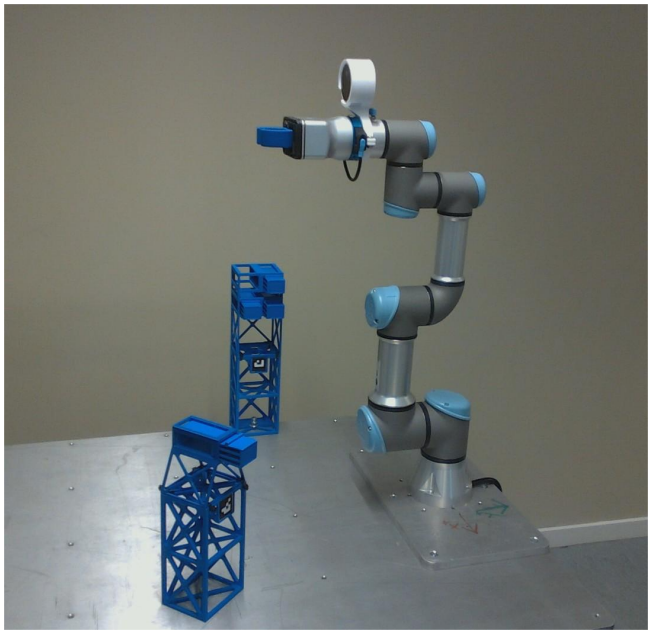
NTNU
Kunnskap for en bedre verden

Portfolio Assignment 3



Object Detection and Classification for Robotic Arm's Pick and Place Working Environment

AIS4002 - Intelligent Machines

Jon Urcelay San Roman

22nd April 2025

Contents

List of Figures iii

List of Tables iii

1 Introduction 2

2 Objective for the Vision Model 3

2.1 Task 1: Drone or Station . . . 3

2.2 Task 2: Hole Classification . . . 3

3 Model Theory 5

3.1 YOLO . . . 5

3.1.1 Output tensor and predictions . . . 5

3.1.2 Non-Maximum Suppression (NMS) . . . 5

3.1.3 Architecture and backbone . . . 6

3.1.4 Multi-Scale Feature Extraction . . . 7

3.1.5 Summary . . . 8

3.2 Faster R-CNN . . . 9

3.2.1 Feature Extraction and Backbone . . . 9

3.2.2 Region Proposal Network (RPN) . . . 9

3.2.3 RoI Pooling and Classification . . . 10

3.2.4 Non-Maximum Suppression (NMS) . . . 11

3.2.5 Multi-Scale Processing . . . 11

3.2.6 Summary . . . 12

4 Dataset 13

4.1 Images . . . 13

4.2 Annotations for task 1: drone or station . . . 13

4.3 Annotations for task 2: hole classification 14
4.4 Splits 15

5 Model Training **16**

6 Model Results **17**

6.1 Training results 17
6.2 Accuracy 20
6.3 Robustness 23
6.4 Real time detection 25

7 Feature map analysis for best model **27**

List of Figures

1 Working environment for UR3e robotic arm with L515 camera. 2
2 Desired output for Task 1. 3
3 Desired output for Task 2. 4
4 Examples of raw images used in the dataset. 13
5 Total training loss for YOLO variants in Task 1 and Task 2. 17
6 Total validation loss for YOLO variants in Task 1 and Task 2. 18
7 Total training and validation loss for Faster R-CNN in Task 1 and Task 2. . . . 18
8 Precision for all the models for Task 1 and Task 2. 19
9 Recall for all the models for Task 1 and Task 2. 19
10 mAP@0.5:0.95 for all the models for Task 1 and 2. 20
11 Confusion matrices on Task 1. 21
12 Confusion matrices on Task 2. 22
13 Robustness tests for task 1. 24
14 Robustness tests for task 2. 25

List of Tables

1 Comparison of model complexity for YOLOv8 variants and Faster R-CNN (ResNet-50 backbone), showing the number of parameters (in millions) and layers. 16
2 Real-time detection metrics for Task 1 on ASUS Zenbook, grouped by hardware-agnostic (model size, MACs) and hardware-dependent (inference time, FPS, CPU utilization, RAM and power) categories. 26
3 Real-time detection metrics for Task 2 on ASUS Zenbook, grouped by hardware-agnostic (model size, MACs) and hardware-dependent (inference time, FPS, CPU utilization, RAM and power) categories. 26

Acronyms

NN Neural Network.

YOLO You Only Look Once.

R-CNN Region-based Convolutional Neural Network.

IoU Intersection over Union.

NMS Non Maximum Suppression.

CNN Convolutional Neural Network.

CSPNet Cross-Stage Partial Network.

FPN Feature Pyramid Network.

PANet Path Agregation Network.

RPN Region Proposal Network.

ResNet Residual Network.

VGG-16 Visual Geometry Group - 16 Layers.

RoI Regions of Interest.

SGD Stochastic Gradient Descent.

CPU Central Processing Unit.

GPU Graphics Processing Unit.

mAP Mean Average Precision.

FPS Frames Per Second.

MACs Multiply-Accumulate Operations.

1 Introduction

A group of students is working on a project where they want to recreate a drone battery swap station. For this task, they are using a UR3e robotic arm with a Realsense L515 camera attached to the robots gripper.

As the project itself is very complex, they will focus on the robotic arm's pick and place functionality, working with a simplified environment.

This working environment can be observed in Figure 1, which consists of the following parts:

- **Batteries.** 3D printed objects that represent batteries. These have a clip type structure which allows the robot to grab them and take them in & out from any compartment.
- **Drone.** 3D printed structure that represents a drone that has landed in the station. To avoid complexity, its structure has been simplified, containing just the compartment for the battery. The drone's position and orientation is variable in the landing area.
- **Charging station.** 3D printed structure that represent the charging station for the batteries. It has 4 compartments, one for each battery.

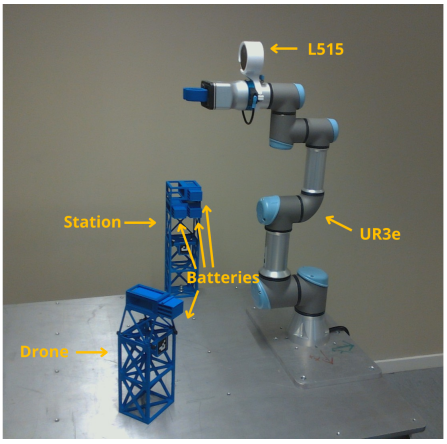


Figure 1: Working environment for UR3e robotic arm with L515 camera.

The workflow for the robotic arm can be described as follows:

- Pick up the battery from the drone.
- Place it in an empty compartment in the charging station.
- Pick up a new battery from the charging station.
- Place it in the drone.

2 Objective for the Vision Model

For achieving the pick and place task with the UR3e robotic arm, training a deep neural network for object detection is been considered.

For the purpose of this assignment, two different tasks will be completed for object detection&classification, the first one being more simple (Section 2.1) and the second more complex (Section 2.2).

For each of the tasks, different neural networks (NN) will be trained and evaluated. The models to compare are **YOLO** and **Faster R-CNN**. The models will be explained in Chapter 3, the training details in Chapter 5 and the results shown in 6.

Finally, the images used for training in both tasks will be the same, but the annotations will be different. This will be explained in Chapter 6.

2.1 Task 1: Drone or Station

This task consists in identifying the structure in the image as a drone or a station, and capturing it in a bounding box. See Figure 2 for the desired output for this task.

Figure 2: Desired output for Task 1.

2.2 Task 2: Hole Classification

This task consists in identifying the compartments (holes) in the image, classifying them as empty or full, and also as on of the following: drone hole, station top left hole, station top right hole, station bottom left hole or station bottom right hole.

See Figure 3 for the desired output for this task.

