

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).

Andrea Filiberto Lucas

Student Name



Signature

Sean David Muscat

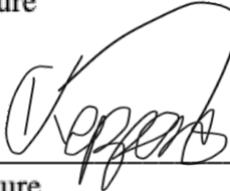
Student Name



Signature

Thomas Koppens

Student Name



Signature

ARI3129

Course Code

ARI3129 - Group Project 2024/25

Title of work submitted

27-01-2025

Date

ARI3129 - Group Project 2024/25

Documentation & GenAI Journal



**L-Università
ta' Malta**

Name: **Andrea F. Lucas, Thomas Koppens & Sean D. Muscat**
Course: ARI3129 - Artificial Intelligence (AI)
ID No: 0279704L, 029204L & 0172004L

Project's Distribution of Work

Section	Responsible Person(s)
README.md & Dataset Description	Andrea F. Lucas
Task 1: Building the Dataset	All
Task 1: Dataset Visualisation (<i>DatasetVisualisation_AFL.ipynb</i>)	Andrea F. Lucas
Task 2: Object Detection: YOLOv8 (<i>Yolov8_AFL.ipynb</i>)	Andrea F. Lucas
Task 2: Object Detection: YOLOv5 (<i>Yolov5_SDM.ipynb</i>)	Sean D. Muscat
Task 2: Object Detection: RetinaNet (<i>RetinaNet_TK.ipynb</i>)	Thomas Koppens
Introduction	Sean D. Muscat
Background: YOLOv8	Andrea F. Lucas
Background: YOLOv5	Sean D. Muscat
Background: RetinaNet	Thomas Koppens
Data Preparation	Andrea F. Lucas
Implementation: YOLOv8	Andrea F. Lucas
Implementation: YOLOv5	Sean D. Muscat
Implementation: RetinaNet	Thomas Koppens
Evaluation: YOLOv8	Andrea F. Lucas
Evaluation: YOLOv5	Sean D. Muscat
Evaluation: RetinaNet	Thomas Koppens
Concluding Remarks	Andrea F. Lucas

Appendix Generative AI Journal - Distribution of Work

Section	Responsible Person(s)
Introduction	Thomas Koppens
Ethical Considerations	Sean D. Muscat
Methodology	Andrea F. Lucas
Prompts & Responses	All
Improvements, Errors and Contributions of GenAI	Thomas Koppens
GenAI Individual Reflection	All

Use of Acronyms for Contributor Identification

Throughout the entire project, we used acronyms to clearly identify contributions made by each team member. These acronyms include **AFL** for Andrea Filiberto Lucas, **SDM** for Sean David Muscat, and **TK** for Thomas Koppens. By attributing tasks and sections of the project to specific individuals, we ensured transparency and accountability in our collaborative efforts.

Dataset GitHub Repository & Roboflow

The MDD (Maltese Domestic Dataset) is accessible via the following GitHub link: GitHub Repository (<https://github.com/AFLucas-UOM/ARI3129-MDD>). The dataset is organized within the **Dataset folder**, which contains three subfolders named after the individuals responsible for taking the photos, as well as a subfolder for online-sourced images. The subfolders for Andrea Filiberto Lucas and Sean David Muscat include an extra subfolder, containing additional photos taken for more balanced splits between the four classes. Note that no augmented photos are available in the GitHub repository; however, these are accessible via the following Roboflow link: MDD on Roboflow (<https://universe.roboflow.com/advanced-cv/maltese-domestic-dataset>).

1 Introduction

The use of computer vision (CV) to address real-world challenges has become increasingly prominent, offering innovative ways to analyse and interpret visual data. This project focuses on the application of CV techniques to the specific context of waste bag classification in Maltese streets, specifically **mixed waste bags, recycling bags, organic waste bags and other waste**, such as containers with glass bottles. By exploring methods for recognising and classifying Maltese domestic waste bags, the project investigates how unstructured visual information can be systematically captured, processed, and analysed. The emphasis on Maltese domestic waste as a focal point provides a clear case study for applying these techniques in a controlled yet challenging environment.

The need to analyse Maltese domestic waste collection visually stems from the complexity of urban environments, where the accurate identification of objects can be hindered by variations in lighting, background, and object placement. This project highlights the intricacies of handling such data, from capturing images that meet specific criteria to ensuring annotations are both consistent and meaningful. Furthermore, the use of CV models, namely: YOLOv8, YOLOv5 and RetinaNet, for this task underscores the importance of robust algorithms capable of handling diverse input scenarios. By creating a structured dataset of Maltese domestic waste bag images (**MDD - Maltese Domestic Dataset**), the project lays the groundwork for the development and evaluation of detection models.

In addition to addressing the technical challenges of capturing and annotating images, this work delves into the broader implications of applying CV to tasks that require high precision. The focus on Maltese domestic waste categories, reflects the necessity for detailed categorisation and highlights the versatility of modern object detection techniques.

2 Backgrounds

In the next section, the background of each chosen model will be discussed by the respective team members, with insights provided into its origins, features, and relevance to the project.

2.1 Background: YOLOv8

YOLOv8 is one of the latest iterations in the You Only Look Once (YOLO) series of object detection models, known for its speed, efficiency, and versatility in computer vision tasks [1]. It builds upon a rich history of YOLO models, starting from YOLOv1, introduced in 2015 by Joseph Redmon as part of his doctoral research at the University of Washington. YOLOv1 framed object detection as a single regression problem, predicting bounding boxes and class probabilities simultaneously in a single pass through a convolutional neural network [1]. This foundational model revolutionized object detection by prioritizing real-time performance, inspiring the development of subsequent versions.

YOLOv8, released by Ultralytics in January 2023, represents a state-of-the-art advancement in real-time object detection. Unlike its predecessors, YOLOv8 introduces an anchor-free detection mechanism, simplifying the prediction process by directly identifying object centers instead of relying on predefined anchor boxes. This design reduces the computational overhead during post-processing, making YOLOv8 faster and more efficient [1, 9].

The model also features a new and improved backbone network based on CSPDarknet53, which enhances feature extraction while maintaining high computational efficiency. YOLOv8 incorporates advanced techniques such as cross-stage partial (CSP) connections and a C2f module to improve the network's ability to capture complex visual patterns [5, 9]. These architectural enhancements enable YOLOv8 to excel in detecting small, occluded, and tilted objects, which are common challenges in many real-world scenarios [7, 3].

Different scaled versions of YOLOv8 are available, including YOLOv8n (nano), YOLOv8s (small), YOLOv8m (medium), YOLOv8l (large), and YOLOv8x (extra-large), providing flexibility to balance accuracy and computational cost. These variants make YOLOv8 suitable for deployment on both low-power devices and high-performance systems [9].

2.1.1 YOLOv8 for Waste Classification and Detection

YOLOv8 has proven to be an effective tool for waste classification and detection, offering solutions to the inefficiencies of manual sorting methods. The model can classify waste into categories such as plastic, paper, glass, and metal, automating the sorting process and reducing human error [2, 4, 5]. Custom datasets, tailored for specific waste types, are used to train YOLOv8 models. These datasets often include images captured in diverse environmental settings to improve the model's generalization capabilities [6].

Data augmentation techniques, are commonly employed to address limitations in dataset size and to improve robustness against variations in lighting and object appearance [6]. YOLOv8's anchor-free design enhances its performance in detecting small or irregularly shaped waste objects, which are often challenging for traditional detection models [7].

In practical applications, YOLOv8 can be deployed in smart waste bins or unmanned vehicles equipped with cameras for real-time waste detection and classification. For instance, one study demonstrated that YOLOv8 outperformed earlier YOLO versions and SSD algorithms in detecting small, tilted, and occluded objects, making it a great choice for waste management systems [8, 3].

Despite its strengths, challenges remain in waste classification tasks. Overlapping objects, diverse waste appearances, and ambiguous categories can impact detection accuracy. YOLOv8, however, mitigates these issues through its advanced feature extraction techniques and ability to handle complex object patterns [5, 9].

2.2 Background: YOLOv5

The real-time object detection algorithm YOLOv5 is renowned for its accuracy and quickness. In 2015, Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi developed the YOLO algorithm, which makes it its fifth iteration [10]. The YOLO algorithm breaks down an image into regions and predicts bounding boxes and probabilities for each region using a single neural network [10, 11]. Since 2015, the Ultralytics team has released multiple iterations of the model as they continue to refine it. A popular platform for moving YOLOv3 models from Darknet to PyTorch, YOLOv5 developed as a development of the YOLOv3 PyTorch implementation [12].

There are five sizes of YOLOv5: n represents nano, s represents small, m represents medium, l represents big, and x represents extra large. Although the number of layers and characteristics vary, these models employ the same techniques. The CSP-Darknet53 backbone, the SPP and PANet in the model neck, and the head utilised in YOLOv4 are the three primary parts of all YOLOv5 models [12]. In order to handle duplicate gradients, the CSP-Darknet53 convolutional network employs the Cross Stage Partial network technique [10]. While the model head applies anchor boxes to feature maps and renders the final output of classes, objectness scores, and bounding boxes, the model neck collects feature pyramids to aid the model in generalising to objects of various sizes.

YOLOv5 incorporates modifications to the model neck, including the use of a Spatial Pyramid Pooling (SPP) variation and the addition of BottleNeckCSP to the Path Aggregation Network (PANet). The SPP block expands the receptive field and separates pertinent context characteristics without slowing down the network by combining data from the inputs and returning a fixed-length output. Another variation of the SPP block that is utilised in YOLOv5 to increase network performance is SPPF. The PANet is a feature pyramid network designed to enhance pixel localisation and information flow. YOLOv5 makes use of sigmoid activation functions and SiLUs (Sigmoid Linear Units). The output layer's convolution operations use the Sigmoid activation function, while the hidden layers' convolution operations use the SiLU activation function [10].

YOLOv5 returns three outputs: the classes of the detected objects, their bounding boxes, and the objectness scores. It uses BCE (Binary Cross Entropy) to compute the classes loss and the objectness loss, and CIoU (Complete Intersection over Union) loss to compute the location loss. Other improvements in YOLOv5 include the Focus Layer, which replaced the first three layers of the network, eliminating grid sensitivity to improve bounding box detection, and implementation in PyTorch. Compared to YOLOv4, YOLOv5 is faster to train, lighter, and has a better inference time while maintaining the same mAP (mean Average Precision). Key changes in YOLOv5 that differentiate it from previous versions include applying the CSPNet to the Darknet53 backbone, integrating the Focus layer to the CSP-Darknet53 backbone, replacing the SPP block with the SPPF block in the model neck, and applying the CSPNet strategy on the PANet model [10].

YOLOv5 is a popular choice for constrained edge deployment scenarios, suitable for applications such as healthcare, security surveillance, and self-driving cars [12]. The transition from the Darknet framework to PyTorch has made it more accessible for custom object detection tasks.

2.3 Background: RetinaNet

RetinaNet [13] is a single-stage object detector developed by Lin et. al. At Facebook AI Research in 2018. During the time of its development, two-stage detectors like R-CNN [14] and SPP-NET [15] had superior accuracy but could not compete with the speed of single-stage detectors. The idea behind RetinaNet was to create a single-stage detector which could obtain results more similar to the multi-stage detectors in terms of accuracy.

In order to achieve those improved results, Lin et. al. developed a new loss function called Focal Loss. As the name implies, the focal loss function shifts the focus of the cross-entropy criterion on the more complex, difficult training instances. The focal loss may be tweaked by a variable γ called the focusing parameter. The greater the value of the focusing parameter, the more emphasis is put on the difficult training instances. Since learning is now focused on these most significant examples, the impact of class imbalance is reduced and the overall accuracy, regardless of the precise value of the focusing parameter, is significantly improved. In fact, the strategy of using focal loss outperforms other common strategies at the time, including hard example mining [16] and heuristic sampling [17], and even matches two-stage approaches on the COCO dataset [18] at the time.

RetinaNet also made use of recent advances in the single-stage detection sphere. It combines a Feature Pyramid Network (FPN) [19] with a ResNet backbone [20] to extract multi-scale features effectively. The FPN allows the model to process images at different resolutions, enhancing its ability to detect objects of varying sizes. Each level of the feature pyramid is connected to classification and regression subnets, which are tasked with predicting the class probabilities and bounding box coordinates for each anchor box. This hierarchical structure contributes to RetinaNet's ability to balance efficiency and accuracy.

They also made use of anchor boxes inspired by [19], which are predefined bounding boxes of different sizes and aspect ratios. These anchor boxes help the model detect objects of diverse shapes and scales in an image. By carefully tuning the anchor box configuration, RetinaNet ensures robust performance across a wide range of object detection tasks.

Overall, the RetinaNet architecture provided single-stage detectors with the jump in accuracy needed to compete with multi-stage detectors, while retaining the simplicity and short inference times of single-stage detectors. This combination is why it was chosen for this project. The idea was that, although it may not be able to run quite as quickly as the recent YOLO variants, it would be more accurate.

3 Data Preparation

This section outlines the dataset preparation process, starting with data gathering and its accessibility through Roboflow and GitHub. It then details preprocessing steps, including resizing and augmentation, to ensure compatibility with all three models used in the 4 section. Finally, dataset visualization and analytics, are discussed to validate and optimize the dataset for training and testing.

