Master in
Computer Vision
Barcelona

Project
Module 6
Coordination

Video Surveillance for Road Traffic Monitoring

J. Ruiz-Hidalgo / X. Giró

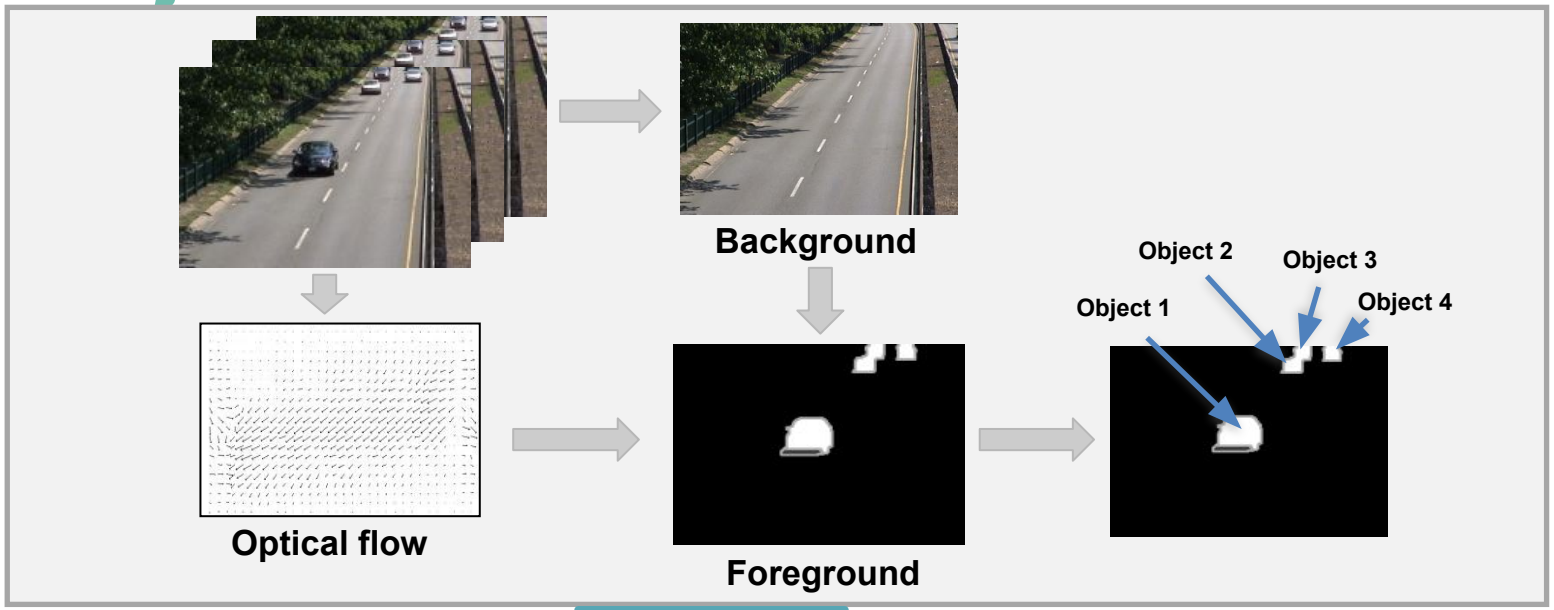
j.ruiz@upc.edu / xavier.giro@upc.edu

Week 3: Report

Teams & Repos (please state any change)

Repo	Students
Team 1	Rachid Boukir, Josep Bravo, Alex Martín, Guillem Martinez, Miquel Romero
Team 2	Álvaro Budria, Alex Carrillo, Sergi Masip, Adrià Molina
Team 3	Albert Barreiro, Manel Guzmán, Jiaqiang Ye Zhu and Advait Dixit
Team 4	Julia Ariadna Blanco Arnaus, Marcos Muñoz González, Abel García Romera and Hicham El Muhandiz Aarab
Team 5	Razvan-Florin Apatean, Michell Vargas, Kyryl Dubovetskyi, Ayan Banerjee and Iñigo Auzmendi
Team 6	Guillem Capellera, Ana Harris, Johnny Núñez, Anna Oliveras

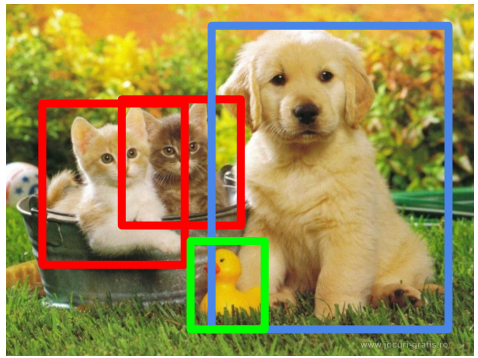
Project Schedule



Week 1	Week 2	Week 3	Week 4	Week 5	Week 6
• Introduction • DB • Evaluation metrics	• Background estimation • Stauffer & Grimson	• Object Detection • Tracking	• Optical flow • Tracking	• Multiple cameras • Speed	• Presentation workshop

Tasks

- Task 1: Object detection
  - **Task 1.1: Off-the-shelf**
  - Task 1.2: Annotation
  - Task 1.3: Fine-tune to your data
  - Task 1.4: K-Fold Cross-validation
- Task 2: Object tracking

