Multimodal Recognition Project

Session 1: Object Detection

Group 6

Abril Piñol
Biel González
Mireia Majó
Alejandro Donaire

***Module**: C5 - Visual Recognition*
***Coordinators**: Ernest Valveny, Carlos Boned, Lei Kang*

Contents

1 Running inference with pre-trained models (task c)

Description of KITTI-MOTS

The KITTI MOTS dataset consists of **21 traffic sequences**, each containing between ~100 to over 800 images, for **detection** of two different classes.

For our split:

- 75% of the sequences (0–15) were used for training.
- The remaining 25% (16–20) were reserved for testing.

Type of scenes:

- **Inner city streets**: scenes with many pedestrians and some vehicles, typically in urban environments.
- **High speed roads**: scenes with dense traffic and with faster moving vehicles and few or no pedestrians.

Inner-city streets example:

High-speed roads example:

Object classes:

Class 1: Cars *(moving vehicles, except motorbikes and bicycles)*

Class 2: Pedestrians *(walking, standing or slightly moving)*

Class 10: Ignore Region

We didn't use this class

1 Running inference with pre-trained models (task c)

Faster R-CNN from Detectron2

For this task, we utilize the Detectron2 framework by Facebook AI, configuring a pre-trained Faster R-CNN model for object detection. We load the model using a custom configuration file and process the test data using Detectron2's built-in data handling utilities. The model is then applied to each dataset image, filtering and mapping COCO-class detections to KITTI-MOTS categories.

Qualitative results

The results at first sight seem promising, we could only extract the predicted bounding boxes of the model, but not combine them with the ground truth bounding boxes. The errors we could find from this results were the next ones:

False Positives:

- Shadows and reflections
- Overdetection of objects (lots of false positives)

Detection Issues:

- Sometimes associates non-squared colour blobs to persons

1 Running inference with pre-trained models (task c)

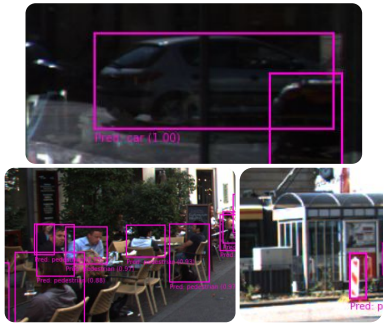
DETR from Hugging Face

For this task, first we load the pre-trained DETR model "**facebook/detr-resnet-50**" from Hugging Face, along with the corresponding image processor. Then, we preprocess the test data with a custom PyTorch **DataLoader**, which uses a tailored **collate** function and the image processor. We then loop over the dataset, **filtering** and **matching** COCO to KITTI-MOTS classes.

Qualitative results

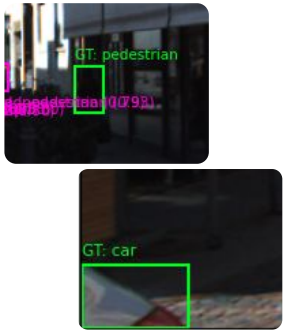
The results were generally very good, raising the confidence threshold helped get cleaner and more precise predictions. However, despite the strong performance, the model was not flawless. Some of the cases in which the model failed where the following:

False Positives:



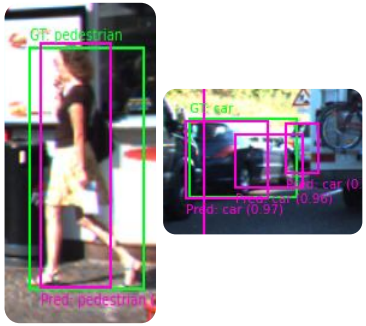
- Reflections as real objects
- Over Detecting pedestrians (small objects , cyclists, sitting people,...)