3.1 Data Gathering

The dataset was compiled using a **total of 356 images**. These were collected as follows: **AFL** gathered 139 images, **SDM** collected 117 images, and **TK** contributed 94 images. Additionally, 6 images were sourced online [21, 22, 23, 24, 25, 26] to complement the dataset.

All manually collected images, apart from the online-sourced ones, were captured using the following devices: iPhone 16 Pro, iPhone 13, One Plus Nord2 5G, Samsung Galaxy A25, Xiaomi Redmi Note 9 Pro, and Google Pixel 7a. To ensure privacy and compliance with data protection guidelines, any personal or identifiable information, such as faces or car number plates, were either avoided during collection or removed during preprocessing. This was achieved using tools such as Apple Intelligence's "Clean Up" tool [27], Microsoft's Photos App's "Generative Erase" [28] and Samsung's Photo Gallery's "Object Eraser" [29].

Efforts were made to balance the instances of the dataset's three main categories (mixed waste, recyclable material, and organic waste).

3.2 Dataset Accessibility & Annotation Process

All 356 images were uploaded to Roboflow and are publicly accessible via the following link: <https://universe.roboflow.com/advanced-cv/maltese-domestic-dataset>. Additionally, the dataset has been included in the "Dataset" folder of the project's GitHub repository: <https://github.com/AFLucas-UOM/ARI3129-MDD>. These steps resulted in the creation of the **MDD - Maltese Domestic Dataset**.

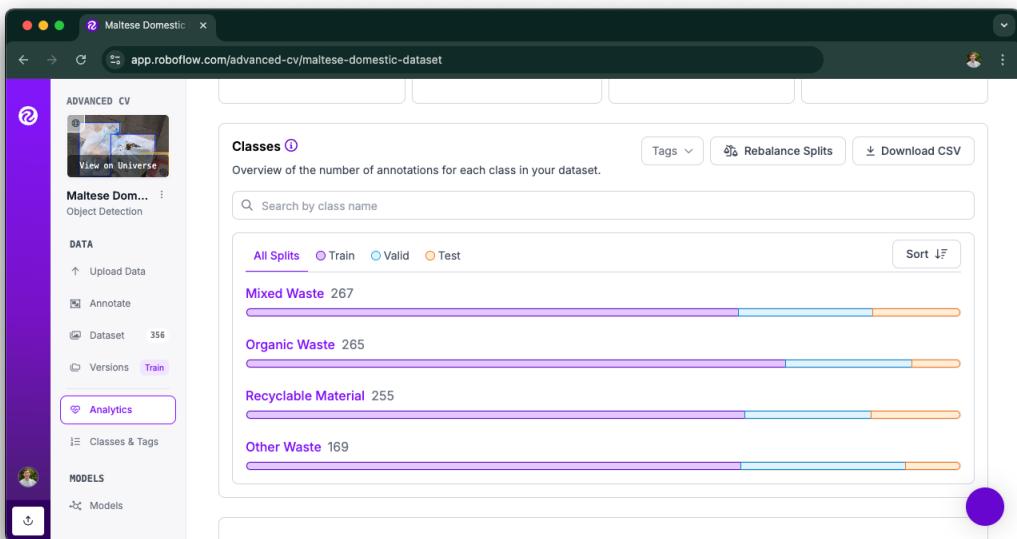


Figure 1: Roboflow Analytics - Distribution of annotations for each waste class in the MDD.

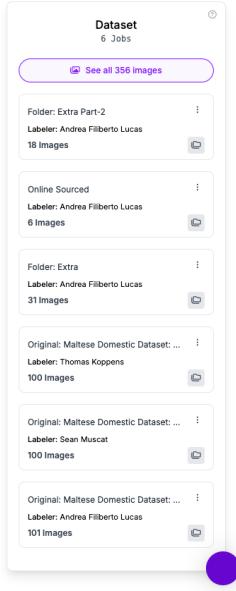


Figure 2: Roboflow Analytics - Jobs created for annotation

Using Roboflow, six annotation jobs were created to evenly distribute the workload among the three contributors, as illustrated in Figure 2. Bounding box annotations followed a standardized colour scheme for organization: **Red** for mixed waste, **Blue** for recyclable materials, **Green** for organic waste, and **Orange** for other waste. Additionally, the online sourced photos, were tagged as "*Online-Sourced*" using Roboflow's tagging system, which allows additional information to be associated with images. An example of these annotations is depicted in Figure 3.

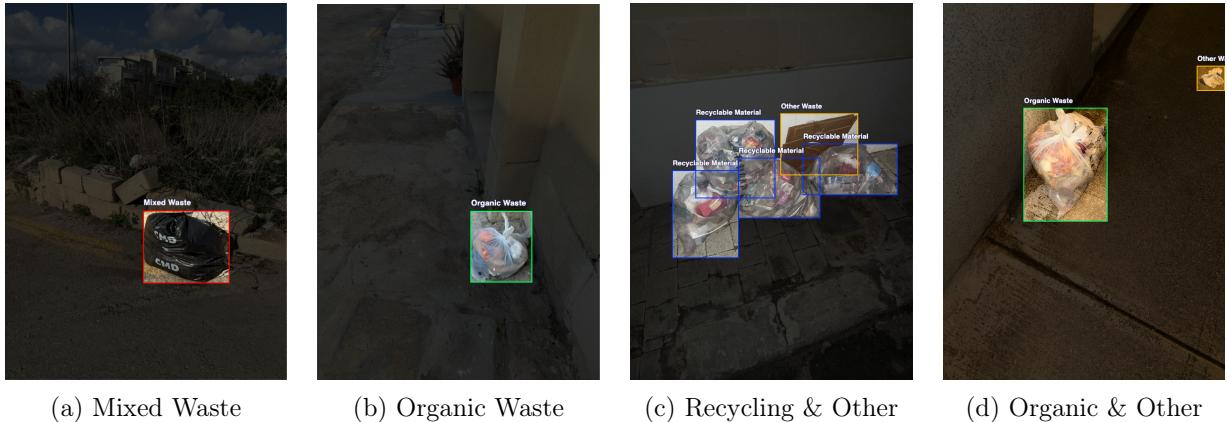
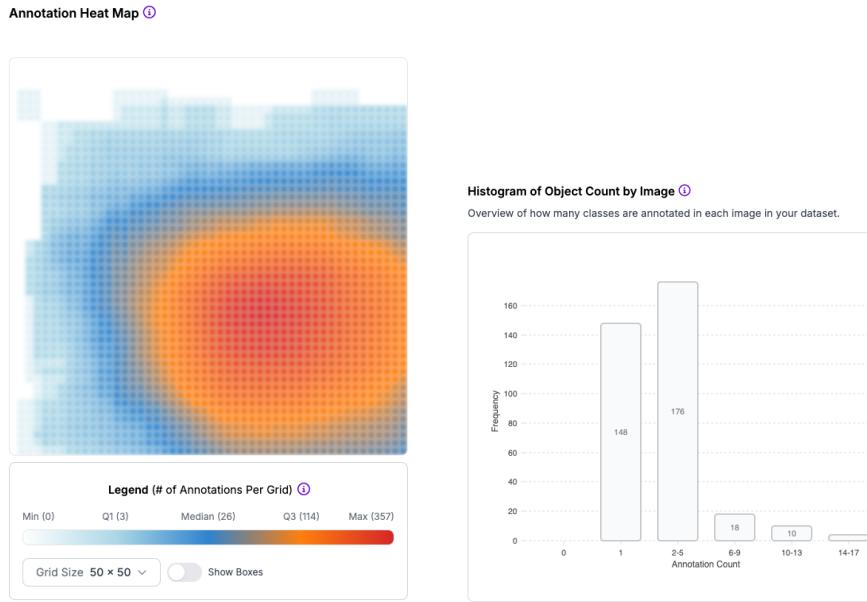


Figure 3: Sample annotations showing labelled data for different waste categories.

This annotation process resulted in a **total of 956 bounding boxes**, averaging 2.7 annotations per image across the four classes. Specifically, there were 267 annotations for mixed waste, 265 for organic waste, 255 for recyclable material, and 169 for other waste, as detailed in Figure 1.

Further insights into the dataset were provided by Roboflow analytics. A heat map, shown in Figure 4a, highlights the areas with the highest density of annotations, using colour gradients to signify annotation counts per grid cell. Additionally, a histogram of object counts per image, presented in Figure 4b, offers an overview of how many classes are annotated in each image.



(a) Annotation Heat Map showing the distribution of annotations in the dataset.
(b) Histogram of Object Count by Image showing the frequency of annotated classes per image.

Figure 4: Roboflow Analytics - Visual Analysis of MDD.

Finally, Roboflow’s class rebalance and split feature was employed to ensure that the dataset was evenly divided for training, validation, and testing purposes, optimizing its use for model development.

3.3 Dataset Version

This subsection introduces the **MDD-AFL** version of the dataset, which was utilized by all team members in the methodology and evaluation sections of this document. It outlines the data pre-processing and augmentation techniques applied to standardize and enhance the dataset, ensuring consistency and robustness for the project’s objectives.

3.3.1 Preprocessing

During preprocessing, two primary operations were applied to the dataset: **Auto-Orient** and **Resize: Fit (reflect edges)**. The images were resized to a resolution of **640x640 pixels**, a dimension chosen due to its compatibility with popular object detection models such as YOLO and COCO. The resolution strikes a balance between computational efficiency and sufficient spatial detail, making it ideal for training robust machine learning models. A standardized resolution like 640x640 ensures uniformity in image size, reducing inconsistencies that could negatively impact model performance.

The **reflect edges** technique used during resizing is a padding method that ensures the original image’s aspect ratio is preserved while fitting it within the target dimensions. If the original image dimensions do not align with 640x640, the excess space is filled by mirroring the edges of the image. This method minimizes visual artifacts and avoids distorting the original image content, making it an effective preprocessing step for object detection tasks. By maintaining the integrity of the image content, this approach enhances model training by providing consistent, high-quality input data.

3.3.2 Data Augmentation

To enhance the diversity of the dataset and improve model generalization, various data augmentation techniques were applied, resulting in three additional variations for each training example. These augmentations included **90° rotations**, where images were rotated clockwise, counter-clockwise, and upside down, introducing spatial variability. **Horizontal and vertical shear transformations** were also applied, with a range of $\pm 10^\circ$, to simulate perspective distortions. **Brightness** adjustments were performed, altering brightness levels randomly between -22% and +22% to replicate diverse lighting conditions. **Gaussian blur**, with a kernel size of up to 1 pixel, was used to mimic slight image defocus, while **random noise**, affecting up to 1.96% of the pixels, was introduced to emulate sensor imperfections.

As a result of these augmentations, the dataset expanded to include a **total of 856 images**. These were subsequently split into three subsets: the training set, comprising 88% of the dataset (750 images); the validation set, comprising 8% (72 images); and the test set, comprising 4% (34 images), as depicted in Figure 5.

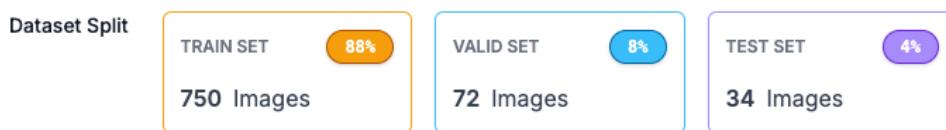


Figure 5: Roboflow Analytics - Total Images per Split after Augmentation

3.4 Dataset Visualisation and Analytics

The dataset visualization and analytics processes were implemented in `DatasetVisualization_AFL.ipynb`. This notebook facilitated dataset exploration and analysis by automating tasks, leveraging visualization tools, and sampling the dataset to ensure it was well-structured and suitable for use in the YOLOv8 model.