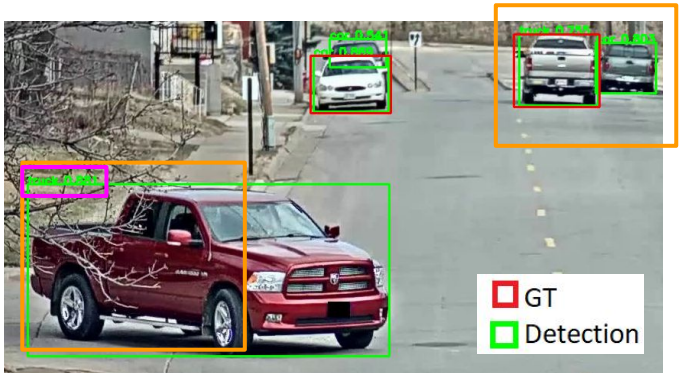


Task 1.1: Object Detection: Off-the-Shelf (Team X)

Copy & paste (max 2 pages)

Task 1.1: Object Detection: Off-the-Shelf (Team 1)

- **Only cars and trucks** are considered for the object detection (COCO classes).
  - Trucks are also considered, as pick-up cars are considered trucks for COCO trained DNNs (as seen in the image).
- When **loading GT**:
  - **Only class car** is considered.
  - **Parked and non-parked cars** are considered as the object detectors will detect both.
  - **"Occluded" and non-occluded cars** are considered. Occluded should mean that no visible part of the object is in the scene, therefore impossible to detect by an object detector. Nonetheless, as seen in the image, the GT considers an object occluded when it is not completely visible. Even some branches make a car occluded for the GT xml. Moreover, non-occluded GT yields worst than normal GT.

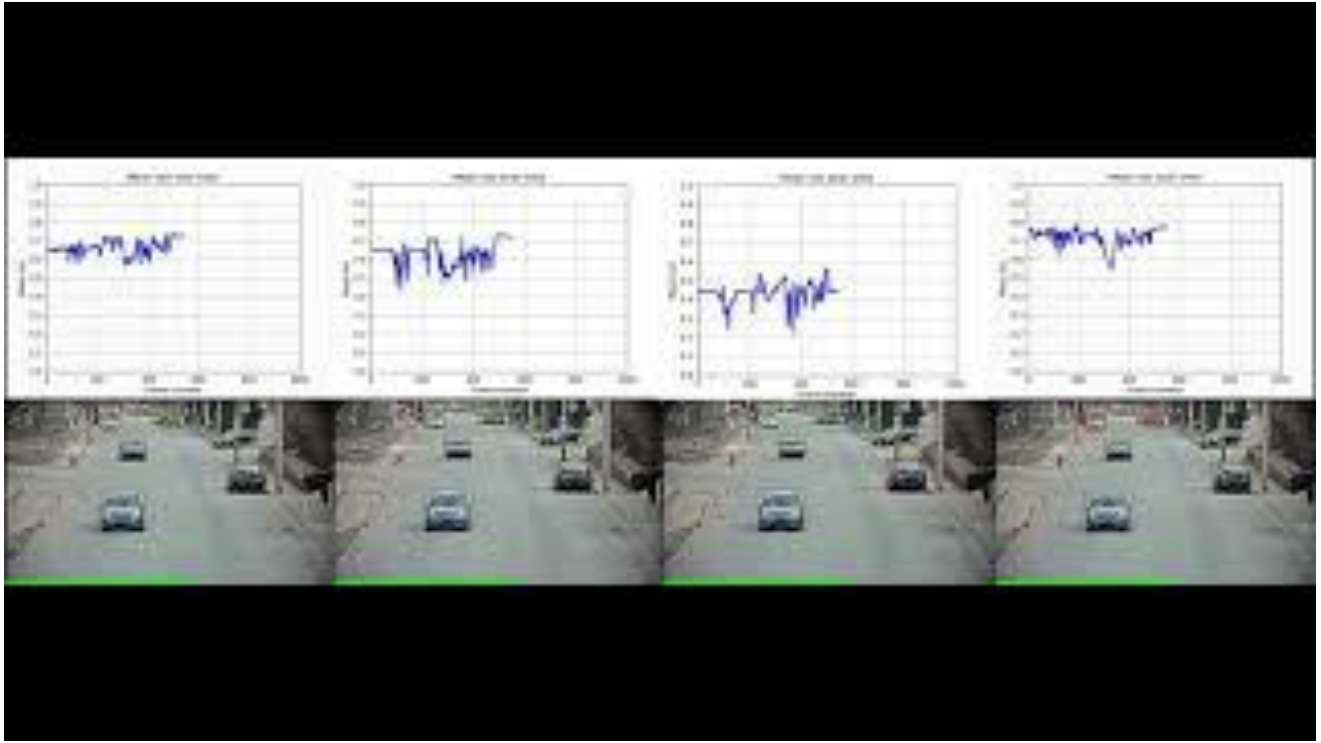


	Model	Implementation	Confidence threshold	mIoU	mAP	Inference time (img/s) (RTX 3060 12GB)
One-Stage	YOLOv5	YOLOv5 Github	0.25	.5420	**.5544**	**13.65**
			0.5	.4365	0.4619	
	YOLOv8	YOLOv8 Github	0.25	**.5770**	.5422	12
			0.5	.4852	.4906	
Two-Stage	Faster-RCNN	Detectron2	0.25	.5315	.5145	4.22
			0.5	**.5830**	.5065	
	RetinaNet	Detectron2	0.25	0.4203	**.64011**	5.35
			0.5	.4606	.5328	