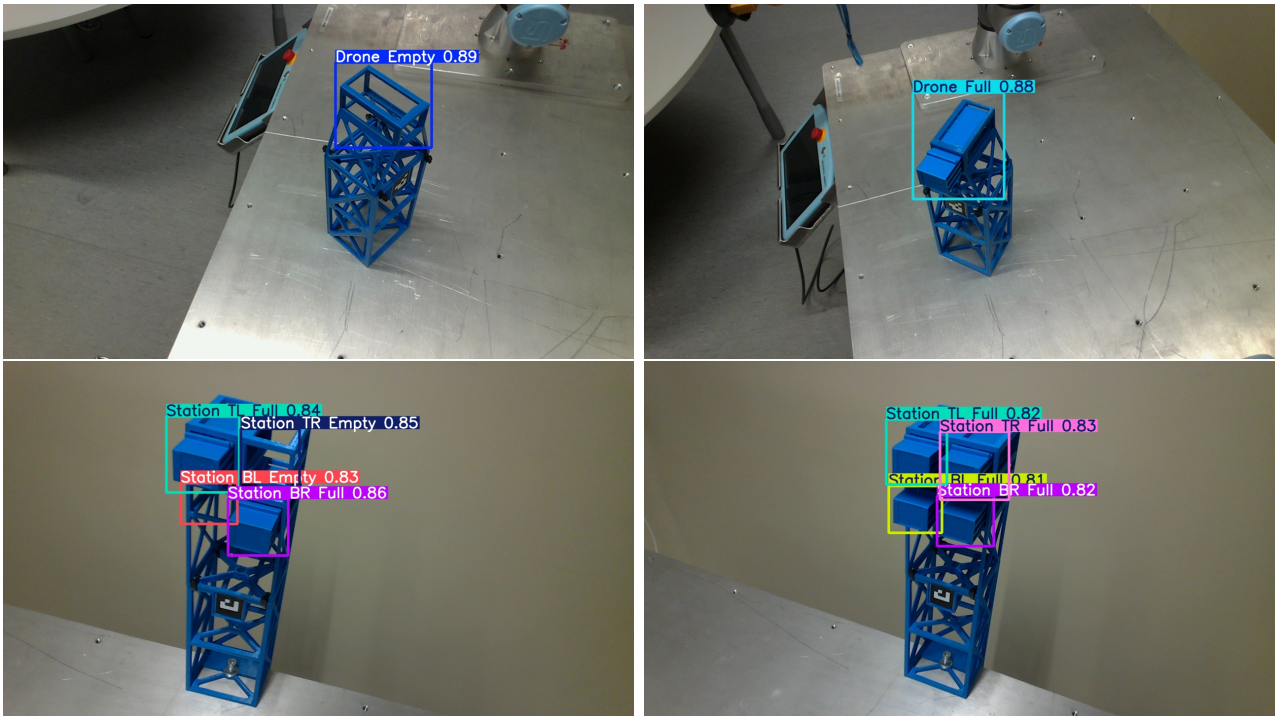


Figure 3: Desired output for Task 2.

3 Model Theory

This chapter will briefly explain the theory behind each of the NN models to be used in this assignment, **YOLO** and **Faster R-CNN**.

3.1 YOLO

YOLO (You Only Look Once) is a single-stage object detection model designed for real-time processing, treating detection as a regression problem. Unlike traditional approaches that scan images multiple times, YOLO processes an entire image in a single forward pass through a convolutional neural network (CNN). This is the reason for its name, as it only passes once.

3.1.1 Output tensor and predictions

The input image is divided into a $S \times S$ size grid. Each grid cell predicts:

- **Bounding Boxes.** Coordinates (x, y, width, height) for potential objects.
- **Confidence scores.** A value between 0 and 1, representing the probability that the box contains an object multiplied by the Intersection over Union (IoU) between the predicted box and the ground truth (if any):

$$Confidence = P(Object) * IoU_{pred,truth}$$

- **Class probabilities.** Likelihood of the object belonging to specific classes, like dog, cat, cat, airplane, etc.

The network outputs a tensor that combines these predictions, which size is $S \times S \times B \times (5+C)$, where $S \times S$ is the grid size, B is the number of bounding boxes per cell, 5 accounts for the box coordinates (x, y, w, h) + the confidence score, and C is the number of classes.

However, modern YOLO versions (YOLOv5, YOLOv8) produce outputs at multiple grid sizes, resulting in multiple tensors that collectively cover objects of various sizes. These tensors combine all predictions into a single structure, which is then processed to generate the final detections.

3.1.2 Non-Maximum Suppression (NMS)

Since YOLO predicts multiple bounding boxes for the same object (from the different grid cells), many of these boxes may overlap and represent the same object. NMS is a post-processing step that eliminates redundant or low-confidence boxes to produce a clean set of detections. This is how NMS works in YOLO:

- **Step 1: Filter by confidence.** All predicted boxes with a confidence score below a threshold (e.g., 0.5) are discarded to remove low-likelihood detections.
- **Step 2: Select the best box.** For each class, the box with the highest confidence score is selected.
- **Step 3: Suppress overlapping boxes.** For the selected box, any other boxes of the same class with an IoU above a threshold are removed. IoU measures the overlap between two boxes as:

$$IoU = \frac{Area_{Intersection}}{Area_{Union}}$$

- **Step 4: Repeat.** The process is repeated for the remaining boxes, selecting the next highest-confidence box and suppressing overlaps, until no boxes remain or all are processed.

NMS ensures that each object is represented by a single, high-confidence bounding box, reducing clutter in the final output. In YOLO, NMS is applied across all predicted boxes from all grid cells and scales.

3.1.3 Architecture and backbone

YOLO's architecture is built around a **backbone**, a deep CNN that extracts features from the input image. The backbone is responsible for transforming the raw pixel data into a rich feature representation that captures both low-level details (edges, textures, etc.) and high-level semantics (object shapes, categories, etc.). Two common backbones in YOLO variants are:

- **Darknet.** Custom CNN architecture designed by the YOLO authors. For example, Darknet-53 (used in YOLOv3) consists of 53 convolutional layers with residual connections (inspired by ResNet) to enable deeper networks without vanishing gradients. It uses operations like 3×3 and 1x1 convolutions, batch normalization, and Leaky ReLU activations to extract features efficiently.
- **CSPNet (Cross-Stage Partial Network).** Used in later versions like YOLOv5 and YOLOX, CSPNet improves computational efficiency and gradient flow. It splits the feature map into two parts: one part undergoes heavy processing (through convolutional layers), while the other is lightly processed. These parts are later merged, reducing redundancy and improving performance on resource-constrained devices.

As the image passes through the backbone's layers, its resolution, channel count, and semantic understanding evolve. Each layer produces feature maps with dimensions $H \times W \times C$, where $H \times W$ is the spatial resolution and C is the number of channels. Each channel encodes a distinct pattern, such as edges, textures, or object shapes, enabling the network to capture diverse visual features.

Early layers, closer to the input, apply small convolutional filters (like 3×3) to generate high-resolution feature maps with fewer channels. These maps capture low-level features like edges, corners, and textures, ideal for detecting small objects due to their fine spatial details but limited semantic context.

Deeper layers downsample feature maps (like 13×13) through strided convolutions or pooling, increasing the number of channels and the receptive field. These low-resolution, high-channel-count maps encode high-level patterns, such as object shapes or categories, making them better suited for detecting large objects.

In conclusion, the backbone transforms the input image into feature maps with varying resolutions and channel counts, which the **neck** and **head** of YOLO process to generate the output tensor for object detection.

3.1.4 Multi-Scale Feature Extraction

The **neck** of YOLO's architecture aggregates feature maps from different backbone layers to create a unified, multi-scale feature representation.

A key component here is the Feature Pyramid Network (FPN), which is widely used in YOLOv3 and later versions. FPN works as follows:

- **Top-down pathway.** FPN starts with the deepest, lowest-resolution feature map from the backbone, which is rich in semantic information. This map is upsampled (using for example nearest-neighbor interpolation) to match the resolution of a higher-resolution feature map.
- **Lateral connections.** The upsampled feature map is combined with the corresponding backbone feature map of the same resolution via element-wise addition or concatenation. Before combining, the backbone feature map is processed with a 1×1 convolution to align the channel dimensions. This fusion injects semantic information from deeper layers into the higher-resolution map.
- **Iterative process.** This process repeats, upsampling each layer's feature map and combining it with the feature map in the next level, creating a pyramid of feature maps at multiple scales.

Some YOLO variants, like YOLOv5, also incorporate a bottom-up pathway (Path Aggregation Network or PANet), where high-resolution features are downsampled and fused back into lower-resolution maps, further improving cross-scale information flow.

