conda-
forge
```

Figure 1: Example of a Generative AI Prompt



Yep — your `conda list` shows the smoking gun: you've got a **mixed PyTorch stack**:

- **Pillow + image libs** are from **conda-forge** ✅
- but **torch / torchvision / torchaudio** are from **pip** (`pypi_0`) ❌
- and you also have **conda's** `pytorch-cuda 12.1` installed at the same time ❌

That mix is exactly the kind of setup that causes Windows to load the *wrong* DLLs at import time (and you end up with `_imaging: The operating system cannot run %1`).

Fix (recommended): make PyTorch stack conda-only (keep your CUDA 12.1)

Run this **exactly**:

```bat
conda activate ARI3129

REM 1) Remove pip-installed torch stack
pip uninstall -y torch torchvision torchaudio

REM 2) Remove conda mutex just to avoid weirdness (optional but safe)
conda remove -y pytorch-mutex

REM 3) Install a consistent PyTorch + CUDA stack from the pytorch channel
conda install -y -c pytorch -c nvidia pytorch torchvision torchaudio pytorch-cuda=12.1
```

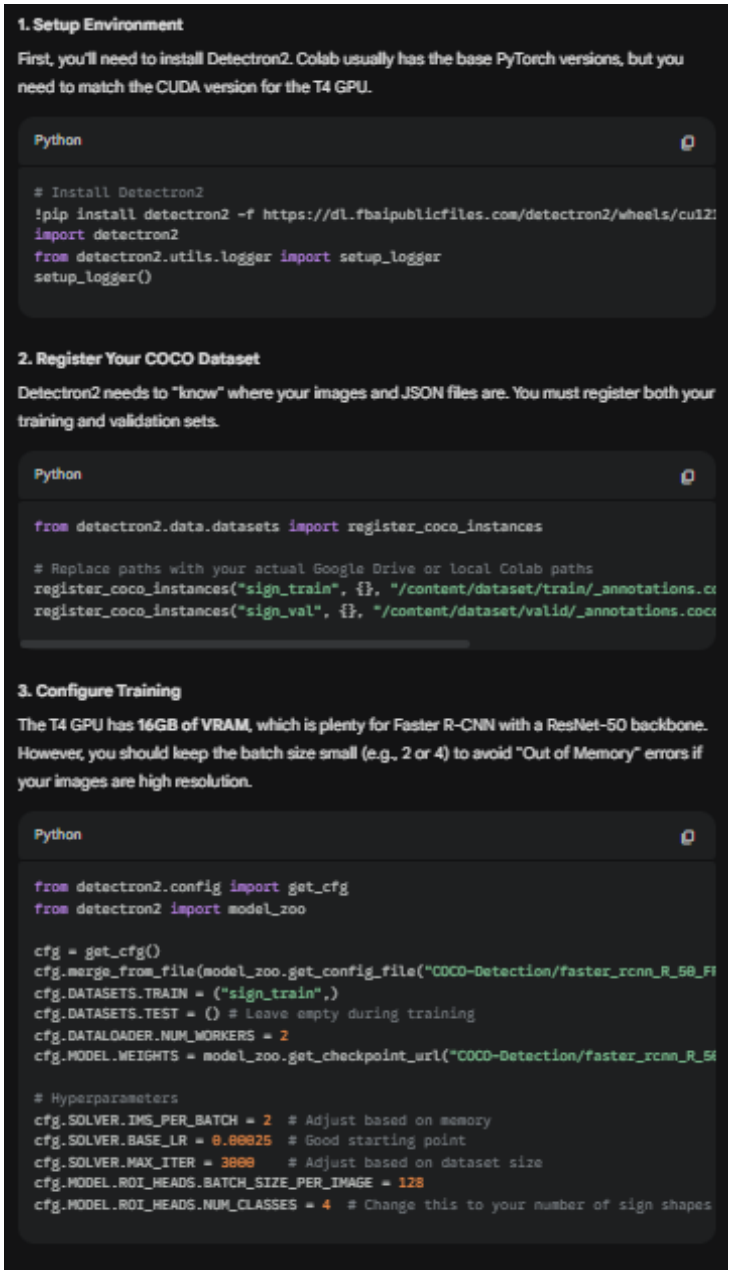
Now test:

```bat
python -c "from PIL import Image; print('PIL OK')"
python -c "from torchvision import transforms as T; print('torchvision OK')"
```

If you use Jupyter: **restart the kernel** after this.

Figure 2: Generative AI Response

In this prompt, ChatGPT helped me fix and restore my environment as my device had a windows update which in turn broke certain libraries in the environment. By giving the entire list of libraries in the environment, ChatGPT was able to pinpoint the problem and resolve the issues that I had in the environment so that I could resume my work by removing the libraries which were now incompatible with the new update and install the working libraries in their place that worked.



1. Setup Environment

First, you'll need to install Detectron2. Colab usually has the base PyTorch versions, but you need to match the CUDA version for the T4 GPU.

Python

```
# Install Detectron2
!pip install detectron2 -f https://dl.fbaipublicfiles.com/detectron2/wheels/cu12
import detectron2
from detectron2.utils.logger import setup_logger
setup_logger()
```

2. Register Your COCO Dataset

Detectron2 needs to "know" where your images and JSON files are. You must register both your training and validation sets.

Python

```
from detectron2.data.datasets import register_coco_instances

# Replace paths with your actual Google Drive or local Colab paths
register_coco_instances("sign_train", {}, "/content/dataset/train/_annotations.c
register_coco_instances("sign_val", {}, "/content/dataset/valid/_annotations.coc
```

3. Configure Training

The T4 GPU has **16GB of VRAM**, which is plenty for Faster R-CNN with a ResNet-50 backbone. However, you should keep the batch size small (e.g., 2 or 4) to avoid "Out of Memory" errors if your images are high resolution.

Python

```
from detectron2.config import get_cfg
from detectron2 import model_zoo

cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_R_50_FP
cfg.DATASETS.TRAIN = ("sign_train",)
cfg.DATASETS.TEST = () # Leave empty during training
cfg.DATALOADER.NUM_WORKERS = 2
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/faster_rcnn_R_50