- We believe that the big gap in inference time between YOLO models and FasterRCNN/RetinaNet for these two reasons:
  a. The framework detectron2 is much more slower than the Ultralytics framework.
  b. YOLO models are one stage object detectors, while Faster-RCNN and RetinaNet are two stage object detectors
- As expected, if we only keep detection with higher confidence, the mIoU increases, but mAP decreases.

Task 1.1: Object Detection: Off-the-Shelf (Team 2) [1/2]

Faster R-CNN	RetinaNet	YOLO	DETR
mIoU = 0.6685	mIoU = 0.6294	mIoU = 0.4392	mIoU = 0.7093



In addition to being the detector with the lowest score, YOLO is also the least stable one, as can be seen by its oscillating IoU curve. Faster R-CNN seems to be the most robust and reliable detector, followed by DETR.

YOLO sometimes misses the cars at the front, and rarely does it capture the ones at the back. DETR on the other hand can detect all of them reliably. Faster RCNN and RetinaNet are somewhere in the middle.

YOLO bounding boxes are quite discontinuous in time. The rest of models consistently detect each of the objects along time without missing frames.

* bounding boxes with confidence lower than 0.5 were filtered out

Task 1.1: Object Detection: Off-the-Shelf (Team 2) [2/2]

Model	Implementation	Confidence threshold	$AP_{0.50}$	$AP_{0.75}$	$AP_{0.90}$	Inference time (s/img)
Faster R-CNN	FAIR/Detectron2	.50	.5316	.4229	.2428	0.1715
		.75	.5316	.4229	.2428	
		.90	.4542	.4229	.2428	
RetinaNet	FAIR/Detectron2	.50	.5312	.4528	**.3001**	0.1340
		.75	.2727	.2727	.2392	
		.90	.1818	.1818	.1734	
YOLOv8	Ultralytics	.50	.3636	.3601	.1450	**0.0578**
		.75	.0909	.0909	.0909	
		.90	**.0909**	**.0909**	**.0909**	
DETR	FAIR/detr	.50	**.6708**	**.4840**	.2179	**0.8760**
		.75	.6708	.4840	.2179	
		.90	.6088	.4339	.2179	

* green/red highlight indicates column-wise best/worst

* we also consider parked cars in the GT, as these detectors can also deal with static objects

* Since pick-up trucks are annotated as "trucks", we also include the "truck" label as a valid prediction for the models.

YOLO is **faster** than the rest of detectors by an order of magnitude, while the transformer-based method **DETR**, is the **slowest**. This makes sense, as YOLO is a one-stage detector, and DETR performs a bipartite graph matching in the loss computation. The slow inference time of RetinaNet is disappointingly low, probably because of the complex Feature Pyramid Network it incorporates.

There seems to be a **tradeoff** between **accuracy** and **inference time**: faster methods produce less accurate detections than slower ones.

YOLO has **low confidence** predictions, with having a confidence between 50% and 75%, as indicated by the sudden jump in AP when switching the confidence threshold.
RetinaNet is also quite unsure abouts its predictions. The contrary occurs with Faster-RCNN, which is more robust.

In addition to producing detections, DETR is also the **most accurate** and most **robust** predictor: even after increasing the confidence threshold to 90%, it retains most of its predictions, and the AP suffers little.

RetinaNet and Faster R-CNN have the lowest drop in AP when increasing the IoU threshold. This indicates that they produce **more accurate**, less sketchy bounding **boxes** than YOLO and DETR.

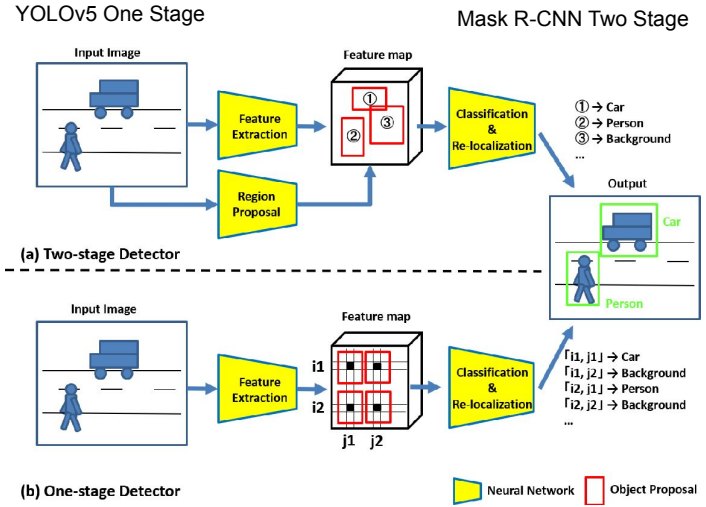
Task 1.1: Object Detection: Off-the-Shelf (Team 3)[1/2]

Used detectors, **off-the-shelf, without touching anything:**

- Two stage detectors:
  - **·Faster-RCNN**
  - **·Mask-RCNN**
- One Stage detectors:
  - **·YOLOv5**
  - **·YOLOv8**

Two-stage detectors, such as **Faster R-CNN** and **Mask R-CNN**, work in two stages. The first stage generates region proposals, which are potential object locations in the image. The second stage classifies these proposals as object or background and refines the object location. This approach typically provides better accuracy but is slower than one-stage detectors.