YOLO uses multiple detection **heads** to make predictions at different scales, leveraging the multi-scale feature maps from the FPN. Each detection head is a set of convolutional layers that processes a specific feature map to produce the output tensor for that scale. Typically, YOLO predicts at three scales, each tailored to different object sizes:

- Large-Scale Head: This head operates on the highest-resolution feature map. The grid cells here are small, so each cell predicts boxes suited for small objects.
- Medium-Scale Head: This head uses a medium-resolution feature map and detects medium-sized objects.
- Small-Scale Head: This head processes the lowest-resolution feature map and detects large objects. The grid cells are coarse, covering large portions of the image, and the feature map's semantic richness helps identify large, prominent objects.

Each detection head outputs a tensor with predictions for bounding box coordinates, confidence scores, and class probabilities, as described earlier. By distributing detection across multiple scales, YOLO ensures that small, medium, and large objects are handled by the appropriate head, improving overall robustness.

To further adapt to objects of different sizes and shapes, YOLO uses anchor boxes, which are predefined rectangular templates with specific sizes and aspect ratios. Each detection head is assigned a set of anchor boxes tailored to the expected object sizes at its scale.

These anchor boxes are typically determined during training on the ground-truth bounding boxes in the dataset, ensuring they match the dataset's object size distribution.

By associating anchor boxes with specific scales, YOLO ensures that each detection head focuses on objects of appropriate sizes, improving detection accuracy across diverse object scales.

3.1.5 Summary

YOLO is a single-stage object detection model, processing an entire image in a single forward pass through a CNN.

YOLO divides the image into a grid, where each cell predicts bounding box coordinates, confidence scores and class probabilities. These predictions are encoded in a tensor, typically across multiple scales, and filtered using NMS to eliminate redundant boxes.

The architecture leverages a backbone to extract features, with a FPN in the neck fusing high-resolution and low-resolution feature maps to detect objects of varying sizes. Multiple detection heads, each using anchor boxes tailored to specific scales, ensure robust detection across small, medium, and large objects.

YOLO's strength lies in its speed and efficiency, making it ideal for real-time applications requiring low latency, such as autonomous driving or video surveillance. However, it may compromise on precision for small or densely packed objects due to its grid-based approach, which can struggle with fine-grained localization in crowded scenes.

3.2 Faster R-CNN

Faster R-CNN is a two-stage object detection model optimized for high accuracy, in contrast to the single-stage YOLO described in the previous section. It first generates region proposals using a Region Proposal Network (RPN) and then refines and classifies these regions, excelling in complex scenes but with higher computational cost. Building on earlier R-CNN models, Faster R-CNN integrates the RPN into an end-to-end trainable network, improving efficiency.

3.2.1 Feature Extraction and Backbone

As with YOLO, Faster R-CNN employs a deep CNN **backbone** to extract features from the input image. Common backbones include:

- **Resnet (Residual Network).** It employs residual connections, where skip connections add the input of a layer to its output, mitigating vanishing gradients and enabling deeper architectures. Each layer applies 3x3 convolutions, batch normalization, and ReLU activations, producing feature maps with increasing channel counts.
- **VGG-16 (Visual Geometry Group of the University of Oxford - 16 layers).** A simpler architecture which consists of 16 layers with stacked 3x3 convolutions and max-pooling, outputting feature maps with 512 channels but higher computational cost due to its fully connected layers.

These backbones are pre-training on ImageNet, a dataset of over 14 million images across 1000 classes. This equips the backbones with robust feature representations for edges, textures and object categories, which are fine-tuned for object detection.

In conclusion, the backbone processes the input image to produce a feature map with reduced spatial resolution and a high number of channels. This feature map, encoding both low-level details and high-level semantics, is shared between the **RPN** and the subsequent **classification stage**, similar to how YOLO's backbone supports its neck and heads.

3.2.2 Region Proposal Network (RPN)

The RPN is a CNN that generates candidate regions of interest (RoIs), distinguishing Faster R-CNN's two-stage approach from YOLO's single-stage grid-based predictions. Operating on the backbone's feature map, the RPN proposes regions likely to contain objects, which are refined in the subsequent stage. Its key steps are:

- **Sliding window and anchor Boxes**: The RPN applies a small convolutional window (like 3x3) that slides over each position of the feature map, generating k anchor boxes per position. These anchors, similar to YOLO's but applied to feature map locations, have

predefined scales and aspect ratios, producing a large number of proposals. Each anchor is centered at a feature map pixel, mapping to a region in the input image based on the backbone's stride.

- **Predictions**: For each anchor, the RPN predicts:
  - **Objectness score**: A binary classification score (foreground vs background) indicating the likelihood of an object. This is analogous to YOLO's confidence score but focused on region proposals. It's trained with a classification loss, computed via a softmax function over two classes:
    $$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$
    where z_i is the score for class i.
  - **Box regression**: Four offsets $(\Delta x, \Delta y, \Delta w, \Delta h)$ to adjust the anchor's center and size, similar to YOLO's box coordinate adjustments. It's trained with a regression loss, a smooth L1 loss to align the anchor with potential objects, penalizing deviations from ground-truth boxes only for foreground anchors:
    $$\text{Smooth L1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$
    where x is the difference between predicted and ground-truth offsets.

  These predictions are made via two sibling convolutional layers: one outputs $2k$ objectness scores (2 per anchor), and another outputs $4k$ regression offsets (4 per anchor), applied across all feature map positions.
- **Proposal Selection**: The RPN adjusts anchors using regression offsets, ranks them by objectness score, and applies NMS (as described in the YOLO section) with an IoU threshold to eliminate overlapping proposals. A subset of high-scoring RoIs is selected for the next stage.

As a result of the RPN, regions with a high likelihood of containing objects are obtained, which are refined in the subsequent stage.

3.2.3 RoI Pooling and Classification

The second stage of Faster R-CNN refines and classifies the RoIs generated by the RPN, a step absent in YOLO's single-pass approach. This involves:

- **RoI Pooling or RoIAlign**: Each RoI corresponds to a region on the feature map. To handle varying RoI sizes, **RoI Pooling** divides the RoI into a fixed grid and max-pools features within each cell, producing a fixed-size feature map (like $7 \times 7 \times 1024$). **RoIAlign**, used in modern implementations, employs bilinear interpolation to avoid quantization errors, improving localization accuracy for small objects.

- **Classification and refinement**: The fixed-size feature map ($7 \times 7 \times 1024$) is processed by fully connected layers (in VGG-16 based models) or a convolutional head (in ResNet-based models). The network predicts:
  - **Class probabilities**: A softmax output over $C + 1$ classes. C refers to the number of object classes and 1 to the background.
  - **Box regression**: The network predicts four additional offsets $(\Delta x, \Delta y, \Delta w, \Delta h)$ to refine the RoI's coordinates, improving alignment with the true object boundaries. These offsets adjust the RoI's center (x, y) and size (w, h) relative to its current position, building on the RPN's initial adjustments. The regression is trained using the smooth L1 loss, as previously defined in the RPN.

This stage uses a multi-task loss, as the RPN, combining classification (softmax loss) and regression (smooth L1 loss), ensuring precise object localization and classification.

As a result of this stage, for each RoI, the output is a class label (highest-probability class), a confidence score (highest probability), and refined coordinates (x, y, w, h). These are then post-processed with **NMS** to produce final detections.

3.2.4 Non-Maximum Suppression (NMS)

As in YOLO, Faster R-CNN applies **NMS** to the final RoI predictions to remove redundant detections. Using the same process described in the YOLO section, NMS filters RoIs by confidence score, selects the highest-scoring box per class, and suppresses overlapping boxes, ensuring each object is represented by a single bounding box.

3.2.5 Multi-Scale Processing

Faster R-CNN handles objects of varying sizes differently from YOLO's explicit multi-scale feature fusion via FPN. It relies on:

- **Anchor boxes in RPN**: The RPN's anchor boxes cover multiple scales and aspect ratios, enabling proposals for small and large objects. This is analogous to YOLO's anchor boxes but applied to feature map positions rather than grid cells.
- **Backbone features**: The backbone's feature map, derived from deep layers, has a large receptive field suitable for large objects, while earlier layers (implicitly used via the backbone's hierarchy) provide finer details for smaller objects.
- **Optional image pyramid**: Some implementations process the input at multiple resolutions, but modern Faster R-CNN relies on anchor boxes and the backbone to avoid this overhead.