False Negatives:



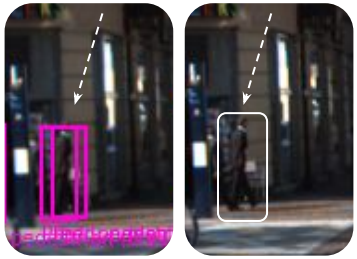
- Objects in low light
- Objects with only a very small part visible

Detection Issues:



- Bbox misalignment with gt
- Over Detection due to sudden illumination changes

GT gaps:



- Correct detections but missing from the gt annotations

1 Running inference with pre-trained models (task c)

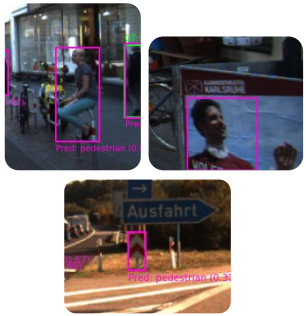
YOLOv11 from Ultralytics

For this task, we utilize a pre-trained YOLO model (yolo11x.pt) to detect objects in KITTI-MOTS test sequences. The model runs inference on test images, filtering detections to match KITTI-MOTS categories (cars and pedestrians). Ground truth annotations are parsed from KITTI-MOTS instance segmentation files, and predictions are evaluated using the COCO evaluation framework.

Qualitative results

The results were generally very good, raising the confidence threshold helped get cleaner and more precise predictions. However, despite the strong performance, the model was not flawless. Some cases in which the model failed were the following:

False Positives:

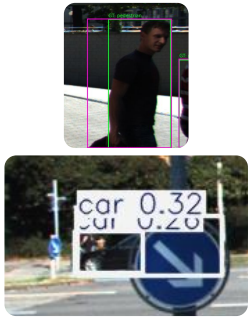


- Over Detecting pedestrians (small objects, cyclists, sitting people,...)
- Wrong predictions when similar proportions.

False Negatives:

- No detection due to lighting conditions.

Detection Issues:



- S Bbox misalignment with gt
- Over Detection due to occlusion

GT gaps:

- Correct detections but missing from the gt annotations

The official COCO metrics

The metrics used to evaluate the different object detection approaches are the official COCO Metrics. These are conformed by a combination of Average Precisions and Average Recalls. To calculate them, we used the **COCOeval** function from the **pycocotools** library, adapting the datasets and outputs to the required format for it to work properly. The metrics are the ones seen below. However, we will only show AP as it is the most commonly shown (AR can be seen on the GitHub of the project).

Average Precision

AP: AP at IoU=.50:.05:.95

$\mathbf{AP}_{50}$: AP at IoU=.50

$\mathbf{AP}_{75}$: AP at IoU=.75

$\mathbf{AP}_{S}$: AR for small objects (area<32^2)

$\mathbf{AP}_{M}$: AR for medium objects (32^2<area<96^2)

$\mathbf{AP}_{L}$: AR for large objects (area>96^2)

Average Recall

$\mathbf{AR}_{1}$: AR for 1 detection per image

$\mathbf{AR}_{10}$: AR for 10 detections per image

$\mathbf{AR}_{100}$: AR for 100 detections per image

$\mathbf{AR}_{S}$: AR for small objects (area<32^2)

$\mathbf{AR}_{M}$: AR for medium objects (32^2<area<96^2)

$\mathbf{AR}_{L}$: AR for large objects (area>96^2)

Faster R-CNN from Detectron2

To assess the performance of the pre-trained Faster R-CNN (F R-CNN) model, we follow a structured approach to collecting and formatting evaluation data. During the evaluation loop, we gather both the ground truth annotations and the model's predictions. It is essential to ensure that these elements are structured correctly so that they can be properly analyzed using our evaluation tools.

A crucial step in this process involves mapping category labels to maintain consistency across datasets. Specifically, we convert all class labels to match the KITTI-MOTS dataset, which is the dataset we are working with. Additionally, to ensure compatibility with the evaluation library we are using, we transform all bbox annotations into the COCO format. The COCO format is a widely used annotation standard that facilitates object detection benchmarking.

Despite following these steps, we encountered multiple issues throughout the evaluation process. These challenges ranged from inconsistencies in data formatting to potential mismatches in the expected input-output structure for the evaluation library. As a result, we faced difficulties in obtaining reliable performance metrics for the pre-trained F R-CNN model. **After extensive debugging and troubleshooting, we were ultimately unable to extract and validate the evaluation metrics successfully.**

DETR from Hugging Face

To evaluate the pre-trained DETR model, we gather the ground truth and predictions during the evaluation loop in the correct format. We **map categories**, so all labels are **KITTI-MOTS**, and **convert all bounding boxes** to the **COCO** format, compatible with the evaluation library we use.

The next table shows the Average Precision metrics:

AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
49.5	77.8	53.7	18.0	51.1	75.2

We can make some observations from this table:

The AP is **good but moderate**, with room for improvement. Thus, we should expect missing detections or over-detections.