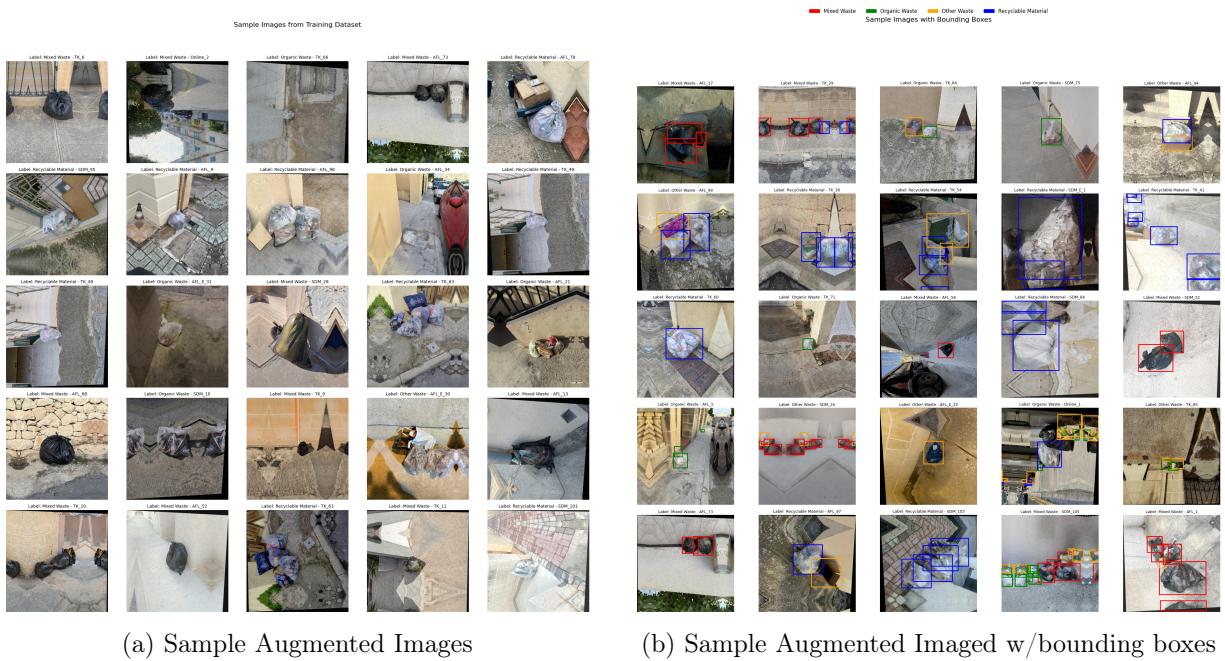
3.4.1 Sample Visualisation

Initially, the dataset structure was organized by validating folder hierarchies and managing dataset versions through Python scripts. Missing dependencies, such as `roboflow`, were automatically installed to ensure seamless integration with the dataset hosted on Roboflow. The dataset was downloaded into a designated directory, `../Versions/MDD-AFL-Yolov8`, which served as the root folder for training, validation, and testing splits compatible with the YOLO model.

Visualizations were generated to inspect both raw images and annotated images with bounding boxes. The consistent colour scheme for bounding boxes, as already mentioned in 3.2, was applied for clarity. An example of these bounding box annotations is shown in Figure 6a and 6b. Additionally, the visualizations enabled the inspection of image-label pairings across dataset splits, confirming annotation accuracy and quality.

3.4.2 Dataset Analytics

The dataset analytics were conducted using a series of automated Python scripts that provided detailed insights into the composition, completeness, and distribution of the dataset. The function `process_file` was used to extract bounding box counts and class distributions from the label files, while the `gather_statistics_parallel` function employed multithreading to efficiently analyse

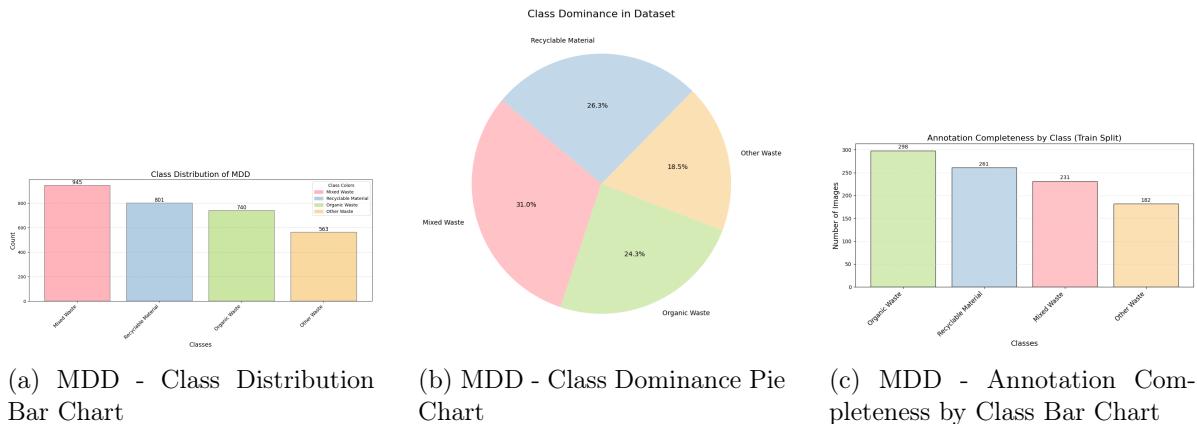


(a) Sample Augmented Images

(b) Sample Augmented Images w/bounding boxes

Figure 6: MDD - Data Visualisation

the dataset splits. This process revealed a total of 856 images and 100% annotation completeness. Further analysis of class distributions was conducted using the `plot_class_distribution` function, which highlighted the distribution of the four main classes. The results, visualised in Figure 7a, demonstrated a reasonable balance among the classes, with counts of 945 for Mixed Waste, 740 for Organic Waste, 563 for Other Waste, and 801 for Recyclable Material. A class dominance pie chart was also generated using the `class_dominance_pie_chart` function, as shown in Figure 7b, providing a clear visual representation of the proportional presence of each class.



(a) MDD - Class Distribution Bar Chart

(b) MDD - Class Dominance Pie Chart

(c) MDD - Annotation Completeness by Class Bar Chart

Figure 7: MDD - Class Visualization

The annotation completeness by class was further examined using the `annotation_completeness_by_class` function, which provided insights into the distribution of class annotations across the dataset splits. This analysis ensured that all classes were adequately represented in the training data, as depicted in Figure 7c.

Dataset statistics by split were summarized using the `display_statistics` function, which calculated the total number of bounding boxes and images for each split. The training split contained 2,686 bounding boxes across 748 images, the validation split had 228 bounding boxes across 70 images, and the test split included 135 bounding boxes across 34 images. These statistics are shown in Figures 8a and 8b, providing a comprehensive overview of the dataset's structure.

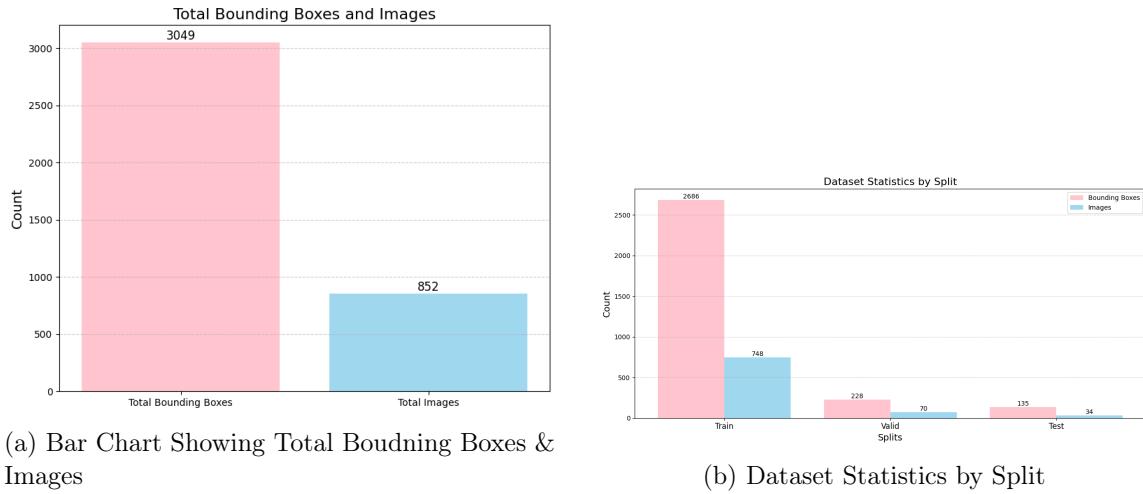


Figure 8: MDD - Dataset Statistics

The `object_count_histogram` function was used to analyse the distribution of object counts per image within the training split. The histogram, displayed in Figure 9, revealed the frequency of images containing various numbers of annotated objects, offering insights into the dataset's complexity.

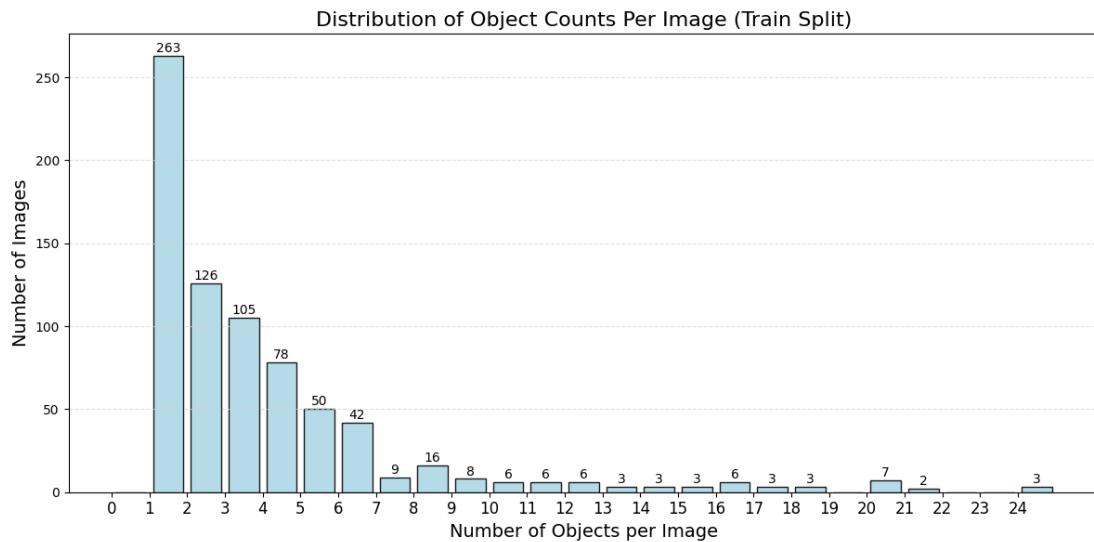


Figure 9: Distribution of Object Counts per Image (Train Split)

The next stage of analytics involved generating density heat maps to visualize the spatial distribution of annotated bounding boxes within the dataset. The `compute_density_heatmap` function calculated the density of annotations across normalized positions for each dataset split and for all splits combined. These heat maps were visualised using the `density_heatmap_all_splits` function, which provided a detailed overview of annotation density for each split. The resulting heat maps, shown in Figure 10a, enabled us to identify areas with higher concentrations of annotations.

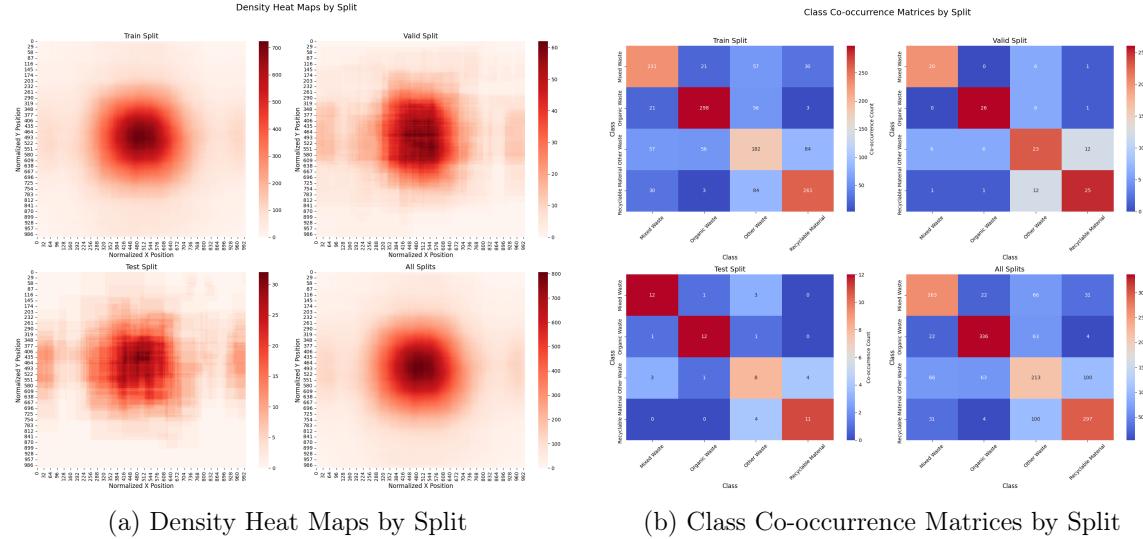


Figure 10: MDD - Visual Analysis

Finally, a class co-occurrence matrix was computed to examine the frequency of co-occurring classes within individual images across the dataset splits. The `compute_cooccurrence` function built matrices by analyzing the presence of multiple classes within the same annotations, while the `class_cooccurrence_matrix_all_splits` function visualised these matrices for the training, validation, testing splits, and the combined dataset. The matrices, visualised in Figure 10b, revealed relationships between classes, highlighting patterns that could influence model performance.

4 Implementation

The following section discusses the implementation of YOLOv8, YOLOv5, and RetinaNet models for the MDD. Each model's integration, training, and evaluation process is detailed, highlighting the methodologies employed to optimise their performance for the dataset.

4.1 Implementation: YOLOv8

The implementation began with the setup of the project directory and the initialization of necessary dependencies. A script was created to manage the folder structure, ensuring the `Versions` directory and the target subfolder `MDD-AFL-Yolov8` were correctly organized. If these folders did not exist, they were automatically created. To streamline dependency management, the script also checked for the installation of required libraries such as `roboflow`, installing them if missing.

Using the `roboflow` library, the dataset was downloaded after the user provided their `roboflow` API key. The dataset was retrieved from the `maltese-domestic-dataset` project and stored locally under the `MDD-AFL-Yolov8` folder within the `Versions` directory. Additional steps were implemented to rename and move the downloaded folder to ensure consistent organization, enabling seamless integration with the training pipeline.