On the other hand, one-stage detectors, such as **YOLOv5** and **YOLOv8**, work in a single stage. They detect objects directly by dividing the input image into a grid of cells and predicting bounding boxes and class probabilities for each cell. One-stage detectors are generally faster but may sacrifice some accuracy compared to two-stage detectors.



Task 1.1: Object Detection: Off-the-Shelf (Team 3) [2/2]

Type	Model	Implementation	Threshold	mIoU	mAP	inference time
Two-Stage	**Faster-RCNN**	Pytorch	0.5	0.521	0.583	0.32s
			0.7	**0.560**	**0.614**	
	Mask-RCNN	Pytorch	0.5	0.586	0.528	1.58s
			0.7	**0.623**	**0.567**	
One-Stage	**YOLOv5**	Ultralytics	0.5	0.392	**0.432**	**0.06s**
			0.7	**0.427**	0.413	
	YOLOv8	Ultralytics	0.5	0.438	**0.521**	0.08s
			0.7	**0.494**	0.513	

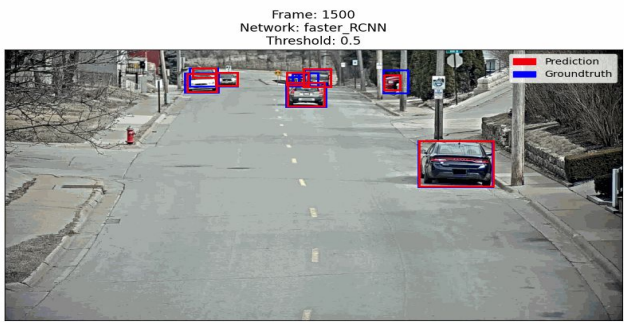
When comparing two-stage detectors and one-stage detectors, we can observe that two-stage detectors like **Faster R-CNN** and **Mask R-CNN** typically provide better accuracy but are slower than one-stage detectors like **YOLOv5** and **YOLOv8**. One-stage detectors have a simpler architecture, with fewer computations needed to make predictions, and can therefore process images much faster. However, they may sacrifice some accuracy, especially for smaller objects or in cluttered scenes.

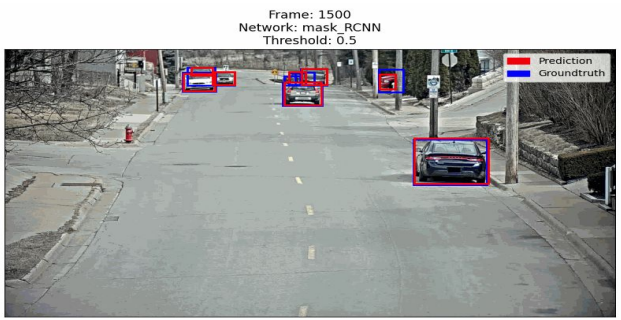
The choice between a two-stage detector and a one-stage detector depends on the specific requirements of the application. For applications that require high accuracy, such as medical imaging or autonomous driving, a two-stage detector like **Mask R-CNN** may be preferred. For applications that require faster processing speed, such as real-time video analysis or surveillance, a one-stage detector like **YOLOv5** may be more suitable.

Task 1.1: Object Detection: Off-the-Shelf (Team 4)

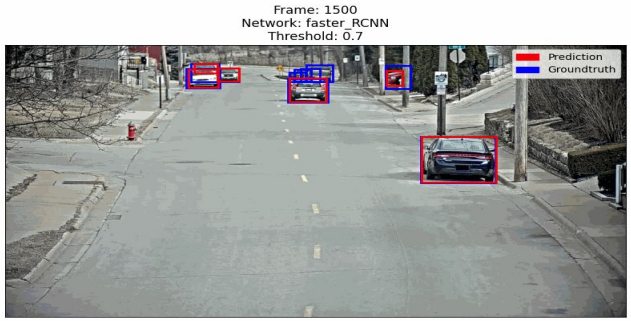
Model	Implementation	Confidence threshold	mAP
Faster-RCNN	Detectron2 (faster_rcnn_X_101_32x8d_FPN_3x)	0.5	0.4039
		0.7	0.4213
Mask-RCNN	Detectron2 (mask_rcnn_X_101_32x8d_FPN_3x)	0.5	0.4143
		0.7	0.4212

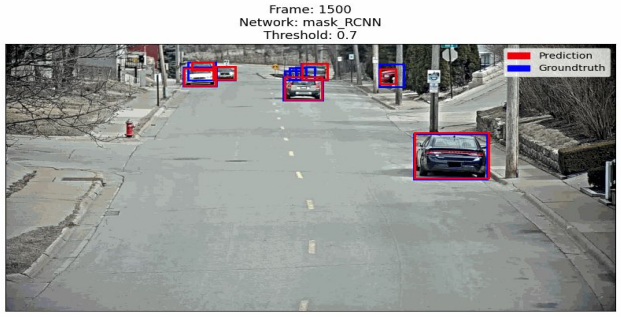
As we can see, Mask-RCNN gets slightly higher quantitative results in terms of mAP using a threshold of 0.5, but qualitatively both networks are able to detect the same cars with almost perfect bounding boxes compared to the GT.