Later variants of Faster R-CNN incorporate a Feature Pyramid Network (FPN), as in YOLO, to enhance small-object detection by fusing multi-resolution feature maps. However, the base model's two-stage refinement ensures robust multi-scale performance.

3.2.6 Summary

Faster R-CNN is a two-stage object detection model that prioritizes accuracy over speed.

A CNN backbone extracts a feature map, which the RPN uses to generate region proposals via anchor boxes. These proposals are refined and classified using RoI Pooling or RoIAlign, followed by NMS to produce clean detections.

Multi-scale processing is achieved through the RPN's diverse anchor boxes and the backbone's hierarchical features.

Faster R-CNN excels in precision, particularly for small or densely packed objects, making it ideal for applications like medical imaging or detailed scene analysis. However, its two-stage design results in higher latency compared to YOLO, limiting its suitability for real-time applications.

4 Dataset

This chapter will describe the datasets used to train the models for each of the tasks.

4.1 Images

The images used for all the models are the same. It consists of a total of 638 images, taken to the drone and to the station from different angles, been full or empty, and in two different places with different background and lighting. See Figure 4 for some examples:

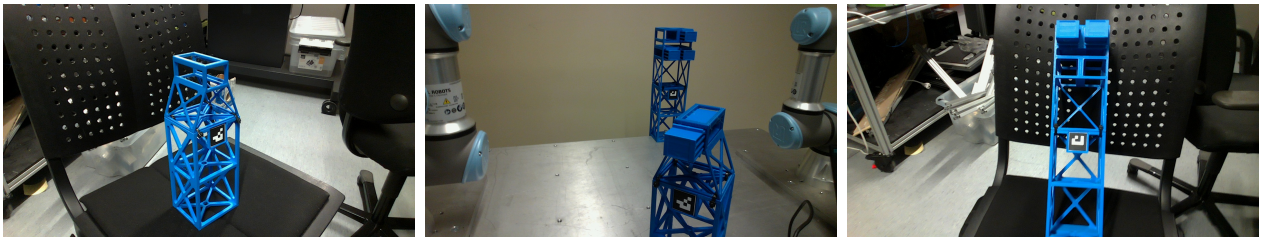


Figure 4: Examples of raw images used in the dataset.

The used resolution for the images has been 1280 × 720 pixels, exported as ".jpg" to handle better with memory.

In object detection tasks like those using YOLO and Faster R-CNN, ".jpg" is often recommended over ".png" due to its smaller file size, sufficient quality and compatibility with deep learning pipelines.

All the pictures have been taken with the L515 camera, so the model can then be used with the same camera in the robotic arm.

4.2 Annotations for task 1: drone or station

For each of the images in the dataset, the following annotations have been recorded:

- **Bounding box coordinates.** These define rectangular regions around objects in an image, represented as a tuple (x, y, w, h). Here, x and y specify the center point of the box relative to the image's top-left corner (in pixels or normalized between 0 and 1), while w and h denote the box's normalized width and height, respectively.
- **Class identifier for drone or station.** This defines if the object inside the bounding box is a drone (0) or a station (1).

As a result, each image (".jpg") has its corresponding annotation file (".txt"), which has the following format: "$class, x, y, w, h$". For each detected object in a single image, there will be one line in the corresponding annotation file, each corresponding to one object.

As doing this manually writing the values would be a very hard work, this job was done using https://www.makesense.ai/, an online tool where all the images can be uploaded, labeled using a visual GUI and exported as the wanted ".txt" files. However, it was still a long process, but shorter than it would be if doing it writing the values.

4.3 Annotations for task 2: hole classification

For each of the images in the dataset, the initial idea was to record the following annotations:

- **Bounding box coordinates.** (x, y, w, h), as explained previously.
- **Class identifier for hole status.** This defines if the hole inside the bounding box is (0) empty or (1) full.
- **Class identifier for hole identification.** This defines if the hole inside the bounding box is (0) the drone; (1) station top left; (2) station top right; (3) station bottom left; or (4) station bottom right.

This would result in annotation files (".txt"), with the format: "$class_{status}, class_{hole}, x, y, w, h$". However, YOLOv8 and Faster R-CNN are designed for one set of classes per model, as the annotations used in Task 1.

Using two classes would be possible with YOLOv8 and Faster R-CNN, but it would involve some changes, as the model would need two classification heads (status: 2 scores, ID: 5 scores), requiring changes to YOLOv8/Faster R-CNN's output layer (modifying head in YOLOv8 or box predictor in Faster R-CNN).

Coding custom heads, adjusting loss functions (separate cross-entropy for status/ID) and debugging increase development effort. In addition, dual-loss tuning requires more hyperparameter experiments, which extends training time.

This is why instead of following this initial annotation approach, both classes have been combined into one class. In this way, for each of the images in the dataset, the following annotations have been recorded:

- **Bounding box coordinates.** (x, y, w, h), as explained previously.
- **Class identifier for hole.** This defines if the hole inside the bounding box is (0) drone empty; (1) drone full; (2) station top left empty; (3) station top left full; (4) station top right empty; (5) station top right full; (6) station bottom left empty; (7) station bottom left full; (8) station bottom right empty; or(9) station bottom right.

As a result, each image (".jpg") has its corresponding annotation file (".txt"), with multiple lines for each object detected in a single image, as for Task 1. The annotation has also the same format as in Task 1 "$class, x, y, w, h$", but with different options for the class identifier.

4.4 Splits

The images and the corresponding annotations are finally divided randomly in three different splits:

- Train. Contains 80% of the images and annotations. This split is used to optimize the model's parameters by minimizing the loss function during training.
- Validate. Contains 10% of the images and annotations. This is used during training to monitor performance on unseen data, tune hyperparameters (like learning rate or anchor box sizes), and select the best model checkpoint, preventing overfitting to the training set.
- Test. Contains the remaining 10% of the images and annotations. This data is reserved for final evaluation after training, providing an unbiased measure of the model's generalization to new data,

5 Model Training

This chapter describes the trained models and experimental setup, providing context for the performance results presented in the subsequent chapter.

Five models have been trained for each task: YOLOv8n, YOLOv8s, YOLOv8m, YOLOv8l and Faster R-CNN.

The training of all the models was conducted on a NVIDIA RTX 4060 GPU and Intel Core Ultra 9 processor 285K CPU.

The YOLO models have been trained using the Ultralytics YOLO library. The models, pre-trained on COCO, are fine-tuned for 100 epochs with a batch size of 8, resizing images to 640x640 and leveraging a CUDA-enabled GPU for accelerated training. In addition, data augmentations ($\pm 50°$ rotation, 20% translation, 50% scaling and HSV adjustments) enhance robustness across diverse objects. Finally, checkpoints are saved every 10 epochs, enabling performance monitoring on validation splits.

In the other hand, Faster R-CNN models have been trained with a ResNet-50 FPN backbone using PyTorch and torchvision. Pre-trained on COCO, the model is fine-tuned for 100 epochs with a batch size of 2, using SGD (learning rate 0.005, momentum 0.9) on a CUDA-enabled GPU. Validation losses are monitored to save the best model, with no early stopping.

Table 1 shows the complexity of each model, including number of parameters and layers, and the training time for 100 epoch for each of them:

Metric	YOLOv8n	YOLOv8s	YOLOv8m	YOLOv8l	Faster R-CNN
Parameters (M)	3.2	11.2	27.2	45.9	41.2
Number of layers	195	225	245	295	340
Train time 1 (mm:ss)	02:25	03:07	05:25	08:27	38:32
Train time 2 (mm:ss)	02:22	03:26	05:25	07:59	40:45

Table 1: Comparison of model complexity for YOLOv8 variants and Faster R-CNN (ResNet-50 backbone), showing the number of parameters (in millions) and layers.

6 Model Results

This chapter will show the results for the five models in both tasks. These results are divided in training results, accuracy, robustness and real time detection, which are the following four subsections.

6.1 Training results

This subsection evaluates the performance of the models during training, focusing on validation metrics to assess learning progress and generalization.

Total training and validation losses.

Figures 5, 6 and 6 show the training and validation losses for all models in both tasks. YOLO and Faster R-CNN losses have been separated in different plots, as the loss scale was different, due to the neural networks been different.

For the YOLO models, it can be seen that the training losses remain similar in all variants, in both tasks. However, for the validation losses, all models converge in a similar way except of YOLOv8l in Task 1, which converges to a higher value than the rest.