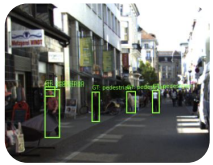
The model performs **substantially better on larger objects** than smaller ones (75.2 vs 18.0). Thus, we should expect missing detections at small scales.

The model is **good at roughly locating objects** (AP_{50}=77.8), but the performance notably drops when the location precision is higher. Thus, we should expect misaligned bounding boxes.

EXAMPLES OF EXPECTED ERRORS

Misaligned bounding box

Many over-detections



Missing detections at small scales

YOLO from Ultralytics

To evaluate the pre-trained YOLO model, we gather the groundtruth and predictions during the evaluation loop in the correct format. We **map categories** so all labels are **KITTI-MOTS**, and **convert all bounding boxes** to the **COCO** format, compatible with the evaluation library we use.

The next table shows the Average Precision metrics:

AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
62.0	80.9	68.6	30.8	64.8	83.9

We can make some observations from this table:

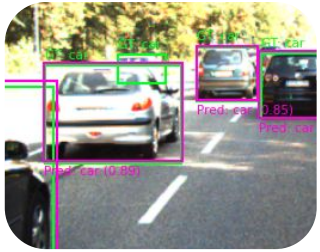
We get considerably good AP, the majority of them surpass by a great margin the 50%

We see that the model performs better with larger objects than smaller, however small objects are detected also but in a lesser proportion

YOLO locates with quite a precision the bounding boxes of the classes as we can see it gets near a 70% of AP_{75} and the big majority of the predictet bounding boxes are at least 50% coincident.

EXAMPLES OF EXPECTED ERRORS

Overdetections



Missed objects

3 Fine-tuning the models on KITTI-MOTS (task e)

DETR from Hugging Face

To fine-tune DETR to KITTI-MOTS, first we initialize the "**facebook/detr-resnet-50**" model from Hugging Face with a custom, 2-label map: "**{0: "car", 1: "pedestrian"}**", changing the classification head. Then, we create a **custom DataLoader** to preprocess the data to match the label map and apply augmentations.

We train the model for 30 epochs with early stopping and with the Trainer and TrainingArguments functions from Hugging Face. We performed **5 different fine-tuning executions**:

- **Baseline execution** with some default arguments (see our GitHub for details).
- With a set of **Albumentations** (see explanation below).
- Changing the **learning rate** (from 1e5 to 3e5), no Albumentations.
- Changing the **image size** (from 480x600 to 600x800), no Albumentations.
- Changing the **image size** (from 480x600 to full size of 800x1333) + **Albumentations**.

The Albumentations we use consist of **flips, slight shifts in scale and rotation, slight brightness, contrast and color changes, and some blur or noise**. Next is an example where we apply them multiple times:

Notice that with Albumentations the bounding boxes (in red) are correctly adjusted to the transformations.

3 Fine-tuning the models on KITTI-MOTS (task e)

QUANTITATIVE RESULTS

In the table below, we show the official COCO metrics for the different executions, together with the only-inference execution, for comparison.

Execution	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Non-FT	49.5	77.8	53.7	18.0	51.1	75.2
FT Base	33.3	61.0	31.7	8.07	31.4	65.7
FT + Aug	37.9	65.2	37.4	9.46	36.5	70.3
FT ↑LR	30.5	63.8	24.2	5.57	28.2	65.2
FT ↑Size	43.7	71.5	44.9	12.8	45.1	73.9
FT ↑↑S. + A.	**52.4**	**78.3**	**57.4**	**21.6**	**55.5**	**76.2**

As expected, the size of the image improves performance, although slows training. The albumentations improve the results, while increasing the learning rate worsens them. Fine-tuned models are typically worse, but with full resolution and albumentations they outperform only-inference.

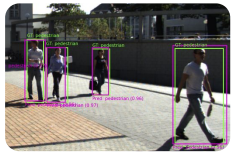
Also, in general, smaller objects are the hardest to predict well for the models, while larger ones are significantly easier.

OTHER COMMENTS

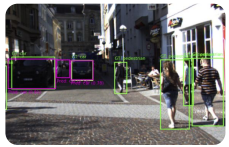
- We experienced **problems with LR=5e5**. We think it might be too high, making training unstable and generating degenerate bounding boxes.