Using a threshold of 0.7 the mAP of both networks increased, specially for the Faster-RCNN that is able to reach the same mAP than Mask-RCNN when both use this threshold value.





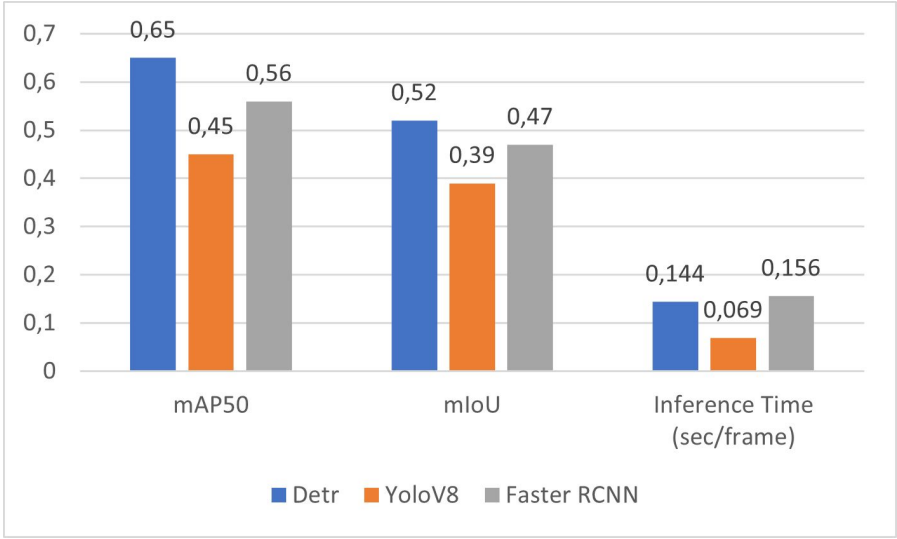
Notice in the GIFs that the vans, trucks and similar vehicles are not detected, thus reducing the mAP since they are in the GT. This is because we filtered the results to just cars (class 3 of COCO) as indicated by the task instructions. The GT makes no distinction between them so we could not filter them in the GT.

Task 1.1: Object Detection: Off-the-Shelf (Team 5) [1/2]

Executed on a RTX 3060

Pretrained models used:

- **DETR** (Resnet50 backbone): CNN and transformer based model (implementation).
- **YoloV8** (YoloV8x version): one-stage CNN based model (implementation).
- **Faster RCNN** (Resnet50_FPNv2 backbone): two-stage CNN based model (implementation).



Implementation details:

- All of the models were pre-trained on the COCO dataset.
- We selected both parked and non-parked cars and bicycles.
- Only detected bounding boxes with a confidence score greater than 0.5 were used.

Conclusions:

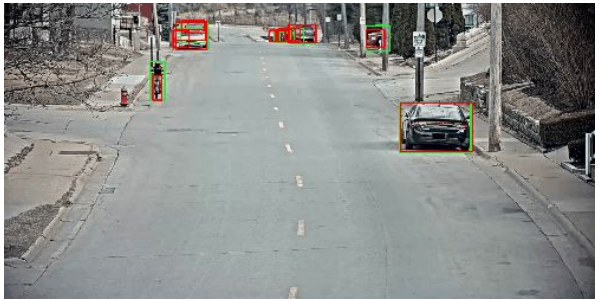
- Although YoloV8 had the fastest inference time thanks to being a one-stage CNN, it yielded the worst results.
- Faster RCNN had worse results than DETR, with even longer inference time.
- DETR showed the best performance in terms of mAP and mIoU.

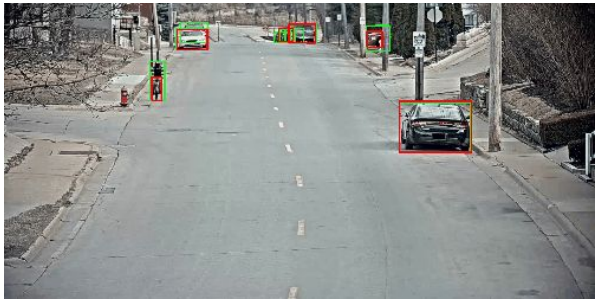
Task 1.1: Object Detection: Off-the-Shelf (Team 5) [2/2]