For the Faster R-CNN models, Task 1 has lower validation losses than training losses, which is the different for Task 2, where they converge to a similar value. However, if looking at the value scale, it can be noticed that the scale for task 1 is much lower than the one for task 2, which makes it look like a bigger difference when it is not.

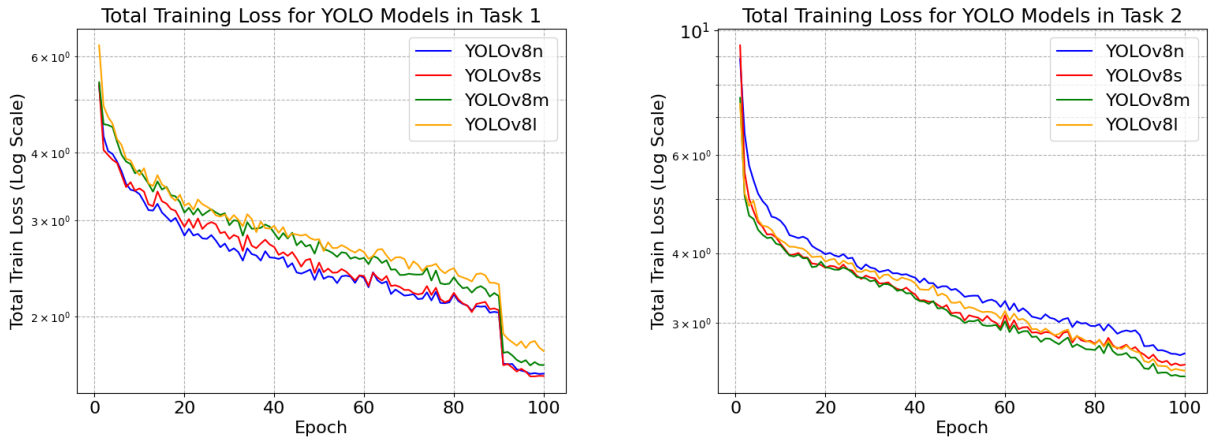


Figure 5: Total training loss for YOLO variants in Task 1 and Task 2.

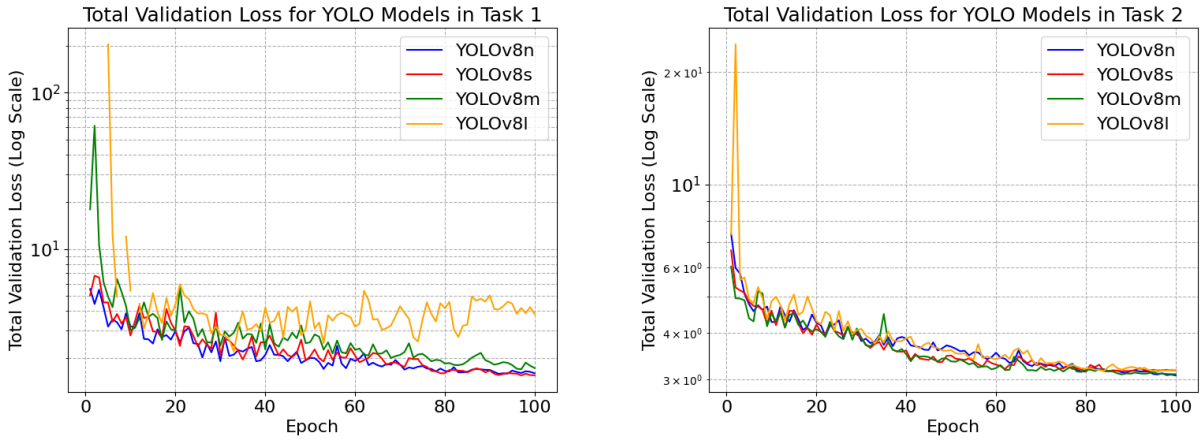


Figure 6: Total validation loss for YOLO variants in Task 1 and Task 2.

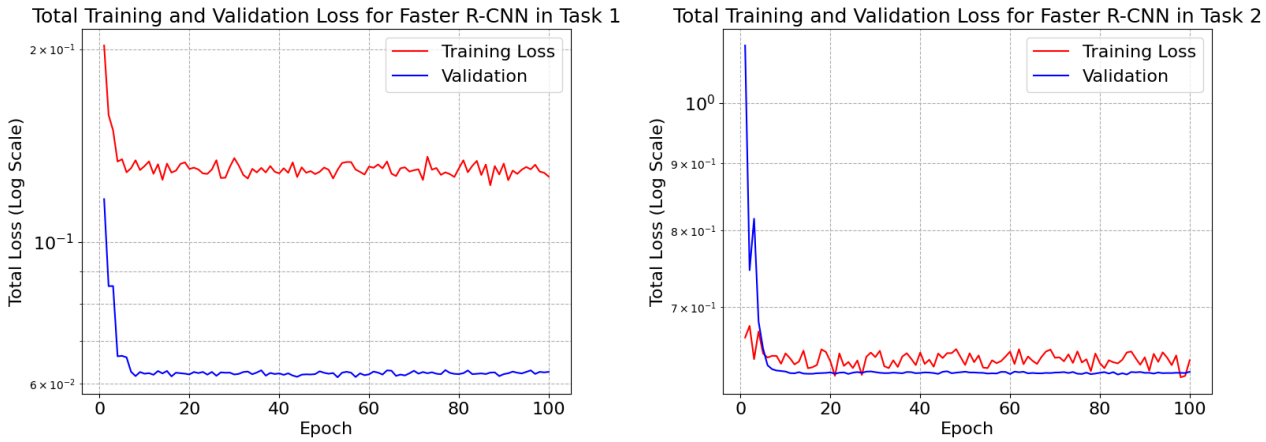


Figure 7: Total training and validation loss for Faster R-CNN in Task 1 and Task 2.

Precision and recall.

The precision is the relation between the true predictions in top of all the predictions made. It shows the precision of the trained model, along the epochs, over the validation dataset.

In the other hand, the recall is the relation between the true predictions in top of the total number of images in the validation dataset. It shows the percentage of correct predictions in the validation dataset, along the epochs.

Figures 8 and 9 show the precision and recall for all models in both tasks.

In task 1, all models converge to 1, yolov8l being the slower and more noisy. This can be related to YOLOv8l larger capacity (more parameters), overfitting to training data and causing noisy precision/recall on the small validation set.

In task 2, precision in all models converges slower than in task 1. Until epoch 40, it looks like its converging to 0.6, when suddenly it starts converging to 1. The recall, however, starts converging to 1, falling down at around epoch 40 and converging back to 1 in the end.

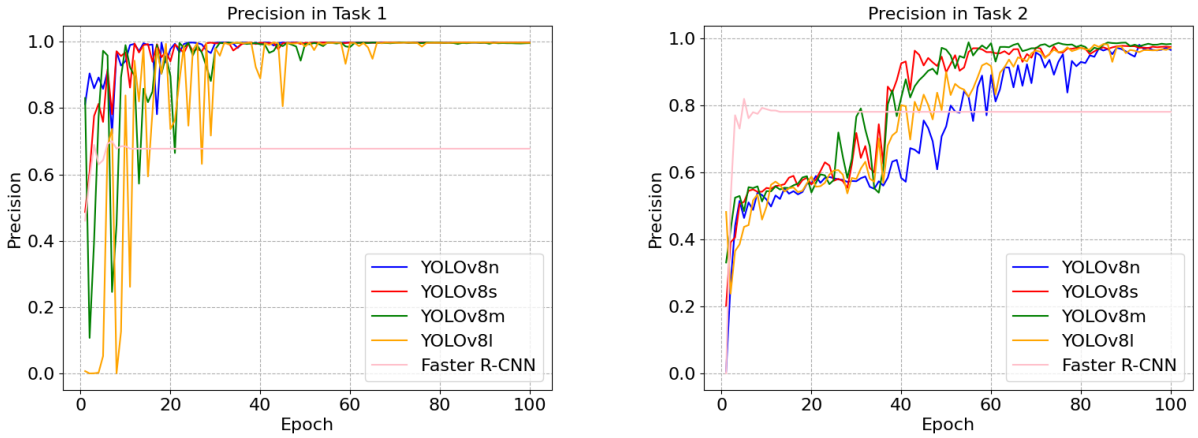


Figure 8: Precision for all the models for Task 1 and Task 2.