Next, the training environment was configured with automated checks and installation of any additional dependencies defined in a JSON configuration file. The setup dynamically determined the computational device; CUDA for NVIDIA GPUs, MPS for Apple Silicon hardware, or CPU for systems lacking hardware acceleration, ensuring efficient use of available resources. The `ultralytics` library was used to manage the training and evaluation of the YOLOv8 model.

To optimize the performance of the YOLOv8 model, multiple experiments were conducted by adjusting hyperparameters over six key iterations. **v1** started simply with the YOLOv8n model and 50 epochs, establishing a baseline. **v2** introduced the YOLOv8s model with 55 epochs, resulting in a significant performance improvement. **v3** increased the epochs to 100 and used the YOLOv8l model, further enhancing results. In **v4**, early stopping was introduced with a patience value of 10, dropout set to 0.3, and 200 epochs, though training stopped automatically after 31 epochs. **v5** used the same configuration as **v4** but experimented with the learning rate (`lr0=0.001` and `lrf=0.00001`). Finally, **v6**, the best-performing iteration, retained the learning rate from **v5** but adjusted `lr0=0.001`. Throughout these experiments, some hyperparameters—such as `imgsz=640` and `batch_size=8`—remained constant due to their consistent effectiveness in earlier trials.

The dataset integration was facilitated via a `data.yaml` configuration file, and training parameters such as epochs, patience, learning rate, and dropout rates were fine-tuned in multiple iterations. YOLOv8 model architectures, including nano, small, and large versions, were evaluated to optimize performance for detecting the four waste classes. Training was conducted from scratch without using pre-trained weights, tailoring the model specifically to the MDD.

Despite the computational demands, particularly on lower-end hardware, training runtimes were recorded for each experiment. These efforts culminated in a robust object detection model capable of effectively classifying waste categories. Evaluation metrics obtained through YOLOv8's validation tools provided insights into model performance, guiding further refinements to the training process.

4.2 Implementation: YOLOv5

The environment is initially set up to ensure that all required directories and dependencies are available. If it is not already present, a folder named Versions is created, and the dataset is saved in the MDD-SDM-Yolov5 subfolder. The user is required to provide their Roboflow API key to download the Maltese Domestic Dataset, and if the roboflow library isn't already present, it is promptly installed. The dataset folder is renamed and moved to the correct location post-download, preparing it for use in subsequent stages. It is essential to understand that not all content in the Versions folder is uploaded to GitHub because of the dataset's large size and the numerous images.

The code accesses a JSON file that outlines the required libraries to ensure their availability. Each library is set up once it has been verified, while those not installed are added dynamically. The execution provides a solid foundation for handling image processing, data visualization, and model training due to essential libraries.

The YOLOv5 model is initialized using the configuration file yolov5n.yaml, which outlines the architecture of the model. This setup establishes the basis for training, utilizing the dataset configuration (data.yaml). To enhance the model's performance, key training parameters are fine-tuned, including the number of epochs, image dimensions, batch size, and patience. Every run is assigned a subfolder that corresponds to its version, and training outcomes are stored in the runs/detect directory. After finishing the training, the model undergoes validation, and metrics are presented to assess its performance.

Users are able to review and observe the results of specific trained models once the training process is complete. The execution identifies folders in runs/detect that fulfill specific criteria, such as containing the SDM identification. When accessible, summary visuals like results.png are provided, alongside images of assessment and prediction results. This interactive process enables users to analyze the outcomes of their training sessions effectively.

The algorithm also offers a comprehensive analysis of training metrics and losses. It analyzes the results and calculates final and average figures for metrics such as precision, recall, mAP (mean average precision), and various other loss types from a CSV file generated during training. These metrics are presented in detailed graphs that offer insights into the model's effectiveness and reveal trends over time. Scatter plots displaying confidence scores and bar charts depicting class distributions add depth to the analysis.

The solution includes features to import images, verify they are in the correct RGB format, and display both the original and predicted images side by side for testing and analysis purposes. Bounding boxes are utilized to represent identified objects once they have been screened by confidence levels. The analytics module's examination of discovered items emphasizes class distributions and confidence scores. It is easier to review single detections when cropped images of recognized objects are displayed in a grouped grid.

The trained model is subsequently used on the photos in the test folder during the inference phase of the workflow. The code analyzes every image, generates predictions, and checks for the existence of the test folder. Bounding boxes are utilized to show the outcomes, and the same instruments employed during the training stage are used to evaluate the detections. This maintains a uniform evaluation process and ensures a seamless shift from training to testing.

4.3 Implementation: RetinaNet

Hyperparameters were determined based on manual experimentation, with the AdamW optimizer using an initial learning rate of 1e-4. For early stopping, the patience was set to 2. However, the early stopping logic saves the model each time it outperforms the current best model. This ensures that the best model may be loaded in the next cell such that the best possible model is used for evaluation. The models are saved as `retinanet_vx.pt` where x denotes the version number. A total of 8 models were trained with the best model being `retinanet_v5.pt`.

The training loop is simple, looping over the epochs and mini batches, feeding the images and targets to the model for the forward step, calculating the losses and completing the backwards step, and saving the train and validation losses at each epoch. The early stopping logic is also rather simple and based on the validation set loss. Finally, the train and validation losses are visualised to understand the training process and stability.

Once training is complete, the weights corresponding to the lowest validation loss are loaded. Then, a series of tests are conducted on the test set. These include a calculation of the mAP metrics and a visualisation of the predictions.

The implementation of the mAP metric is left to PyTorch Metric's MeanAveragePrecision module. This takes the outputs of the final model and the targets and updates the metrics with each epoch. Then, the metrics are displayed. Furthermore, the predictions from the model's outputs are first processed. This processing consists of applying non-max suppression (NMS) such that only predictions above a certain confidence threshold are considered. The models tended to be quite confident in many of their predictions, and thus the best confidence threshold was determined to be 0.5. The second stage of prediction processing ensures that there are not too many predictions, as this led to memory errors. Thus, only the first 100 predictions are considered. These filtered predictions are then passed to the mAP metrics to update their values.

For the visualisation of the predictions, care was taken to make the bounding boxes and general visualisation similar to the other notebooks. The image is displayed on the left with the ground truth boxes and labels to easily compare the predictions to. Then, a copy of the image is displayed on the right with the predicted bounding boxes, labels, and confidence scores. Naturally, NMS is once again applied such that low confidence predictions are filtered out.

5 Evaluation

In this section, the evaluation of each implementation will be presented in individual subsections, written by the respective contributors. The final comparison of results will be discussed in Section 6.

5.1 Evaluation: YOLOv8

This evaluation focuses on MDD-AFL-Yolov8_v6, the experiment with the best results. Training was configured with 500 epochs (stopping at 94 due to early stopping), a patience of 10, a dropout rate of 0.4, a learning rate schedule using `cos_lr`, `lr0` set to 0.0001, and `lrf` set to 0.00001, achieving strongest performance for the augmented MDD dataset with a final precision of 0.623.

5.1.1 Ultralytics Evaluation Metrics

Confusion Matrix: The confusion matrix highlights the model's classification accuracy, with 42 correct predictions for both Mixed Waste and Recyclable Material and 39 for Organic Waste. Background misclassifications were most frequent with Mixed Waste (33 false positives), indicating challenges in differentiating between waste and background features in cluttered environments. Despite this, the correct classifications for the main three categories demonstrate the model's ability to generalize effectively for practical use.

Normalized Confusion Matrix: The normalized confusion matrix shows Mixed Waste, Organic Waste, and Recyclable Material achieving correct prediction rates of 63%, 74%, and 81%, respectively. The higher rates for Organic and Recyclable Waste suggest the model excels in identifying structured patterns, such as bottles and biodegradable materials, while still performing well for Mixed Waste.

Training and Validation Loss: Loss metrics consistently improved across epochs, converging at epoch 94, with minimal overfitting. The model achieved a peak `mAP@0.5` of 60.9%, demonstrating strong generalization across the dataset and suitability for deployment in waste classification systems.

Precision-Confidence Curve: Recyclable Material achieved the highest precision, exceeding 90% at higher confidence thresholds, followed by Organic Waste with 80–85%. Mixed Waste reached around 60%, showing moderate precision. These results highlight the model's reliability for precise detection, particularly in scenarios like recycling plants where minimizing false positives is critical.

Recall-Confidence Curve: Organic Waste and Recyclable Material achieved recall values of 84% and 81%, respectively, while Mixed Waste reached 74%. This demonstrates the model's ability to detect most instances of the main categories, ensuring high coverage in waste classification.

Precision-Recall Curve: Recyclable Material and Organic Waste achieved precision-recall scores of 0.839 and 0.806, with Mixed Waste at 0.607. This indicates the model's strong ability to balance precision and recall for the main categories, ensuring accurate and comprehensive detection, crucial for real-world applications like automated waste sorting.

F1-Confidence Curve: Organic Waste and Recyclable Material achieved peak F1-scores of 0.74 and 0.81, reflecting strong detection and classification balance. Mixed Waste had a peak F1 score of 0.63, showing reasonable performance. These results indicate the model effectively balances detection and accuracy for the primary waste categories, ensuring reliable performance across diverse conditions.

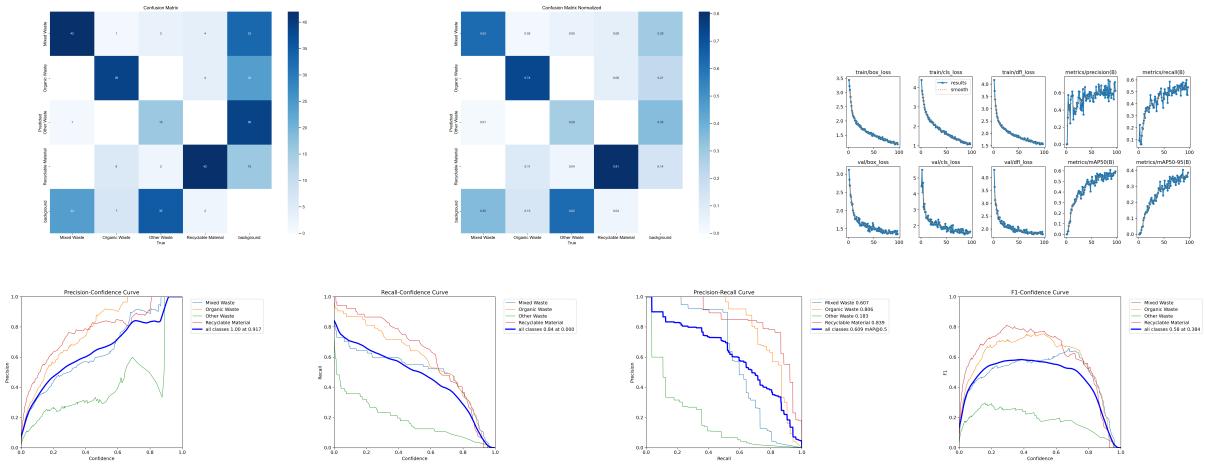


Figure 11: Evaluations [1] - Ultralytics found in MDD-AFL-Yolov8-v6R

5.1.2 Code-Based Metrics

The analysis of code-based metrics involved reading the results stored in the `results.csv` file located in the selected version’s directory. A custom script computed the final and average values for key metrics, including precision, recall, $mAP@50$, and $mAP@50-95$, along with loss values for training and validation phases.

Precision: The final precision for the model was 0.623, with an average of 0.593. This reflects the model’s reliability in making correct predictions, particularly for Recyclable and Organic Waste. However, the lower precision for Mixed Waste and occasional confusion with background items impacted the overall precision score. For practical applications, this means the model performs well in high-confidence scenarios but requires refinement in distinguishing more ambiguous waste types.

Recall: The final recall achieved was 0.539, with an average of 0.435. This metric highlights the model’s ability to detect waste instances effectively, with Organic Waste and Recyclable Material having higher recall rates than Mixed Waste. While adequate for many tasks, improving recall would enhance the model’s applicability in environments requiring comprehensive waste detection, such as automated sorting systems.

$mAP@50$ and $mAP@50-95$: The final $mAP@50$ value of 0.595 and $mAP@50-95$ value of 0.384 indicate good performance for the primary categories but highlight room for improvement in fine-grained localization and detection. The average values of 0.437 ($mAP@50$) and 0.259 ($mAP@50-95$) confirm the model’s consistent performance across epochs.

Training and Validation Loss: Training box loss decreased to 1.197 with an average of 1.633, while classification loss ended at 1.007 with an average of 1.813. Validation box loss reached 1.385 (average 1.617), and validation classification loss concluded at 1.635 (average 2.062). The steady decline in loss values over epochs demonstrates the model’s effective learning process, with minimal signs of overfitting. These metrics underscore the stability of the model throughout training, reflecting its ability to generalize well across the dataset.

Learning Rate: The learning rate schedule demonstrated a peak at 0.0012, gradually declining over epochs. This approach ensured stable convergence while preventing abrupt oscillations

in loss values, contributing to the model's steady improvement across all metrics.

Precision vs. Recall: The precision vs. recall distribution showed clustering around higher precision values for Organic and Recyclable Waste, while Mixed Waste had more dispersed points, reflecting its lower detection confidence. This suggests that the model is more adept at classifying structured waste types but struggles slightly with visually ambiguous items, which could be improved with further training or augmentation.