# Hyperparameters
cfg.SOLVER.IMS_PER_BATCH = 2  # Adjust based on memory
cfg.SOLVER.BASE_LR = 0.00025  # Good starting point
cfg.SOLVER.MAX_ITER = 3000    # Adjust based on dataset size
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 128
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 4  # Change this to your number of sign shapes
```

Figure 3: Example of a Generative AI Prompt

5 Improvements, Errors, and Contributions

Generative AI tools significantly enhanced various aspects of the project, from idea generation to debugging, while also presenting challenges that required verification and correction. These tools contributed to overall efficiency but required careful management to mitigate inaccuracies.

In terms of improvements to quality and efficiency, generative AI excelled in code-related

tasks. For example, during model implementation, Claude was used to debug issues in object detection architectures, such as resolving incorrect configurations in training pipelines. By providing error logs and code snippets as prompts, the AI suggested optimisations, which reduced training time and improved model stability. Similarly, AI-assisted code generation for techniques like image rotations and mosaics, tailored to the dataset's real-world variations, leading to better generalisation across attributes like viewing angles or sign conditions.

Generative AI also helped in idea generation and literature review. For the background on object detection techniques, prompts were used to summarise architectural evolutions in models like the YOLO family, yielding overviews with key references that were adapted into the report. This streamlined research, saving time on sourcing and understanding information from recent publications.

However, errors and misleading information were introduced in some outputs. In one literature review instance, ChatGPT incorrectly described YOLOv11 as a community driven update. This mistake was identified by cross-checking with official documentation and corrected to ensure accuracy. Another error occurred in debugging suggestions, where incompatible configurations were recommended, leading to temporary performance drops in metrics. Validation runs and dataset-specific adjustments resolved this. In privacy discussions, AI sometimes exaggerated risks, requiring additional reviews that ultimately reinforced data protection practices but caused minor delays.

Overall, AI's contributions in areas like debugging and the literature review, boosted project quality, with errors addressed via human oversight, primary source verification and testing. This approach maximised benefits while maintaining reliability.

6 Individual Reflection

6.1 Daniel Simon Galea

From using these generative AI tools, I gained a deeper understanding of how different neural network architectures operate and how their respective hyperparameters influence model performance. Generative AI was particularly valuable in explaining neural network

architectures, allowing me to grasp underlying concepts more effectively.

Generative AI significantly improved my overall efficiency, as it could be applied across multiple stages of the project. Its ability to debug code, suggest improvements and clearly explain concepts related to the various architectures I implemented reduced development time and minimised trial-and-error. This allowed me to focus more on refining my models and improve performance than on resolving avoidable technical issues.

The task that benefited the most from AI assistance was hyperparameter tuning. Generative AI provided guidance on selecting appropriate parameter ranges and optimisation strategies, enabling us to fine-tune my models more effectively. As a result, model performance and accuracy were improved, contributing to stronger results and a higher-quality final project overall.

6.2 Liam Jake Vella

My perspective on using Generative AI in academic projects has evolved from viewing it as a simple shortcut to seeing it as a sophisticated collaborative tool. Gen AI helped streamline the setup of the Detectron2 framework for Faster R-CNN (Figure 3). Configuring the `register_coco_instances` and custom `cfg` parameters for the 5 specific classes was a part of the project where Gen AI was most helpful for rapid debugging and code refinement. I found that Gen AI saved me a significant amount of time rather than wasting it. Instead of spending hours troubleshooting, I could focus more on analysing results and fine-tuning hyperparameters.

In writing the report, Generative AI also helped me refine my language, making it easier to understand and straight to the point. As a group, we decided to use LaTeXto write it. As I have minimal experience in using LaTeX, I used Generative AI to help me format and use the appropriate functions. The knowledge that I gained from this will be very useful for future projects, such as my Final Year Project.

6.3 Matthew Privitera

Throughout this project, I primarily used Claude for technical explanations, data analysis and writing assistance. I also used NotebookLM to help me understand YOLOv12 and

RF-DETR architectures and how they improved on their predecessors. This experience provided valuable insights into both the benefits and limitations of AI tools in academic work.

GenAI tools improved efficiency in specific areas. Technical documentation that would normally take hours of reading was summarised into clear explanations in minutes. For example, understanding the RF-DETR architecture which involves concepts like DINOv2 backbones (LW-DETR improvement). This guided reading approach saved substantial time while ensuring I understood the core concepts. Code generation was another area where AI proved highly efficient. Tasks like parsing JSON files, creating LaTeX tables and generating visualisations were completed in minutes. The AI could produce working Python scripts that I could test and modify, which eliminated the tedious process of going through long documentation.

However, AI usage also created inefficiencies in some cases. The AI often provided overly comprehensive responses when concise answers would suffice. For instance when asking about what type of metrics/plots would be appropriate for this assignment, it suggested a six-plot layout with detailed implementation code. This required time to read through and evaluate, only to decide that a simpler two-figure approach was more appropriate and effective. Additionally, while Claude is a very powerful tool, clarifying questions sometimes required multiple rounds of back-and-forth because it would overcomplicate the topic.

Overall, the integration of GenAI tools like Claude and NotebookLM proved to be an important asset in managing the technical complexity of this project. While the tendency for these models to provide overly verbose outputs required a critical and selective approach, their ability to simplify complex architectures like RF-DETR and automate boilerplate coding tasks was invaluable. These tools served as a bridge between high-level theoretical concepts and practical implementation to accelerate the development cycle while allowing more time for developing the coding architecture and thoughtful analysis.

6.4 Liam Azzopardi

Throughout the assignment, I mainly used Claude Sonnet 4.5 to understand technical concepts related with YOLOv8 and RetinaNet, and get aid in setting up the conda environment setup, debugging code and getting guidance on the implementation process. Using Claude Sonnet 4.5 throughout the assignment led to getting a better technical understanding about each model and the general implementation pipeline needed to implement such models, but with some notable limitations.

When asking for particular model information, Claude provided some misinformation within its response, making it important to cross reference with the official Ultralytics for YOLOv8 [1] and the research paper for RetinaNet by Lin et al. [2]. Misinformation included incorrect release dates and general assumptions including architectural details.

The use of generative AI helped in the efficiency of the project, where writing about technical concepts and individual model details proved to be much more efficient. Generative AI also proved to be useful in efficient code debugging with loading the training set, test set, and validation set into each model and general hyper-parameter suggestions to optimise accuracy but also manage limited computational resources.

7 References and Resources Used

References

[1] Ultralytics, *Explore Ultralytics YOLOv8*, en. Accessed: Jan. 31, 2026. [Online]. Available: `https://docs.ultralytics.com/models/yolov8/`.

[2] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, *Focal Loss for Dense Object Detection*, arXiv:1708.02002 [cs], Feb. 2018. DOI: `10.48550/arXiv.1708.02002`. Accessed: Jan. 31, 2026. [Online]. Available: `http://arxiv.org/abs/1708.02002`.