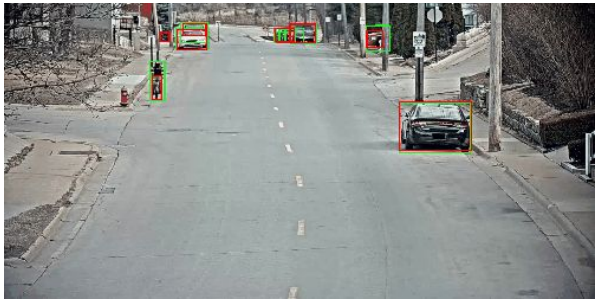
Ground truth box

Detected box

Only showing car and bicycle detections







DETR

YoloV8

Faster RCNN

- The results show that both Faster RCNN and DETR networks outperform YoloV8 in terms of predicting more accurate and robust bounding boxes.
- All models struggle to detect distant cars, with YoloV8 performing the worst.
- The networks detect the class bicycle object as only the bicycle, whereas in the ground truth it is annotated with both the bicycle and the person riding it.
- Additionally, pretrained models fail to recognize vans and pick-ups as cars, often labeling them as trucks, despite being annotated as cars in the ground truth.

Task 1.1: Object Detection: Off-the-Shelf (Team 6) [1/2]

Detectron2

Faster R-CNN:
- Two-stage object detection model
- Uses Region Proposal Network (RPN) to propose candidate regions
- Classifies objects and refines bounding box location in second stage
- Backbone →X101-FPN

Mask R-CNN:
- Extension of Faster R-CNN that adds a third stage for object mask prediction
- Generates binary mask for each proposed region
- Useful for detailed segmentation of objects, such as in medical imaging or robotics
- Backbone →X101-FPN

RetinaNet:
- Extension of Faster R-CNN that adds a third stage for object mask prediction
- Single-stage object detection model
- Uses focal loss function to improve model's ability to detect objects with rare or difficult-to-detect features
- Backbone →R101

Network	Confidence threshold	AP_{50}	mIoU	Inference time (s/iter)*
Faster R-CNN	0.5	0.48	0.47	0.098
Mask R-CNN	0.5	0.49	0.48	0.103
RetinaNet	0.5	**0.52**	0.42	**0.054**

Faster R-CNN and Mask R-CNN have similar performance in terms of AP50 and mIOU. The two models are comparable in terms of their ability to detect cars, at least at the confidence threshold of 0.5 that we have set.

RetinaNet has a higher AP50 indicating that is able to detect cars more accurately than the other two models. However, its mIoU is lower, suggesting that some of the bounding boxes are not aligned as well as in the two previous networks. In terms of inference time, RetinaNet is the fastest so it'll be a good choice if speed is a critical factor.

Right now we will focus our work on **Faster R-CNN and RetinaNet**

(*) inference time extracted from the model zoo repository

Wu, Y., Kirillov, A., Massa, F., Lo, W.-Y., & Girshick, R. (2019). Detectron2. Retrieved from https://github.com/facebookresearch/detectron2

Task 1.1: Object Detection: Off-the-Shelf (Team 6) [2/2]

Ground truth
Predicted

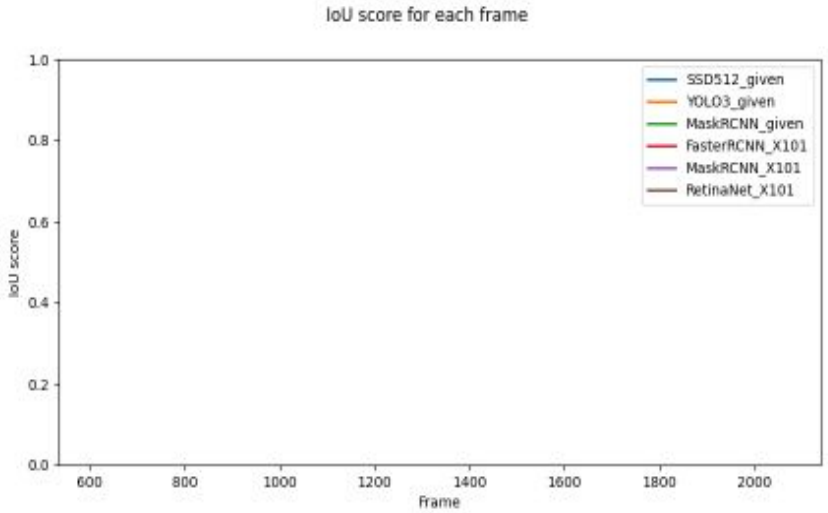
It is observed in the video sequences that both models RetinaNet and Faster R-CNN can detect moving cars located far away from the camera.

To evaluate the adjustment of the bounding boxes, we plot the mean Intersection over Union (mIoU) per frame for the three networks mentioned earlier (Faster R-CNN, Mask R-CNN, RetinaNet), as well as for the "given" data from YoloV3, SSD512, and an unknown backbone Mask R-CNN. The first three networks clearly outperform the given ones in terms of mIoU. Furthermore, when compared to the results obtained in the first week of the project, the average precision (AP) is also higher.

It is worth mentioning that all detectors exhibit good performance on close cars. In week 1 we saw that SSD and YoloV3 seem to struggle with occlusions and small objects, such as cars far away. In the video, it can be observed that RetinaNet and Faster R-CNN address this issue effectively.

The new models also give lower confidence scores on vehicles that are not cars, such as a van appearing in the last frames.