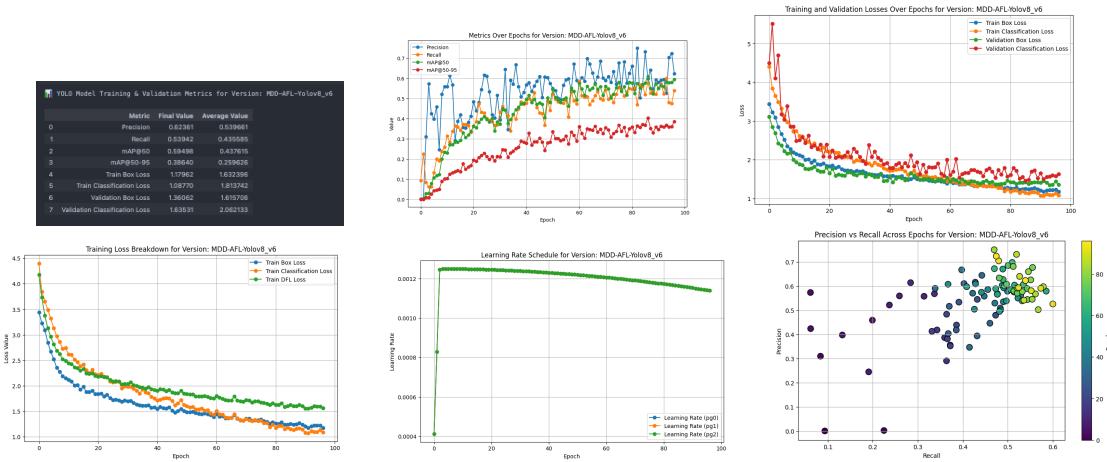


Figure 12: Evaluations [2] - Code-Based Metrics

Practical Implications: The evaluated metrics highlight the model's robustness for primary waste categories, with strong precision and reasonable recall, making it suitable for real-world applications like waste management and recycling. However, the slightly lower performance on Mixed Waste and mAP@50–95 suggests that refining the model's feature extraction capabilities could further enhance its generalizability and accuracy.

5.1.3 Model Visualisation

The YOLOv8 model was evaluated on unseen test images to assess its detection and classification capabilities for waste categories. Each image was processed to display bounding boxes for detected objects and filtered using a confidence threshold of 0.36. The visualizations provided insights into the model's performance, with bounding boxes rendered on predicted images and additional analytics, including class counts and confidence scores.

In one sample (Figure 13), the model detected four objects, including three instances of Mixed Waste with confidence scores of 0.91, 0.89, and 0.65, and one instance of Other Waste at 0.95 confidence. The bar chart confirmed the dominance of Mixed Waste, while the scatter plot highlighted consistently high confidence scores. This demonstrates the model's strength in identifying Mixed Waste, though the varying confidence levels suggest potential visual overlaps or ambiguities.

Another sample (Figure 14) revealed two detections classified as Organic Waste, with confidence scores of 0.85 and 0.49. While the higher-confidence prediction accurately identified Organic Waste, the lower-confidence prediction may indicate occlusions or overlapping visual features. The bar chart and scatter plot verified these observations, reflecting the model's general reliability for detecting organic materials while highlighting areas for improvement in distinguishing ambiguous objects.

These results underscore the model's robust performance in identifying key waste categories with high-confidence predictions. However, the occasional lower-confidence predictions emphasize the need for further refinement in handling cluttered or visually overlapping environments, ensuring consistent accuracy in real-world applications.

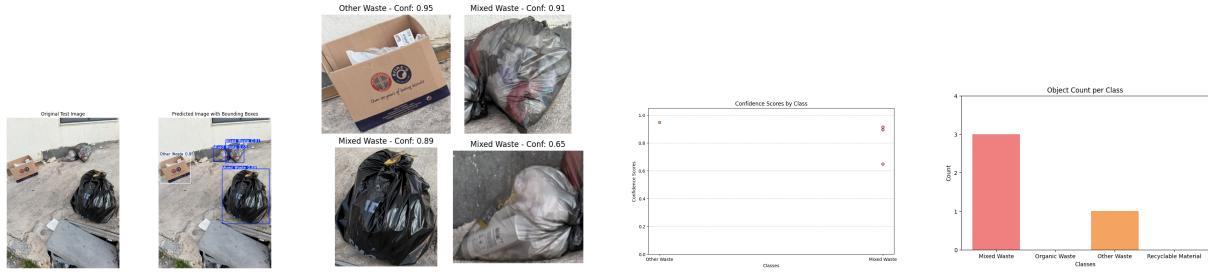


Figure 13: Visualisation Sample 1



Figure 14: Visualisation Sample 2

5.2 Evaluation: YOLOv5

For reference, we will be evaluating the YOLOv5 run which was trained with 60 epochs, so run v5.

Confusion Matrix: The confusion matrix shows the actual versus predicted labels, with "Recyclable Material" achieving the highest correct count of 41 instances. Notable misclassifications include 33 "Background" instances as "Mixed Waste" and 17 "Organic Waste" instances as "Background," highlighting areas for improvement. This matrix helps identify misclassification trends to refine the model.

Normalised Confusion Matrix: The normalised confusion matrix breaks down classification performance by showing correctly classified instances as proportions and highlighting misclassifications. "Recyclable Material" has a strong performance with 0.79 correct predictions, though confusion exists with "Mixed Waste" and "Background." This matrix aids in spotting frequently confused classes.

F1-Confidence Curve: The F1-confidence curve shows the balance between precision and recall across confidence thresholds. For "all classes," the highest F1 score is 0.54 at a confidence of 0.474, indicating an optimal threshold for balancing these metrics.

Precision-Confidence Curve: This curve shows how precision varies with confidence thresholds. For "all classes," precision reaches 1.00 at a confidence of 0.902, indicating high reliability at high thresholds but decreasing at lower confidence levels.

Precision-Recall Curve: The precision-recall curve shows the trade-off between these metrics for each class. "Recyclable Material" achieves the highest AUC of 0.799, while "Other Waste" has the lowest AUC of 0.200. The overall mAP at IoU 0.5 is 0.552, reflecting moderate performance.

Recall-Confidence Curve: This graph illustrates how recall varies with confidence thresholds. A steeper curve indicates better performance in maintaining recall across thresholds. The bold blue line for "all classes" achieves a recall of 0.79 at a confidence of 0.0, capturing most true positives.

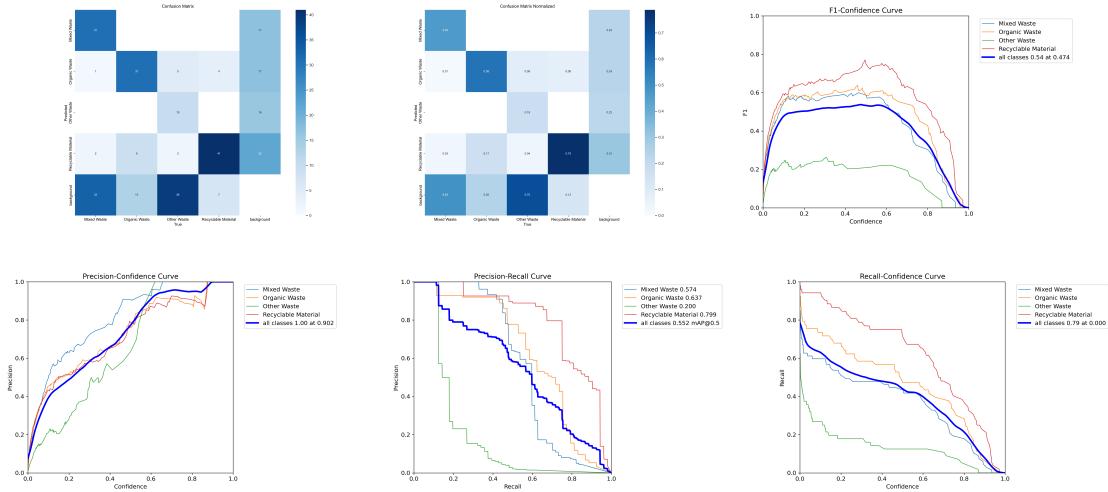


Figure 15: Visualisations found in the MDD-SDM-Yolov5-v5R folder

This is the 'results.png' visualisation from 'MDD-SDM-Yolov5-v5', showing training and validation metrics across epochs for the YOLOv5 model. The loss curves for box, classification, and distributional focal loss decrease consistently, indicating effective learning. Validation losses show a similar trend with slight fluctuations, reflecting dataset variability. Metrics for precision, recall, and mean average precision (mAP) improve gradually, highlighting the model's growing accuracy in detecting and classifying objects. Overall, the trends indicate balanced optimisation.

Following that, the other figure below summarises the training and validation metrics for the YOLOv5 model. The model achieved a final precision of 0.660, showing reliable positive predictions, while recall, at 0.468, indicates potential for improvement. Mean average precision values of 0.541 at IoU 0.5 and 0.317 at IoU 0.5–0.95 reflect moderate detection accuracy. Training and validation losses for box and classification consistently decreased, with final values such as 1.258 for train box loss and 1.515 for validation box loss, demonstrating effective learning and stable optimisation.

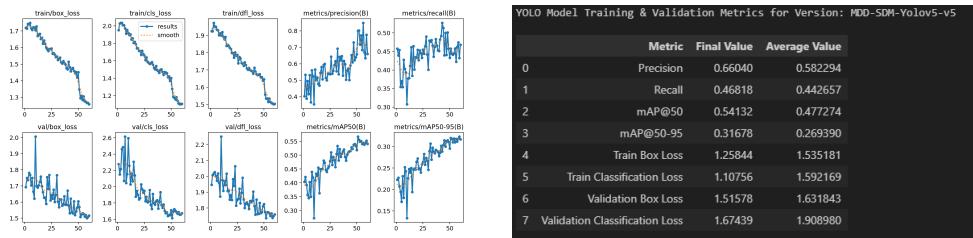


Figure 16: results.png & training and validation metrics table

The following visualisations have been plotted and generated by us in the notebooks, for the purpose of monitoring Yolov5 model performances.

Metrics Over Epochs: This graph shows steady improvements in precision, recall, and mAP metrics as training progresses. Precision increases significantly, while mAP@50 and mAP@50–95 rise steadily, indicating improved performance across different IoU thresholds. **Training Loss Breakdown:** This graph shows training losses for box, classification, and DFL, all of which decrease consistently. The sharper decline in early epochs confirms effective optimisation and the model's ability to minimise errors.

Training and Validation Losses Over Epochs: Training and validation losses decrease consistently across epochs. Slight fluctuations in validation losses suggest dataset variability but remain closely aligned with training losses, reflecting good generalisation.

Precision vs Recall Across Epochs: The scatter plot shows precision and recall trends for each epoch, coloured by epoch number. Later epochs show higher precision, while recall varies slightly, helping identify balanced precision-recall trade-offs.

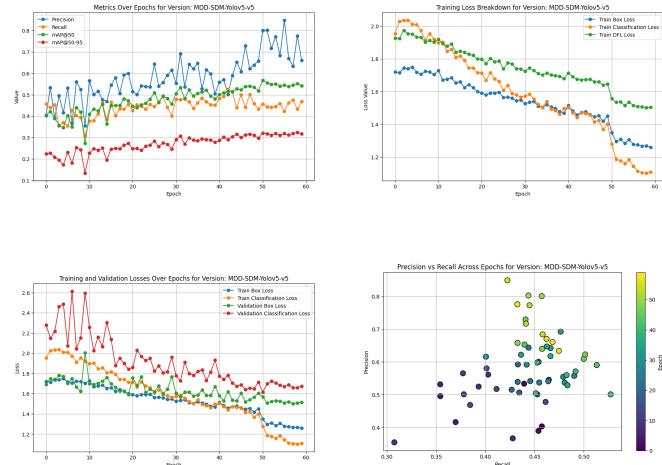


Figure 17: Visualizations we created in the notebook to assess model performance.

Model results: In the figures below you can see some of the model outputs for this run, for the most part the model detects all the classes well, only slightly struggling with the smaller bags further in the distance.



Figure 18: Object detection in test images

5.3 Evaluation: RetinaNet

Unlike the Ultralytics approaches which may simply make use of the automatically generated metrics and plots, the evaluation for the RetinaNet model must be done manually. This makes generating things like confusion matrices much more complex and tedious, especially since many inbuilt metrics run into the issue of not being explicitly made for object detection purposes (i.e. handling varying numbers of predictions compared to the number of ground truths). Thus, only the mAP family of metrics were used to evaluate this model.

The mAP50 for the final model with NMS applied at `confidence_threshold=0.5` is 65.9%, outperforming the two YOLO models as expected given the trade-off of accuracy for speed. The mAP50-95 also beats the YOLO's scores at 46.9%. Below is a table displaying the mAP50-95 for each class individually.