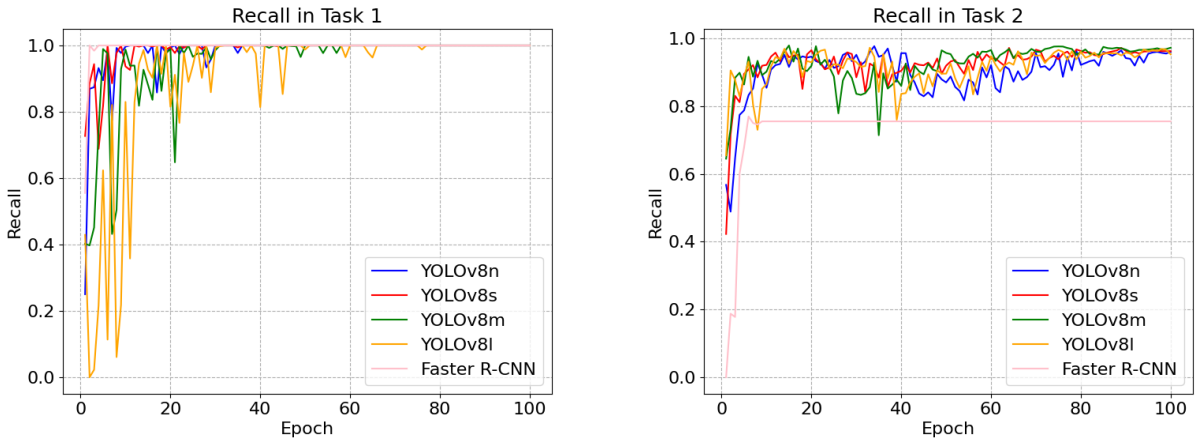


Figure 9: Recall for all the models for Task 1 and Task 2.

mAP@0.5:0.95.

The mean average precision (mAP) calculates the average precision (AP) for each class, and then averages across all classes. AP is the area under the precision-recall curve, balancing precision (correct detections/total detections) and recall (correct detections/total ground truths) at a given IoU threshold: 0.5 to 0.95.

As it can be observed in Figure 10, all models in Task 1 converge to around 0.8, except of YOLOv8l, which is also more noisy. This can be related to YOLOv8l larger capacity (more parameters), overfitting to training data.

In task 2, there is no such difference among YOLO models, and all models converge to around 0.65. This can be related to the higher complexity of the task, which deletes the capacity differences that was affecting YOLOv8l in Task 1 and also reduces the average precision for all models.

In both tasks, Faster R-CNN converges faster than YOLO models to the same values, probably related to the models architectures.

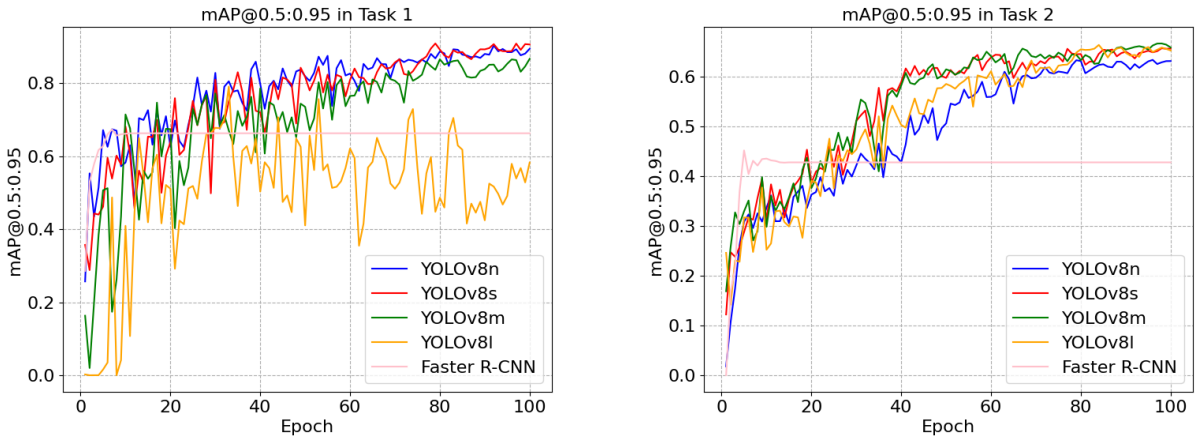


Figure 10: mAP@0.5:0.95 for all the models for Task 1 and 2.

6.2 Accuracy

This subsection analyzes the detection accuracy of the five models on the test sets, quantifying their ability to correctly detect and classify objects. For this, the confusion matrices at confidence = 0.7 and Uio = 0.5 are shown in Figures 11 and 12.

In task 1, YOLO models perfom very well, while Faster R-CNN predict a lot of false positives (predictions that are not true). YOLOv8l is the more accurate among the YOLO models.

In task 2, Faster R-CNN fails to predict objects, while YOLO models look to perform better. It's hard to visualize which of the YOLO models performs the best, but it looks like YOLOv8s predict the most in the main diagonal.

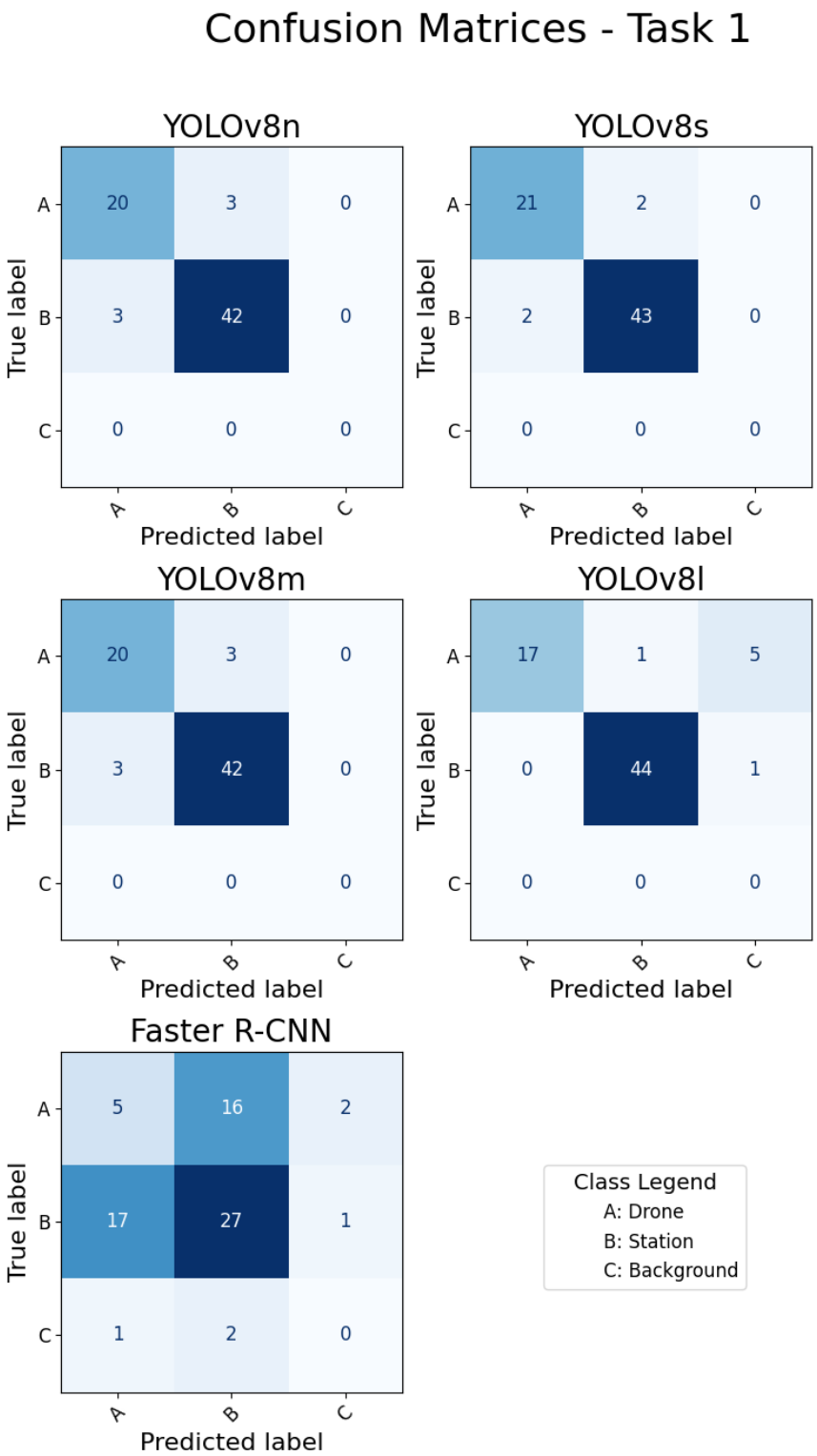


Figure 11: Confusion matrices on Task 1.