RetinaNet R101 | Faster R-CNN X101



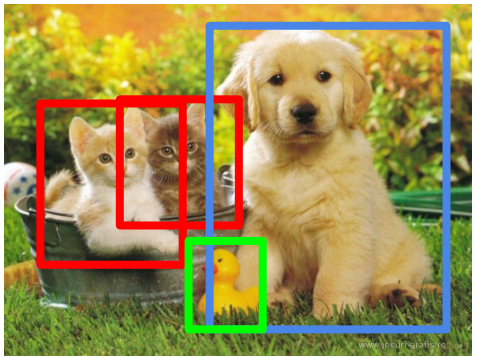
Wu, Y., Kirillov, A., Massa, F., Lo, W.-Y., & Girshick, R. (2019). Detectron2. Retrieved from https://github.com/facebookresearch/detectron2

Task 1.1: Feedback

	feedback
Team 1	Good and clear explanation of classes used from detectors and GT 4 object detectors used with mIoU and mAP reporting with links to implementation (missing references) Bonus point for reporting inference time and very good discussions (e.g. analysis through time) Missing explanation with differences between methods. Conclusion: RetinaNet best mAP, yolov5 fastest
Team 2	Good qualitative analysis of the various methods YOLOv8 results seem wrong? Interesting analysis of the robustness of the detectors, but conclusions not clear. The two last sentences in slide 2 do not seem to say the same. Inference time provided. Good! 4 object detectors used.
Team 3	Bonus point for explanation of one-stage vs two-stage detectors and inference time. 4 object detectors used. Missing qualitative evaluation discussions. Missing references. Missing discussion about ground truth classes used. Bold in tables? Missing explanation with differences between methods. Conclusion: FasterRCNN best mAP, yolov8 fastest
Team 4	
Team 5	Bonus point for a clear explanation of classes used and confidence and inference time. 3 detectors used with mAP and mIoU reporting with links to implementation. Bonus point for a minimal explanation of differences between detectors. Conclusion: DETR best mAP, yolov8 fastest
Team 6	Three methods tested Good details and explanations of the methods use (backbone, etc.) Results for mAP, IoU and inference time provided. Good! Good qualitative analysis of the results, with a plot of IoU along frames.

Tasks

- Task 1: Object detection
  - Task 1.1: Off-the-shelf
  - **Task 1.2: Annotation**
  - Task 1.3: Fine-tune to your data
  - Task 1.4: K-Fold Cross-validation
- Task 2: Object tracking

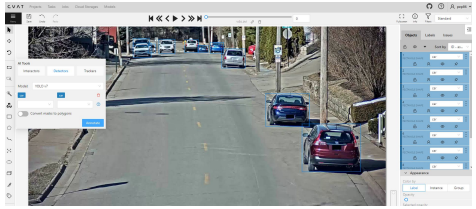


Task 1.2: Object Detection: Annotation (Team X)

- Copy & paste (max 2 pages)

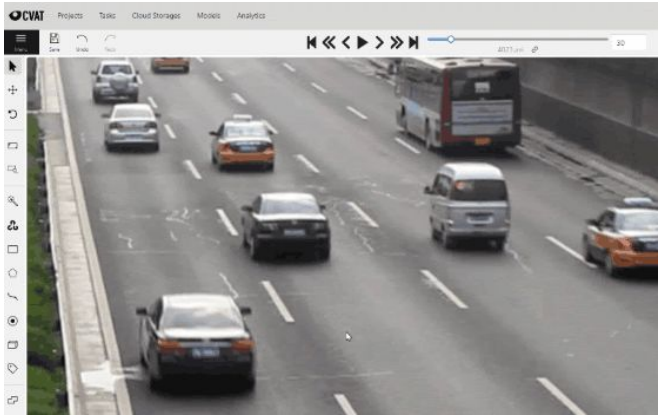
Task 1.2: Object Detection: Annotation (Team 1)

To label the sequence AICity_data_S05_C010 CVAT has been used. This tool helps annotating multiple task, such as segmentation and object detection. Nonetheless, we focus on the integrated AI Tools for object detections, that are very helpful to annotate very fast (In the case that your classes you want to annotate are in the COCO dataset).



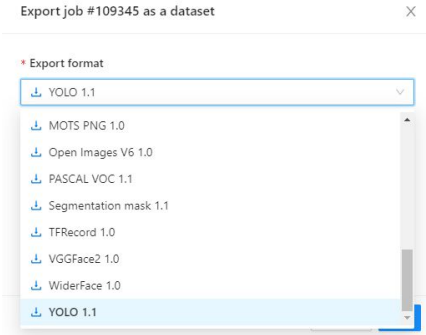
AI Detector Tool: Map your classes to COCO ones, and does inference on models such as YOLOv7 to get the detections. This tool is very slow for video sequences.

Draw rectangles: If certain bounding box has not been detected, it can be manually annotated.



Trackers: Apart from using AI object detectors, we also use a tracking AI algorithm of CVAT that creates annotation also in the time axis, as seen in the GIF [1]

Manual Trackers: For certain detections that AI tracker is not detecting, we used the OpenCV tracker, which simply creates a bounding box that in each iteration it has to be manually moved, and once it is out of the frame you can manually disable the tracker. In the end it is more comfortable than creating in each frame a bounding box.