Category	Mixed Waste	Organic Waste	Recyclable Material	Other Waste
mAP50-95 (%)	51.6	43.8	68.8	23.5

Table 1: mAP50-95 per category

The model had the most trouble with the Other Waste category, which makes sense seeing as the category could refer to anything from a soft drink can to a cardboard box. On the other hand, the model performed best on the Recyclable Material category, with the bags being white and distinct but not arranged in as complex patterns as the Mixed Waste bags in the test set's images.



Figure 19: Perfect Predictions



Figure 20: Overlapping Predictions and Hallucinations

As seen in the above figures, the model is capable of perfectly predicting difficult images. On the other hand, it is still susceptible to overlapping predictions, that is placing multiple boxes on a single object, and finding waste where there is not any, or mislabelling it. While more could likely be done to train the model more efficiently such that it learns the correct concepts and detects waste more accurately, a larger dataset with more complex examples with many bags sitting on each other would benefit the model the most. These difficult instances are what the focal loss is all about, and those would push RetinaNet to learn the concepts of these waste categories better.

6 Concluding Remarks

This research explored the classification of Maltese domestic waste bags using YOLOv8, YOLOv5, and RetinaNet, emphasising data preprocessing, implementation, and evaluation. Data preparation involved creating the Maltese Domestic Dataset (MDD) with consistent resizing, reflective padding, and extensive augmentation techniques to ensure robust model training.

The evaluation compared the models' performance, highlighting RetinaNet's superior accuracy with an mAP@50-95 of 46.9%, outperforming YOLOv8 (38.4%) and YOLOv5 (31.7%). RetinaNet's focal loss effectively handled class imbalances, particularly for challenging categories like "Other Waste." In contrast, YOLOv8 excelled in speed and achieved a higher mAP@50 of 59.5%, while YOLOv5 demonstrated balanced performance across detection precision and recall. These results showcase the trade-offs between speed, accuracy, and robustness across models.

Overall, this study underscores the importance of preprocessing and model selection for object detection in complex real-world scenarios. Future work will focus on refining detection of ambiguous waste types and scaling datasets to improve classification accuracy and practical applicability.

7 References

- [1] J. R. Terven and D. M. Cordova-Esparza, "A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS," *Machine Learning and Knowledge Extraction*, vol. 5, no. 4, pp. 931–956, 2024. [Online]. Available: <https://doi.org/10.3390/make5040083>.
- [2] D. Vieri, R. Susanto, E. S. Purwanto, and M. K. Ario, "Enhancing Waste Classification with YOLOv8 Models for Efficient and Accurate Sorting," *Procedia Computer Science*, vol. 245, pp. 889–895, Jan. 2024, doi: [10.1016/j.procs.2024.10.316](https://doi.org/10.1016/j.procs.2024.10.316).
- [3] D. Reis, J. Hong, J. Kupec, A. Daoudi, and Georgia Institute of Technology, "Real-Time Flying Object Detection with YOLOv8," May 22, 2024.
- [4] Y. Zhou, L. Lin, and T. Wang, "Garbage classification detection system based on the YOLOv8 algorithm," *AIP Advances*, vol. 14, no. 12, Dec. 2024, doi: [10.1063/5.0244795](https://doi.org/10.1063/5.0244795).
- [5] M. Sohan, T. S. Ram, and Ch. V. R. Reddy, "A review on YOLOv8 and its advancements," *Algorithms for Intelligent Systems*, pp. 529–545, Jan. 2024, doi: [10.1007/978-981-99-7962-2_39](https://doi.org/10.1007/978-981-99-7962-2_39).

- [6] S. Pan, N. Wang, Y. Lin, J. Tang, and Wenzhou Polytechnic, "Based on YOLOv8 Intelligent Trash can Garbage Classification Detection algorithm," journal-article, 2024.
- [7] H. Huang, B. Wang, J. Xiao, and T. Zhu, "Improved small-object detection using YOLOv8: A comparative study," Applied and Computational Engineering, vol. 41, no. 1, pp. 80–88, Feb. 2024, doi: 10.54254/2755-2721/41/20230714.
- [8] M. Shroff, A. Desai, and D. Garg, "YOLOv8-based Waste Detection System for Recycling Plants: A Deep Learning Approach," Oct. 18, 2023, pp. 01–09. doi: 10.1109/icssas57918.2023.10331643.
- [9] Ultralytics, "YOLOv8 Models Documentation," [Online]. Available: <https://docs.ultralytics.com/models/yolov8/>. [Accessed: 23-Jan-2025].
- [10] YOLO v5 model architecture [Explained] *opengenus*. [Online]. Available: <https://iq.opengenus.org/yolov5/>. [Accessed: Jan. 17, 2025]
- [11] M. L. Ali and Z. Zhang, "The YOLO Framework: A Comprehensive Review of Evolution, Applications, and Benchmarks in Object Detection," *Computers*, vol. 13, no. 12, p. 336, Dec. 2024. DOI: 10.3390/computers13120336.
- [12] R. Khanam and M. Hussain, "What is YOLOv5: A Deep Look into the Internal Features of the Popular Object Detector," *Department of Computer Science, Huddersfield University*, Jul. 2024. DOI: arXiv:2407.20892.
- [13] R. Del Prete, M. D. Graziano, and A. Renga, "RetinaNet: A deep learning architecture to achieve a robust wake detector in SAR images," 2018 IEEE 4th International Forum on Research and Technology for Society and Industry (RTSI), pp. 171–176, Sep. 2021, doi: 10.1109/rtsi50628.2021.9597297.
- [14] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 2014, pp. 580-587, doi: 10.1109/CVPR.2014.81.
- [15] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," in Lecture notes in computer science, 2014, pp. 346–361. doi: 10.1007/978-3-319-10578-9_23.
- [16] L. Li, F. Xue, D. Liang, and X. Chen, "A hard example mining approach for concealed Multi-Object detection of active terahertz image," Applied Sciences, vol. 11, no. 23, p. 11241, Nov. 2021, doi: 10.3390/app112311241.
- [17] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," arXiv (Cornell University), Jan. 2015, doi: 10.48550/arxiv.1506.01497.
- [18] T.-Y. Lin et al., "Microsoft COCO: Common Objects in context," arXiv (Cornell University), Jan. 2014, doi: 10.48550/arxiv.1405.0312.
- [19] T. -Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan and S. Belongie, "Feature Pyramid Networks for Object Detection," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 2017, pp. 936-944, doi: 10.1109/CVPR.2017.106.
- [20] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

- [21] F. Debono, "Franco Debono calls on Wasteserv CEO to resign after suggestion to freeze organic waste," *The Malta Independent*, Aug. 30, 2023. [Online]. Available: <https://www.independent.com.mt/articles/2023-08-30/local-news/Franco-Debono-calls-on-Wasteserv-CEO-to-resign-after-suggestion-to-freeze-organic-waste-6736254464>. [Accessed: Jan. 16, 2025].
- [22] "There are other places apart from freezers to stuff our organic waste," *Malta Today*. [Online]. Available: https://www.maltatoday.com.mt/comment/blogs/124757/there_are_other_places_apart_from_freezers_to_stuff_our_organic_waste. [Accessed: Jan. 16, 2025].
- [23] "A messy situation: Ministry says Malta's waste is being separated but some organic refuse mixed with black bag waste," *Lovin Malta*. [Online]. Available: <https://lovinmalta.com/lifestyle/environment/a-messy-situation-ministry-says-maltas-waste-is-being-separated-but-some-organic-refuse-mixed-with-black-bag-waste/>. [Accessed: Jan. 16, 2025].
- [24] "Half a million grey recycling bags given away for free," *Times of Malta*. [Online]. Available: <https://timesofmalta.com/article/half-a-million-grey-recycling-bags-given-away-for-free.639920>. [Accessed: Jan. 16, 2025].
- [25] "Mixed waste collection decreases by a fifth," *Times of Malta*. [Online]. Available: <https://timesofmalta.com/article/mixed-waste-collection-decreases-fifth.1063962>. [Accessed: Jan. 16, 2025].
- [26] "White bag, black bag, bag: New waste collection schedule begins," *Times of Malta*. [Online]. Available: <https://timesofmalta.com/article/white-bag-black-bag-bag-new-waste-collection-schedule-begins.1004851>. [Accessed: Jan. 16, 2025].
- [27] "Use the Clean Up feature on your Mac," Apple Support, [Online]. Available: <https://support.apple.com/en-mn/guide/mac-help/mchl35c53342/mac#:~:text=You%20can%20use%20the%20Clean%20Up%20in%20the%20toolbar>. [Accessed: Jan. 17, 2025].
- [28] "Using Generative Erase in Microsoft Photos," Microsoft Support, [Online]. Available: <https://support.microsoft.com/en-us/windows/using-generative-erase-in-microsoft-photos-e0b4df42-3372-4dfd-9d28-c4ef408454a7#:~:text=Generative%20erase%20is%20an%20AI, based%20on%20the%20surrounding%20area>. [Accessed: Jan. 17, 2025].
- [29] "Using Object Eraser on your Galaxy device," Samsung Support, [Online]. Available: https://www.samsung.com/au/support/mobile-devices/using-object-eraser/?srsltid=AfmB0orIx-i300kBc0hZ0w3oVKYnoc6MgzJxxGOI1N0_pFb5pEo3xuFM. [Accessed: Jan. 17, 2025].

8 Appendix Generative AI - Journal

This Generative AI Journal explores how tools such as **GPT-4o**, **GPT o1**, **GPT o1-mini**, **Claude 3.5 Sonnet**, **Quillbot**, **NotebookLM**, **Gemini 2.0 Flash Experimental**, and **1.5 Flash** have been incorporated into the project. It examines their roles, methodologies, ethical implications, and personal perspectives, emphasizing their influence on productivity and the challenges posed by issues like bias and hallucinations. Through these discussions, the journal offers guidance on leveraging AI responsibly to meet the project's objectives.

8.1 Introduction

The generative AI space has evolved since the ChatGPT boom. A multitude of options now exist for users to generate ideas, documents, and code alike. Each team member has a different philosophy regarding generative AI and the tools they prefer to use for each use case.

When it came to developing an understanding of certain concepts or technologies, some team members turned to NotebookLM. NotebookLM is a generative AI research assistant built on Google's Gemini 2.0. NoteBookLM accesses documents, videos, audio files, and any other data you provide it and generates the key insights you would extract from consuming that data yourself, thus saving time and improving the fluidity of the team's workflow.

On the other hand, debugging was handled using a combination of Claude and ChatGPT. These tools have been commonly used for some time now and the newer versions excel in code debugging. Claude was primarily used for debugging, since it usually outperforms ChatGPT in that respect. However, ChatGPT, as well as Gemini, offer more versatility and were thus used for miscellaneous use cases such as verifying and correcting information and language, as well as generating test code and steering code approaches in the right direction when team members were stuck.

Finally, for the purely language and writing assistance, especially in the case of styling references, QuillBot provided solid support. Language models are able to generate consistent and error-free text, which helps especially in these tricky situations.

8.2 Ethical Considerations

Generative AI, known for its remarkable ability to create text, images, and diverse content, has emerged as an essential resource across multiple fields. Nonetheless, its implementation raises numerous ethical issues that need to be tackled to guarantee responsible and fair application.

8.2.1 Bias and Fairness

Generative AI models learn from extensive datasets that might harbor biases indicative of societal disparities. Such biases may appear in outputs, possibly strengthening stereotypes or unjust portrayals. For instance, language models may inadvertently generate biased language or reinforce discriminatory concepts. Addressing these problems necessitates careful dataset management, regular bias evaluations, and open development methods. Additionally, ongoing monitoring and feedback systems can assist in recognizing and resolving problematic behaviours.

8.2.2 Accuracy and Hallucination

A major problem of generative AI is its propensity to hallucinate—creating seemingly credible yet false or deceptive information. This presents dangers in areas where factual precision is essential, especially when it comes to assignments, where accuracy is crucial, relying on generative AI is not ideal or advised in this area, rather it should be used to supplement the content. Developers and users should prioritize verification methods and have a solid awareness of the AI's constraints. Human supervision is essential to guarantee the trustworthiness of information produced by these systems.

8.2.3 Accountability and Governance

Accountability systems need to be implemented to tackle ethical violations. This entails establishing governance frameworks to manage AI development and implementation, involving stakeholders from various backgrounds, and promoting interdisciplinary discussions. Developers, policymakers, and users collectively bear the responsibility of ensuring that AI technologies are utilized in an ethical and responsible manner.

Generative AI possesses significant potential to revolutionize sectors and improve human abilities. Nonetheless, identifying and tackling its ethical consequences is essential for harnessing its advantages while reducing harm. By promoting transparency, equity, and responsibility, we can shape a future where generative AI has a beneficial impact on society while maintaining fundamental ethical standards.

8.3 Methodology

The integration of the use of GenAI varied among team members, with different individuals employing tools uniquely based on their specific requirements. However, the following general outline provides a structured approach to how GenAI tools were utilized, the sequence of their usage, and their contributions to the project.

8.3.1 Research Phase

During the research phase, **NotebookLM** was pivotal in exploring and extracting insights from academic papers. Team members used it to pose questions and generate podcasts, aiding in auditory comprehension of complex topics. This process refined the project scope and helped prioritize key literature.

Once relevant academic papers were identified, they were uploaded to **Quillbot** to generate references in IEEE 2018 style. These references were then integrated into the Overleaf document to maintain consistent formatting throughout the project.