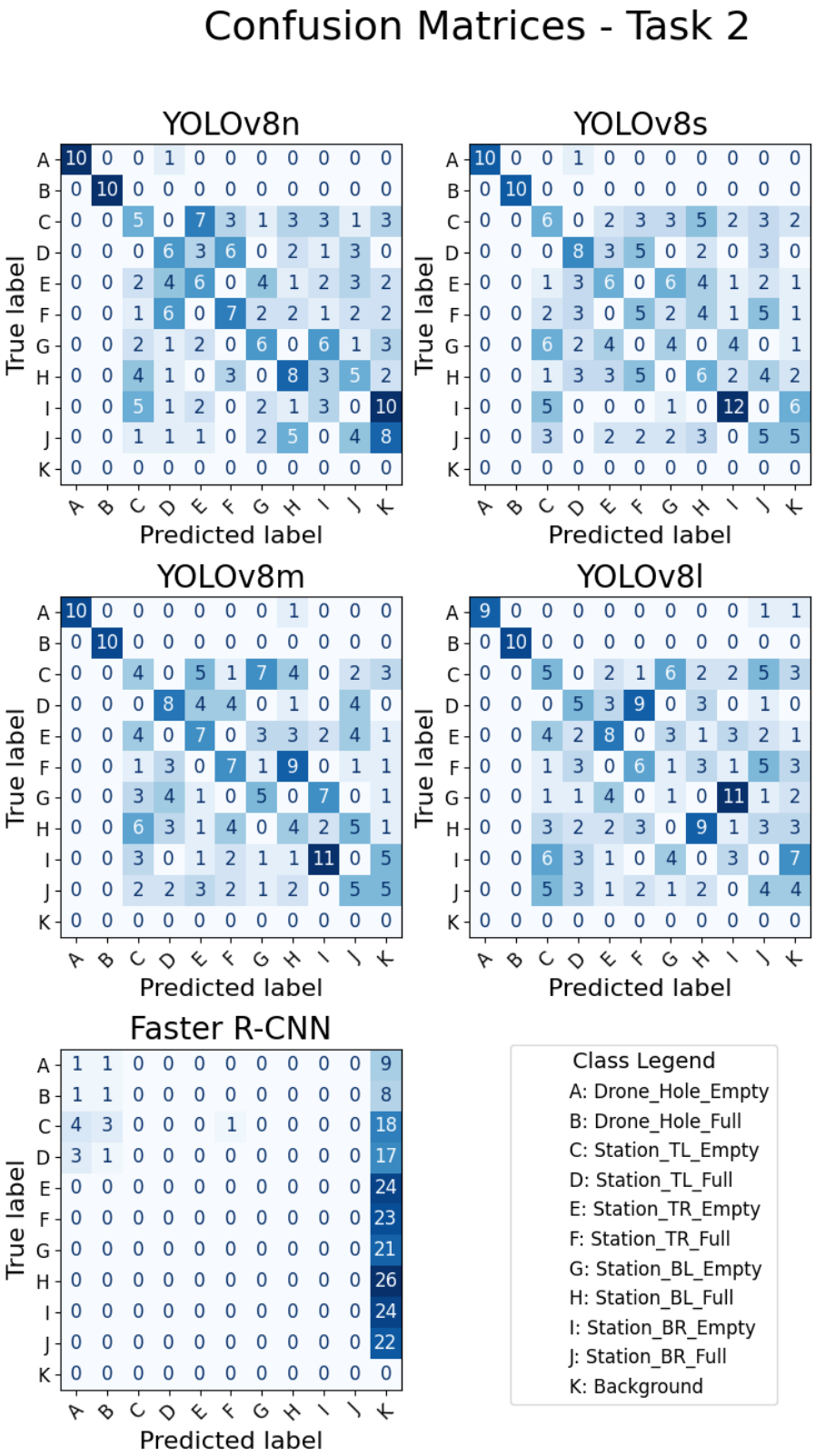


Figure 12: Confusion matrices on Task 2.

6.3 Robustness

This subsection assesses the robustness of the models under varied conditions, testing their generalization to challenging scenarios. Four test images, created from the test sets, represent distinct conditions: low brightness, far distance, upside down orientation and crowded environment. Qualitative detection outputs are analyzed for each condition, comparing YOLOv8's augmentation-driven robustness (like $\pm 50°$ rotation) against Faster R-CNN's precision-focused design.

For task 1, as it can be seen in Figure 13, the next observations are made from each model:

- Yolo Nano: success in normal, dark and rotated conditions, but sometimes predicts false positives. Doesn't predict nothing for the far test. In the crowded test it predicts lots of false positives.
- Yolo Small: Very similar to Yolo Nano, but it predicts the station from distance.
- Yolo Medium: Similar results as the previous, detecting more objects from distance, been these false positives. Better results in normal conditions, without false positives.
- Yolo Large: Fails to predict in normal and rotated conditions. Success in dark environment. Doesn't provide predictions from far distance. In the crowded environment it reduces the number of predictions, detecting the drone but not the station.
- Faster R-CNN:Success in normal, dark and rotated tests. Doesn't predict nothing from far distance. In the crowded environment it predicts ther drone, not the stations, but at least doesn't provide false positives.

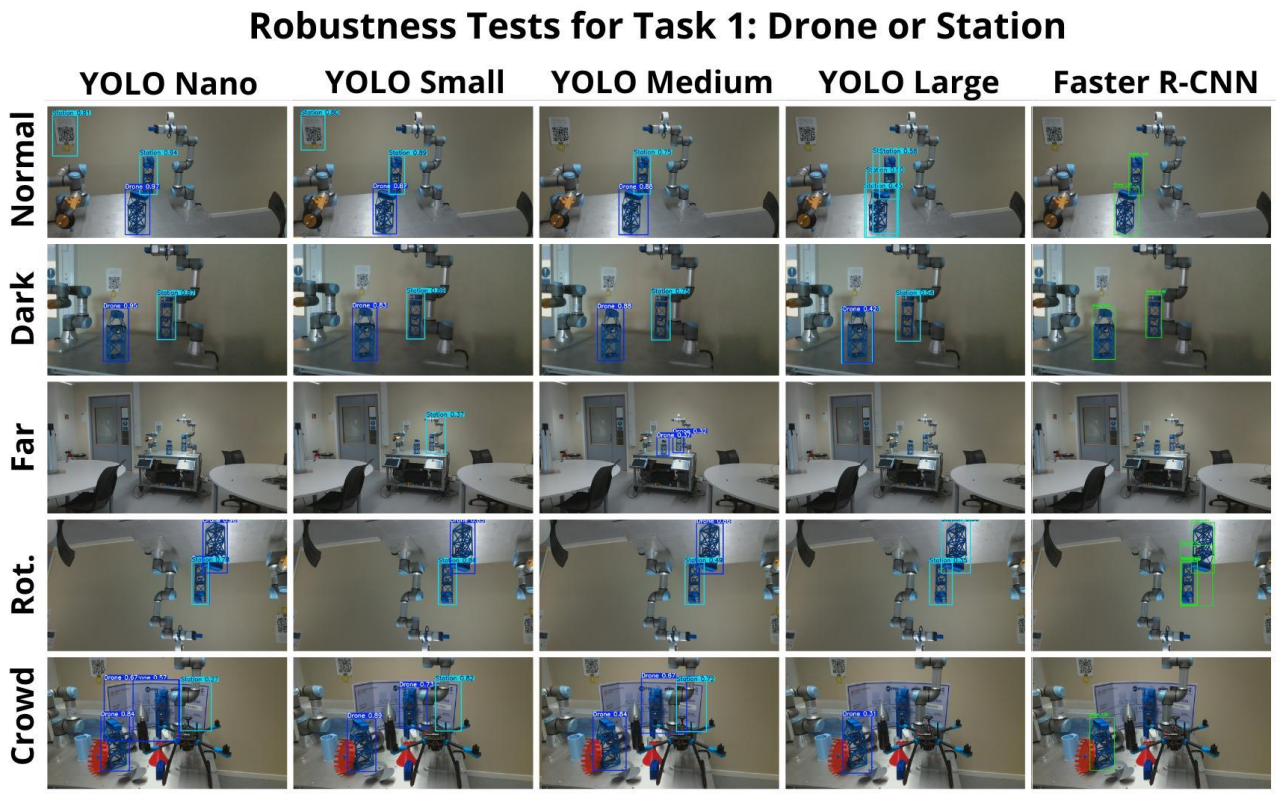


Figure 13: Robustness tests for task 1.