QUALITATIVE RESULTS

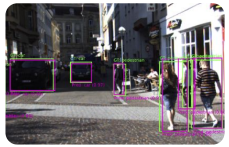
They are generally good. To the left is a good example from the best model (FT ↑↑S. + A.).



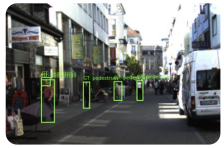
Cars are easier to learn. Predictions for the class pedestrian don't appear until later epochs. We think it's due to some class imbalance.



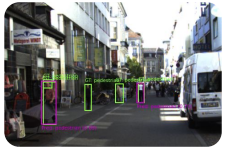
← Epoch **1**
Epoch **13** →

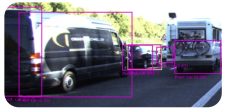


Small objects are hard to detect for models trained with small images.



← **Low** res
High res →





Still predicting **reflections**, **cyclists** (as pedestrians) or **vans** (as cars)

3 Fine-tuning the models on KITTI-MOTS (task e)

YOLOv11 from Ultralytics

QUANTITATIVE RESULTS

In the table below, we show the official COCO metrics for the different executions, together with the only-inference execution, for comparison.

Execution	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Non-FT	**62.0**	**80.9**	**68.6**	**30.8**	**64.8**	**83.9**
FT Base	55.1	72.6	60.7	20.2	59.1	81.3
FT + Aug_1	44.8	61.1	50.0	18.8	46.8	67.2
FT+Aug_2	52.2	71.6	59.0	15.8	57.3	79.6

We tried two different augmentations and both decreased the metrics when comparing to the base YOLO without any kind of fine-tuning. The $Augment_1$ uses the “randaugment” and the $Augmentation_2$ uses the “autoaugment” with other adjusted parameters like hsv, scale or translation.

4 Fine-tuning on Chest X-Ray dataset (domain shift) (task f)

Chosen Dataset: Chest X-ray N Object Detection (from Hugging Face)

- Dataset of Chest X-rays about different conditions
- Contains 8 different classes to detect
- Has only a Bbox per image
- Has only a train split with 880 images
- We divided it in train, validation, and test (75-15-15)

Link to the dataset: [Tsomaros/Chest_Xray_N_Object_Detection · Datasets at Hugging Face]

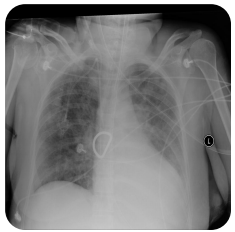
4 Fine-tuning on Chest X-Ray dataset (domain shift) (task f)

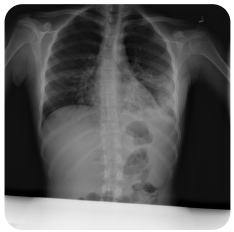
Faster R-CNN from Detectron2

We found out that probably the dataset was too small for the domain shift, as it only contained 880 images for 8 classes and the task was very specific.

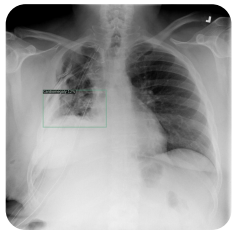
This caused the model to predict the wrong classes, also to make predictions with small confidenced on the images and on the boundung boxes sometimes they aren't predicted.

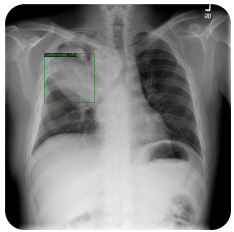
We think that probably with more images this task could have yield better results.





We also found that there are a lot of images with no predictions at all.





Generally very low confidence scores in the predictions.

5 Analysing the differences between the models (task g)

QUANTITATIVE RESULTS COMPARISON

To compare the different approaches, we gather a series of quantitative metrics that we obtained throughout our experiments: training time, inference time, number of parameters and average precision. We summarize them in the following tables:

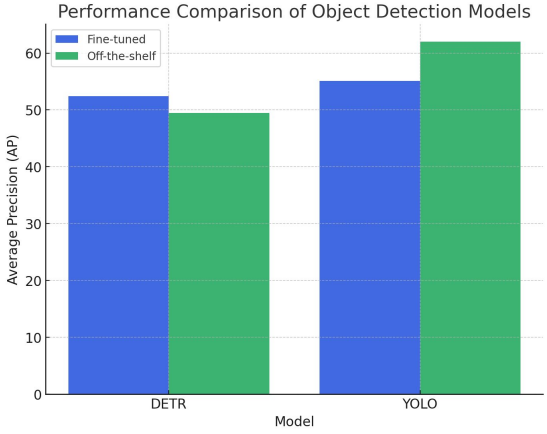
Fine-tuned (best run)

Method	AP	Training time*	Inference time**	Num. Params.
Faster R-CNN	-	-	-	-
DETR	52.4	~3h	~22 img/s	41.5M
YOLO	55.1	-	-	-

Non fine-tuned (inference only)

Method	AP	Inference time**	Num. Params.
Faster R-CNN	-	-	-
DETR	49.5	~20 img/s	41.5M
YOLO	62.0	-	-

- YOLO performs best off-the-shelf (62.0 AP), while DETR benefits more from fine-tuning (52.4 AP). This suggests that YOLO is already well-optimized for general object detection tasks without additional training, while DETR benefits from fine-tuning.
- Faster R-CNN results are missing, due to evaluation issues mentioned before. YOLO off-the-shelf shows the strongest performance, surprisingly, better than when it's fine-tuned.



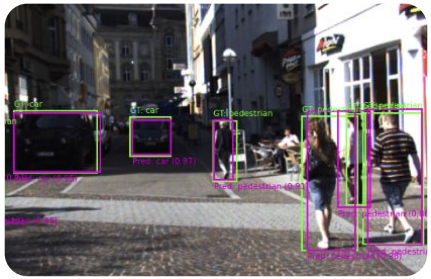
*approximate time to convergence
**tested in the full test set, dividing total samples over total inference time. Approximate value.

5 Analysing the differences between the models (task g)

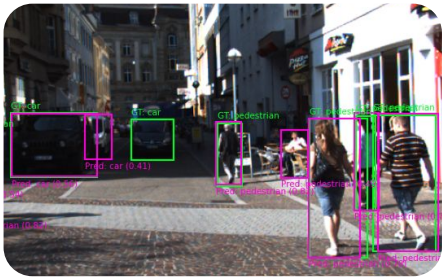
DIFFICULTY OF IMPLEMENTATION

Notably, implementing DETR with Hugging Face has been the hardest. Despite semi-automatic image processors and pre-trained models being readily available, most of the pre-processing has to be done by hand and it lacked important automations such as the ability to quickly include Albumentations. Similarly, implementing Faster R-CNN also presented difficulties, as its repository consists of many files that need to be modified in a non-intuitive way for custom training and evaluation. In contrast, Ultralytics offers a user-friendly framework, making YOLO significantly easier to implement and work with.

QUALITATIVE RESULTS COMPARISON



DETR



YOLO

While DETR is good at roughly localizing objects, it is still imprecise many times, and significantly over-detects objects, sometimes producing double or triple same-detections and false positives. Regarding YOLO, we can see that the improvement of metrics has a real impact in the bounding boxes, however in this image we can also see that the model is not perfect and produces errors different from the DETR, like the underected car at the center.

Summary

Results, Challenges and Conclusions

MAIN QUANTITATIVE METRICS

	Approach	AP
Fine-tuning	Faster R-CNN	-
	DETR	52.4
	YOLO	55.1
Inference	Faster R-CNN	-
	DETR	49.5
	YOLO	**62.0**

BEST APPROACH

YOLO (inference) with Augmentations (AP **62.0**)

PERFORMANCE BOOSTERS

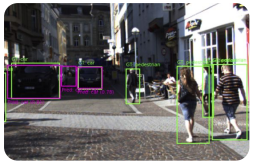
Extensive augmentations

Using full size Images for training

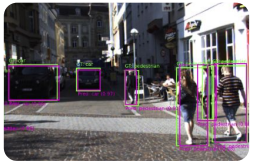
OTHER COMMENTS

DETR from Hugging Face was the hardest approach to implement

Pedestrians were notably harder to learn than cars.

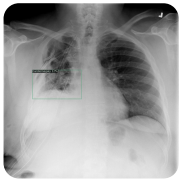


Epoch **1**



Epoch **13**

DOMAIN SHIFT OBSERVATIONS



The model struggled to predict the classes and location on the images, likely due to not enough data and the dataset being too different.