8.3.2 Code Development

The foundation of the code was initially developed independently to ensure a deeper understanding of the implementation. Once a functional codebase was achieved, **GPT-4o** and **Gemini 1.5/2.0 Flash Experimental** were used for code improvement. These tools optimized performance, improved user interactions, and provided in-line comments for better clarity and maintainability.

In cases where errors were encountered, **Claude 3.5-Sonnet** and **GPT-o1/o1-Mini** were utilized to efficiently identify and resolve issues, streamlining the debugging process. Additionally, **Claude 3.5-Sonnet** played a key role in determining the appropriate hyper-parameters for the model, ensuring optimal performance.

8.3.3 Results Validation

To ensure the validity of results, output images generated during the project were uploaded to **GPT-4o** for secondary verification. This step ensured that the outputs aligned with the project's objectives and met expected quality standards. Feedback from the AI further validated the accuracy and relevance of the results.

8.3.4 Documentation and Finalization

Markdown content in Jupyter notebooks was refined and polished using these tools. Additionally, the final project report underwent a peer review process using **GPT-4o** and **Gemini 2.0 Flash Experimental**, which provided critical feedback to ensure all requirements were met and highlighted areas for improvement. This process contributed to a comprehensive and well-rounded final submission.

8.3.5 Pipeline Overview

The overall process incorporated independent development supplemented by generative AI tools at various stages. **Gemini 1.5/2.0 Flash Experimental** and **GPT-4o** enhanced the efficiency of code optimization and documentation, while **Claude 3.5-Sonnet**, **GPT-o1**, and **o1-Mini** streamlined debugging and model tuning. This iterative pipeline ensured both a deeper understanding of the project implementation and enhanced efficiency in achieving project goals.

8.4 Prompts and Responses

8.4.1 Andrea Filiberto Lucas

Prompt 1: "What architectural innovations in YOLOv8 improve accuracy and speed for object detection?" (Fig. 21)

Model Used: NotebookLM

Reflection: NotebookLM helped in understanding basic concepts about YOLOv8 by providing detailed answers based on previously found sources, making it easier to grasp how the model addresses small or overlapping object detection challenges.

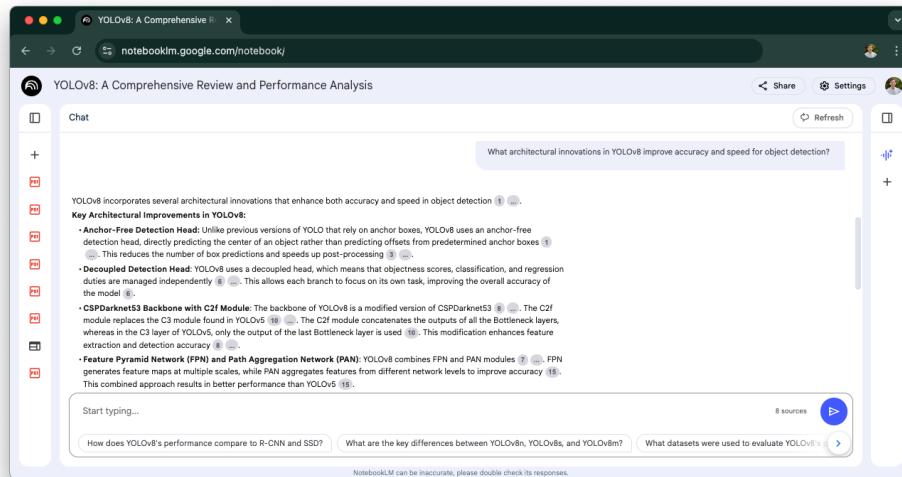


Figure 21: Prompt 1 - NotebookLM

Prompt 2: "Generate reference in IEEE 2018 (with url) style." (Fig. 22)

Model Used: QuillBot

Reflection: The GenAI provided well-formatted references in IEEE 2018 style, which helps maintain consistency across all citations in the report.

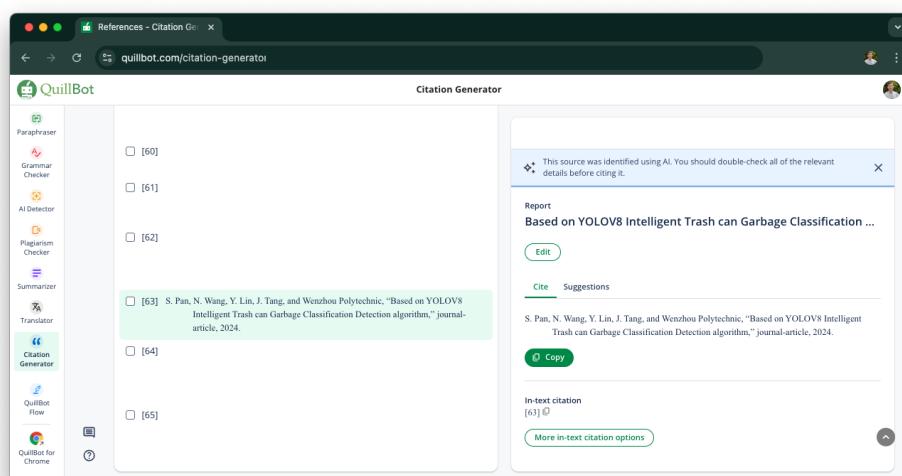


Figure 22: Prompt 2 - Quillbot

Prompt 3: "Add in-line comments to the following code snippet: Snippet" (Fig. 23)

Model Used: ChatGPT-4o

Reflection: GPT-4o added in-line comments to the code snippet, enhancing clarity and making it easier to understand the purpose and functionality of the snippet.

The screenshot shows a ChatGPT interface with a code editor window titled "Folder Setup Script". The code is a Python script for creating a folder structure and installing Roboflow. ChatGPT has added in-line comments to explain each step. The code includes imports for os, subprocess, and shutil, defines colors for console output, and handles folder creation and Roboflow installation. A message at the bottom says "ChatGPT can make mistakes. Check important info."

```
1 import os
2 import subprocess
3 import shutil
4
5 COLORS = {
6     "green": "\033[92m",
7     "red": "\033[91m"
8     "reset": "\033[0m"
9 }
10
11 versions_path = os.path.abspath(os.path.join(".", "Versions"))
12 target_subfolder = os.path.join(versions_path, "yolo-AFL-Yolov8")
13
14 if not os.path.exists(versions_path):
15     os.makedirs(versions_path)
16     print(f"\033[92m{COLORS['green']}{{}} Folder created at: {versions_path}{COLORS['reset']}\033[0m")
17
18 if os.path.exists(target_subfolder):
19     print(f"\033[91m{COLORS['red']}{{}} The subfolder '{target_subfolder}' already exists. Skipping.\033[0m")
20 else:
21     if importlib.util.find_spec("roboflow") is not None:
22         print(f"\033[92m{COLORS['green']}{{}} Roboflow is already installed!\033[0m")
23     else:
24         try:
25             subprocess.check_call(["pip", "install", "roboflow"])
26             print(f"\033[92m{COLORS['green']}{{}} Roboflow successfully installed!\033[0m")
27         except subprocess.CalledProcessError as e:
28             print(f"\033[91m{COLORS['red']}{{}} Failed to install Roboflow. Please check your setup.\033[0m")
29
30 from roboflow import Roboflow
31
32 print("Please enter your Roboflow API key to download the dataset...")
33 api_key = input("Please enter your Roboflow API key: ")
34
35 rf = Roboflow(api_key)
36
37 project = rf.workspace("advanced-cv").project("maltese-domestic-dataset")
38 version = project.version()
39
40 dataset = version.download("yolov8n")
```

Figure 23: Prompt 3 - GPT-4o

Prompt 4: "Create another Iteration to this experiment by changing the hyper-parameters results.png, Code Snippet" (Fig. 23)

Model Used: Gemini 2.0 Flash Experimental

Reflection: Gemini suggested changing the learning rate (lr0) to 0.0001, simplifying the process of exploring hyper-parameter adjustments and enabling a more focused experiment iteration.

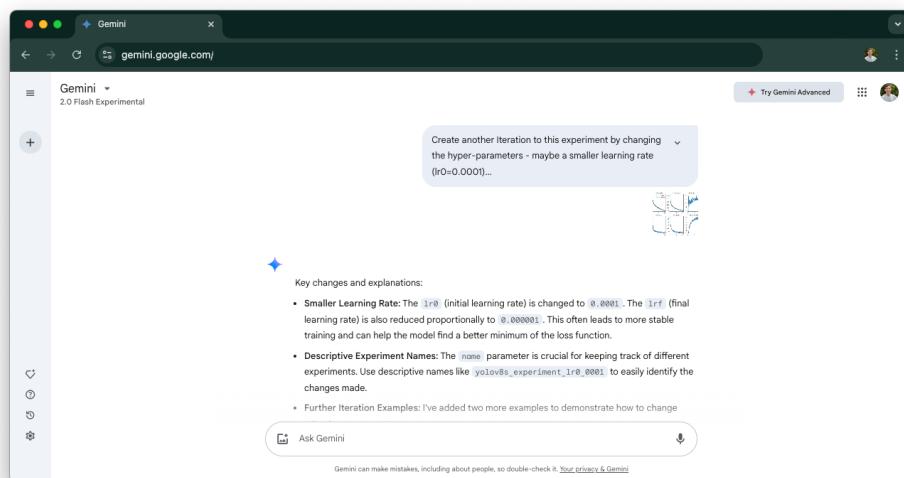


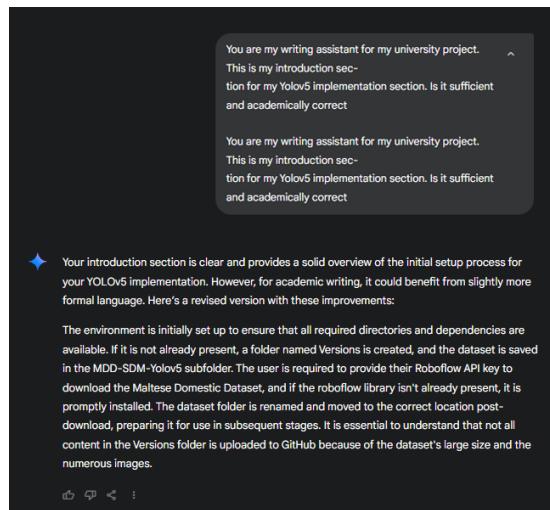
Figure 24: Prompt 4 - Gemini 2.0 Flash Experimental

8.4.2 Sean David Muscat

Prompt 1: You are my writing assistant for my university project. This is my introduction section for my Yolov5 implementation section. Is it sufficient and academically correct? {Introduction Paragraph}

Gemini Response: Your introduction section is clear and provides a solid overview of the initial setup process for your YOLOv5 implementation. However, for academic writing, it could benefit from slightly more formal language. Here's a revised version with these improvements: {New Introduction Paragraph}

How this helped: This was very helpful to make sure all my writing was consistently academic, and that I have not missed anything crucial.



Prompt 2: Can you create a check that my yolov5 model is being found correctly? {yolo path code}

GPT-4o Response: Here's a code snippet to check if the YOLOv5 model configuration file (yolov5n.yaml) is being found correctly: {code for checks}

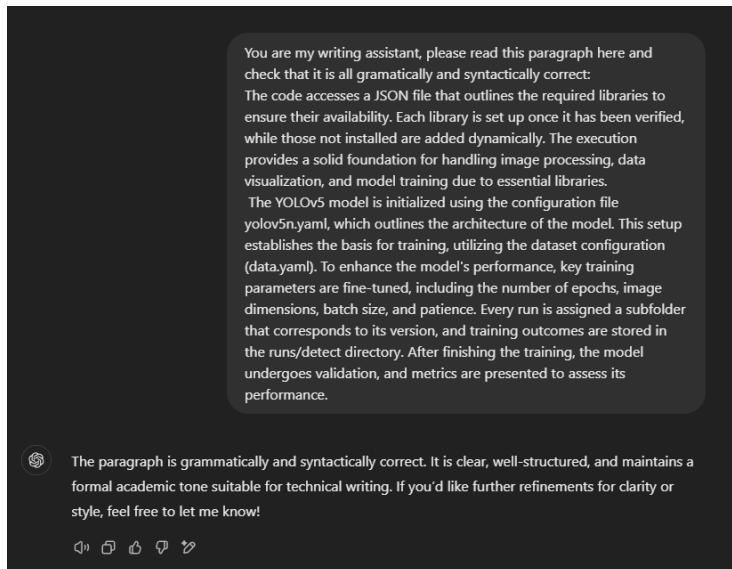
How this helped: This response helped verify the file paths for all of my data such as my data.yaml or model location, especially during late nights when the error was not that easy to find.

A screenshot of the GPT-4o interface. The user has asked for a code snippet to check if the YOLOv5 model configuration file (yolov5n.yaml) is being found correctly. The AI has provided a Python script that defines a function to check if the model config file exists at a specified path. The code uses the os.path module to construct the path and check for existence. It also includes a print statement for success and an error message for failure. The code is presented in a code editor window with syntax highlighting for Python.