For task 2, as it can be seen in Figure 14, the next observations are made from each model:

- Yolo Nano: In general, it detects the drone easier than the station. It fails from far distance.
- Yolo Small: It provides better predictions in dark conditions. Form far distance, it doesn't predict nothing. It fails in the rotated test. In the crowded environment it predicts some false negatives.
- Yolo Medium: Similar results, but with worse results in dark environment and rotated. Successfully predicts the drone in crowded environment, but misses the station.
- Yolo Large: Very similar results to the Yolo Medium, but predicts better for station in crowded environment, even if some are false positives.
- Faster R-CNN: Bad results for all tests.

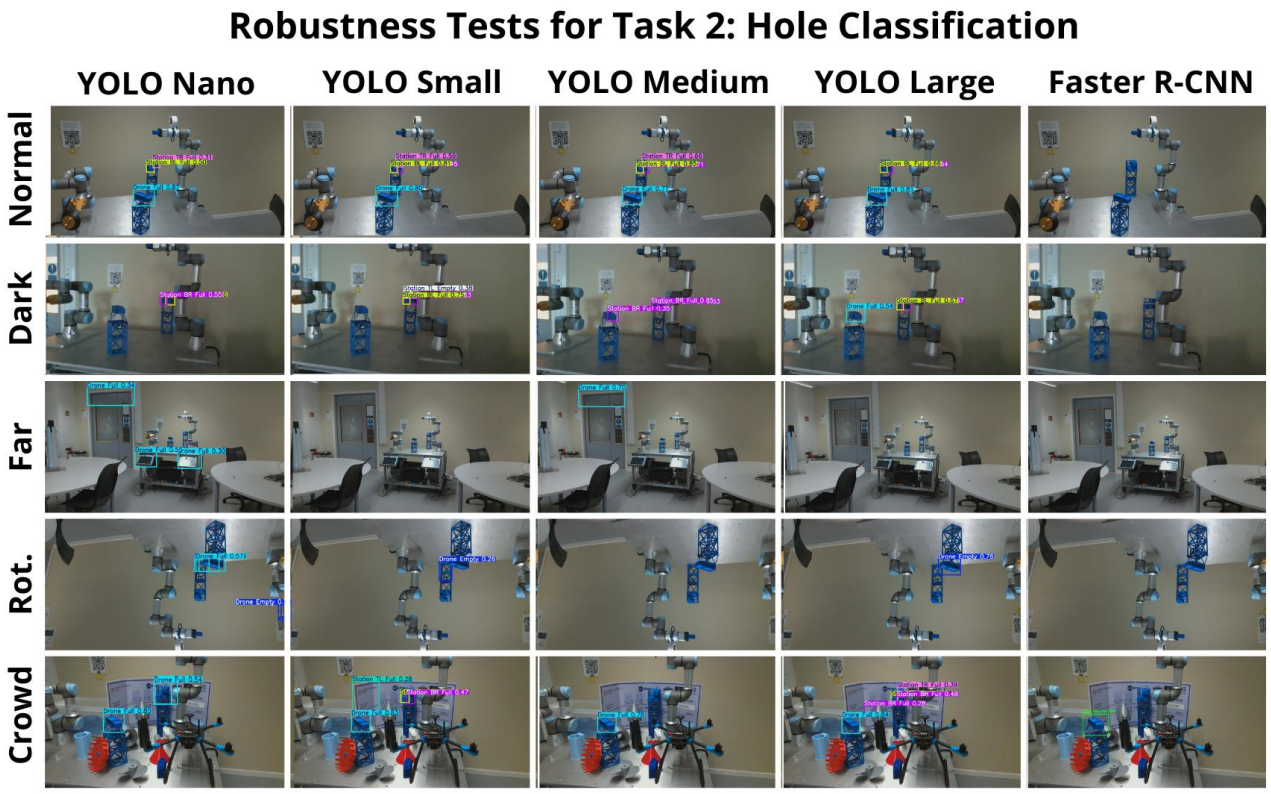


Figure 14: Robustness tests for task 2.

6.4 Real time detection

This subsection evaluates the real-time inference capabilities of the models.

First, hardware-agnostic metrics will be measured, these been the memory size of the model (MB) and the number of Multiply-Accumulate Operations (MACs). This last refers to the number of fundamental operations required to process an input through the network (forward pass for prediction), where a fundamental operation is when a value is multiplied by a weight and the result is added to an accumulator.

Then, hardware-dependent metrics will be measured on the personal laptop used for the robotic arm project, an ASUS Zenbook. These metrics are the inference time (ms), frames per second (FPS), CPU utilization (%), RAM (GB) and power consumption (W), which are measured using the L515 in real time. Other metrics like GPU utilization (%) and maximum VRAM (GB) are not included, as the laptop runs on CPU.

Real time metrics for Task 1 and 2 can be observed in Tables 2 and 3, respectively. In general, it can be seen how the model size affects he inference time and the fps, slowing down the fps as the model gets bigger. The only one that can be used smoothly with Asus Zenbook is Yolov8n, as the rest slow down the fps too much.

Metric	YOLOv8n	YOLOv8s	YOLOv8m	YOLOv8l	Faster R-CNN
Model size (MB)	6.2	22.5	52	87.6	165.7
MACs (GMac)	9.4	32.89	90.83	190.06	208.03
Inference time (ms)	106.06	213.34	467.27	696.35	2028.06
FPS	13.54	6.03	2.47	1.59	0.49
CPU util. (%)	19.88	23.68	26.77	28.62	43.56
RAM (GB)	13.63	13.76	13.66	13.82	14.32
Power (W)	7.68	8.55	9.02	9.29	11.53

Table 2: Real-time detection metrics for Task 1 on ASUS Zenbook, grouped by hardware-agnostic (model size, MACs) and hardware-dependent (inference time, FPS, CPU utilization, RAM and power) categories.

Metric	YOLOv8n	YOLOv8s	YOLOv8m	YOLOv8l	Faster R-CNN
Model size (MB)	6.2	22.5	52	87.6	165.9
MACs (GMac)	9.41	32.91	90.86	190.1	208.07
Inference time (ms)	108.10	187.10	427.81	692.59	3073.11
FPS	12.92	7.91	2.65	1.57	0.42
CPU util. (%)	17.89	26.91	26.47	30.24	47.14
RAM (GB)	13.91	14.05	14.06	13.84	14.48
Power (W)	7.68	9.04	8.97	9.54	12.07

Table 3: Real-time detection metrics for Task 2 on ASUS Zenbook, grouped by hardware-agnostic (model size, MACs) and hardware-dependent (inference time, FPS, CPU utilization, RAM and power) categories.

Finally, a video is created showing each of models for each of the task in real time. The video is attached with the report. In this video, it can be appreciated the slowness with bigger models.

7 Feature map analysis for best model

Finally, one of the best performing models in Task 2 will be analyzed by trying to understand its feature maps.

For this, a video has been created, where the feature maps from the channels in each of the layers are shown, from input to the prediction in sequence. As some channels have up to 512 channels, the figures shown in the video are limited to show 36 channels per layer. The channel amount and the resolution is shown in the title.

The chosen model is YOLOv8s (small), as its predictions in the confusion matrix were the most accurate.

In the video, it can be noticed that the first layers have higher resolution and less channels. While going deeper into the layers, the amount of channel increases, while the resolution decreases. Finally, as the layers approach the end, the channel amount decreases again, resulting in the final prediction.

In these last layer, it can be appreciated that the general shape of the station is conserved, and in some feature maps the batteries are highlighted from the rest of the picture, been brighter or darker.