Prompt 3: You are my writing assistant, please read this paragraph here and check that it is all grammatically and syntactically correct: paragraph

GPT-4o Response: The paragraph you provided is grammatically and syntactically correct. It is clear, coherent, and effectively communicates the intended information. If you have any specific concerns or would like stylistic improvements, feel free to let me know!

How this helped: Similarly to the first prompt, this served to make sure that all my writing was good, and this time instead of checking if I missed anything, making sure my sentence structure and other basic English structure was in place.



Prompt 4: Add comments to this code: {Uncommented yolov5 training code}

GPT-4o Response: Here's the code with comments added: {Commented code} This version adds comments to explain each line and parameter, ensuring clarity for anyone reviewing or modifying the code. Let me know if you need further assistance!

How this helped: This was very helpful, since I had multiple runs and they were days apart, I could go back and not forget everything, so having gen AI automate that process and make it easier to go back and recall made it better.

The screenshot shows a code editor with two panes. The left pane contains the original uncommented Python code for training a YOLOv5 model. The right pane shows the same code with extensive comments added to explain each line and parameter. A note at the bottom of the right pane states: "This version adds comments to explain each line and parameter, ensuring clarity for anyone reviewing or modifying the code. Let me know if you need further assistance!"

```
data_yaml_path = os.path.join(os.path.abspath(os.path.join(os.getcwd(), os.pardir)), "Versions", "MDD-SDM-Yolov5", "data.yaml")

results = model.train(
    data=data_yaml_path,
    epochs=100,
    imgsz=640,
    batch=16,
    workers=4,
    device='cpu',
    project='runs/detect',
    name='MDD-SDM-Yolov5-v3',
    patience = 10,
    pretrained = False
)

print(f"[✓] Training completed! Results saved in: {results.save_dir}")
metrics = model.val()
print("Validation Metrics:", metrics)
```

```
# Path to the data.yaml file, which contains the dataset configuration
data_yaml_path = os.path.join(os.path.abspath(os.path.join(os.getcwd(), os.pardir)), "Versions", "MDD-SDM-Yolov5", "data.yaml")

# Train the YOLO model with the specified parameters and save results in runs/detect
results = model.train(
    data=data_yaml_path, # Path to the dataset configuration file
    epochs=100, # Number of training epochs
    imgsz=640, # Input image size for training
    batch=16, # Batch size for training
    workers=4, # Number of worker threads for data loading
    device='cpu', # Specify device for training ('cpu' or GPU indices like '0')
    project='runs/detect', # Directory to save the results of the training
    name='MDD-SDM-Yolov5-v3', # Name of the specific training run
    patience=10, # Number of epochs to wait for improvement before early stop
    pretrained=False # Whether to use pretrained weights (False for training from scratch)
)

# Print a success message with the location of the training results
print(f"[✓] Training completed! Results saved in: {results.save_dir}")

# Validate the model on the dataset and print validation metrics
metrics = model.val()
print("Validation Metrics:", metrics)
```

8.4.3 Thomas Koppens

Prompt 1: "PyTorch resnet vs resnet_v2"

How this helped: PyTorch's documentation does not actually explain the difference between these two. ChatGPT provided what the actual difference was.

PyTorch resnet vs resnet_v2

In PyTorch, the difference between `ResNet` and `ResNet_v2` implementations lies primarily in the ordering of operations within their residual blocks, which affects the overall training behavior and performance. Here's a detailed comparison:

ResNet (Original ResNet)

- Proposed in:** "Deep Residual Learning for Image Recognition" (He et al., 2015).
- Block Structure:**
 - Order of Operations:** Convolution → Batch Normalization → ReLU.
 - Each residual block has the following structure:
$$y = \text{ReLU}(\text{BN}(\text{Conv}(x)))$$
 - Skip connection is added after the final operation in the block.
- Training Characteristics:**
 - Works well for a wide range of tasks.
 - Can suffer from exploding/vanishing gradients for very deep networks (e.g., 1000+ layers).
- Common Implementations in PyTorch:**
 - `torchvision.models.resnet18`, `resnet34`, `resnet50`, etc.

ResNet_v2 (Pre-activation ResNet)

- Proposed in:** "Identity Mappings in Deep Residual Networks" (He et al., 2016).
- Block Structure:**
 - Order of Operations:** Batch Normalization → ReLU → Convolution.
 - Each residual block has the following structure:
$$y = \text{Conv}(\text{ReLU}(\text{BN}(x)))$$
 - Skip connection is added directly to the input, making it identity-preserving.
- Advantages:**
 - Improved gradient flow due to the pre-activation design.
 - Better training stability, especially for very deep networks (e.g., 1000+ layers). \downarrow

Write an introduction to

① 📄 ⊕ ⌂ ↑

Prompt 2: <right after prompt 1> "What should I use for retinanet"

How this helped: This response quickly summarised the variables I should account for when determining which version of ResNet I should use, leading to quick decision making.

The screenshot shows a dark-themed AI-generated response. At the top right, there is a rounded rectangular button labeled "What should I use for retinanet". Below it, a circular icon with a gear symbol is followed by text: "For implementing RetinaNet, the choice between ResNet and ResNet_v2 as the backbone largely depends on your goals and constraints, such as performance, stability, and ease of integration. Here's a breakdown to help you decide:"

RetinaNet Overview

RetinaNet is a one-stage object detection model that uses a backbone network (e.g., ResNet) for feature extraction and adds Feature Pyramid Networks (FPN) for multi-scale feature detection. The backbone plays a critical role in determining the model's feature quality and performance.

Using ResNet as the Backbone

1. **Advantages:**
 - Widely used in RetinaNet implementations, including in popular frameworks like PyTorch and Detectron2.
 - Pretrained weights are readily available in `torchvision`, making it easy to integrate.
 - Sufficiently stable for training on common object detection tasks.
2. **Recommended Scenarios:**
 - If you're working with moderate-sized datasets or standard tasks like COCO.
 - When ease of use and pretrained weights are important.
 - For models like ResNet50 or ResNet101, which balance performance and computational cost.
3. **Implementation:** You can use pretrained ResNet models from `torchvision` and plug them into RetinaNet.

Prompt 3: "Turn this data into a latex table

Mixed Waste & Organic Waste & Recyclable Material & Other Waste
51.6% & 43.8% & 68.8% & 23.5%"

How this helped: This quickly generated the table for me with styling applied. No LATEXtable generator would apply appropriate styling or infer which value should be placed in which slot.

Turn this data into a latex table

Mixed Waste & Organic Waste & Recyclable Material & Other Waste \\ \ 51.6\% & 43.8\% & 68.8\% & 23.5\%

Here is the LaTeX code to create a table for the given data:

```
latex
\documentclass{article}
\usepackage{booktabs}

\begin{document}

\begin{table}[h!]
\centering
\begin{tabular}{@{}lcccc@{}}
\toprule
Category & Mixed Waste & Organic Waste & Recyclable Material & Other Waste \\
\hline
Percentage (%) & 51.6 & 43.8 & 68.8 & 23.5 \\
\end{tabular}
\caption{Percentage distribution of waste categories.}
\label{tab:waste_distribution}
\end{table}

\end{document}
```

Key Points:

1. The `\toprule`, `\midrule`, and `\bottomrule` commands are used for clean table borders.
2. The `@{}` in the `tabular` environment removes extra padding around the columns.
3. The `\caption` command provides a description of the table, and `\label` assigns a reference for cross-referencing.

Copy and paste this code into a `.tex` file and compile it with a LaTeX editor (e.g., Overleaf, TeXShop) to generate the table.

⟳ ⌂ ⌄ ⌅ ⌆ ⌇ ⌈ ⌉

8.5 Improvements, Errors, and Contributions of GenAI

Generative AI is popular not only because it makes life simpler and more efficient, but also because it really can improve the quality of work. In our case, this mostly means using Gen AI as a reviewer, meaning that the tool reviews text and determines how well the intended message is conveyed, how well the text is structured, and how appropriate the language used is.

This can be seen in section 8.4.2 – Sean David Muscat’s prompt 3, which prompts ChatGPT to review the grammar and structure of the inserted paragraph. ChatGPT gives a clear overview of the paragraph, highlighting what was done right and reassuring the prompter of the validity of the text.

On the other hand, generative AI is always going to have a difficult time in rarely seen problems. Working on natural language such as generating paragraphs, tweaking the language used in them, or analysing the key points within a text are simple, commonly seen problems for it. The more error prone tasks include the debugging of code snippets. Naturally, if it is difficult for a developer to solve a code problem themselves, even with the support of online resources, then it should be more difficult for generative AI to solve it. Thus, the use of generative AI for debugging is a last resort for team members which is usually done with a pessimistic attitude – that is assuming that the Large Language Model (LLM) is going to over-engineer a complex mess of a ‘solution’, which is often the case. Nevertheless, the solutions are attempted and new errors are fed back to the LLM until a solution is found or patience runs out. Of course, in the case an appropriate solution is found, it is very important that time is dedicated to properly understanding *why* the solution works and what you missed when attempting to resolve the issue yourself.

A good example of this is the struggle with generating a confusion matrix in the RetinaNet notebook which could be compared to the YOLO models. Neither Claude nor ChatGPT succeeded in finding a solution for this problem, despite prompting it with multiple different potential approaches and pointing out clear errors it was making. Since the confusion matrix was never actually successful, it is not in the prompts section. However, the conversation can be found here.

8.6 GenAI Individual Reflections

8.6.1 Andrea Filiberto Lucas

Generative AI proved to be an excellent assistant throughout the project, particularly in determining hyper-parameters and learning how to use Ultralytics effectively. I learned how to approach hyper-parameter tuning in a more systematic way, as the AI provided insights based on the experimental results I shared with it. This was especially surprising, as it highlighted patterns I might have overlooked. GenAI was also very helpful for tasks like inline code suggestions, general peer review, and structuring settings, which made the workflow more efficient and streamlined.

However, it wasn't without its challenges. A significant issue was the AI's tendency to hallucinate, often giving incorrect answers or making unrealistic suggestions. This made me realize the importance of double-checking its outputs and relying on my own skills for critical tasks, such as coding and debugging. While AI saved time in some areas, it also occasionally led to wasted time when inaccurate information had to be corrected. Despite this, I found its greatest strengths in assisting with interpreting results and improving the structure of settings, rather than in tasks like literature review or direct coding. Overall, while my perspective on generative AI hasn't drastically changed, I now see it as a supportive tool that needs careful oversight rather than a replacement for hands-on problem-solving.

8.6.2 Sean David Muscat

Overall, Generative AI is a valuable asset when working, and it has enhanced my workflow immensely throughout. I feel like it was time saving at points, especially when debugging specific errors which I would have spent a lot longer on without it. It also helps as a writing assistant, since I struggle with writing at length at times, GenAI does help a lot when it comes to steering me back on the right path. My stance on GenAI in academic projects has not changed overall, it should be used to support oneself, not overshadow or remove other learning methods. It should work alongside you not stifling your own creativity and thought process.

8.6.3 Thomas Koppens

As stated in previous years, I prefer not to use Generative AI. I certainly do use it, but only as a last resort. During this project, I ran into many walls, and thus I had to turn to generative AI on more than one occasion. However, it generally proved to be futile as my issues were too rarely encountered for it to find proper solutions. I also found it dangerous whenever it would find a 'solution', which worked on the surface level but did not actually work as expected, using the wrong values under the hood. Thus, in the aforementioned case of the confusion matrix, matrices were indeed generated, but upon inspection could not possibly be correct. However, in the cases of tedious work such as the generation of the latex table, it saved time and generated satisfying results. While latex table generators exist online, GenAI will style it based on the most common standards by nature of how it works. Furthermore, GenAI will practically always give the high quality, in-depth answers for understanding topics such as the ResNet backbone question. When I know that the question is a bit too specific, I turn to GenAI – keeping in mind that the information it provides could be wrong. Regarding writing and literature reviews, I like to do those myself, otherwise I do not feel like I properly understand what I am doing. This approach is consistent with my attitude towards GenAI in previous years, and thus my perspective on how it is used in my academic projects has not significantly changed. I still believe it should be used only as a last resort when it comes to code, and only in certain situations with specific contexts for research and text generation.

8.7 List of GenAI Resources Used

- [1] "Quillbot Citation Generator," [Online]. Available: <https://quillbot.com/citation-generator>. [Accessed: Jan. 14, 2025].
 - [2] "Hello GPT-4o," OpenAI, [Online]. Available: <https://openai.com/index/hello-gpt-4o/>. [Accessed: Jan. 14, 2025].
 - [3] "OpenAI O1," OpenAI, [Online]. Available: <https://openai.com/o1/>. [Accessed: Jan. 14, 2025].
 - [4] "OpenAI O1 Mini: Advancing Cost-Efficient Reasoning," OpenAI, [Online]. Available: <https://openai.com/index/openai-o1-mini-advancing-cost-efficient-reasoning/>. [Accessed: Jan. 14, 2025].
 - [5] "Claude AI," [Online]. Available: <https://claude.ai/new>. [Accessed: Jan. 14, 2025].
 - [6] "Notebook LM," Google, [Online]. Available: <https://notebooklm.google/>. [Accessed: Jan. 14, 2025].
 - [7] Google, "Gemini Application," [Online]. Available: <https://gemini.google.com/app>. [Accessed: 23-Jan-2025].
-