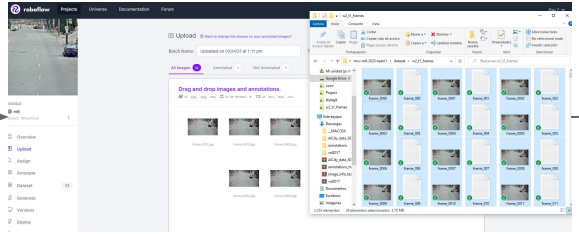
Export annotations: Finally CVAT allows us to export the annotation in lots of different formats.

[1] - CVAT: Annotation Tool for Computer Vision [2023 Tutorial]

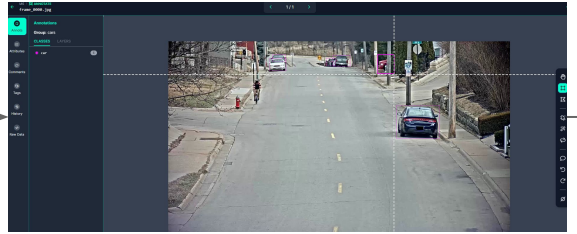
Task 1.2: Object Detection: Annotation (Team 1) - Plan B

The method mention before had a drawback, which is that some of the features can be used only with a premium plan. As we are poor students, we have created a pipeline to create a good dataset.

get_labels_dataset.py



Upload predictions of every frame to Roboflow.

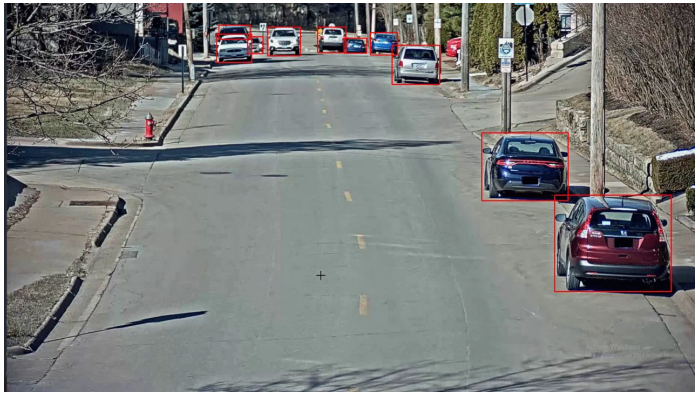


Manually move or delete bad detections

Export dataset (very good and interesting data augmentation could be done in this step)

In this Python script, we perform inference on the **YOLOv5** model, which has yielded the best results. Specifically, we use the largest available weights for this model (**yolov5x**) and perform **inference on the complete image**, instead of using the default resizing to 640 pixels that YOLO typically uses.

Although this approach results in slower inference times, our primary goal is to create a **high-quality dataset**, rather than optimizing for speed.

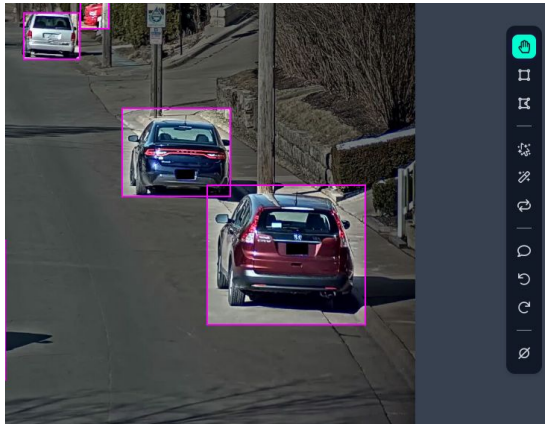


Own GT

[1] - CVAT: Annotation Tool for Computer Vision [2023 Tutorial]

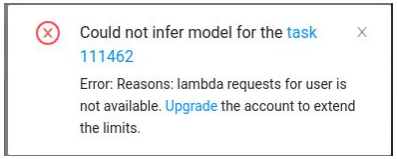
Task 1.2: Object Detection: Annotation (Team 2)

We initially tried **Roboflow** but quickly discarded it because it only offered manual annotation for free. AI functionalities where pay-walled.



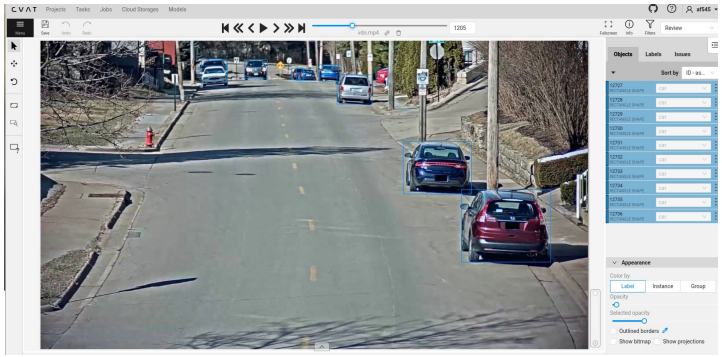
We also tried **LabelStudio**. We found it does not support .avi so we converted the video to .mp4. Even so, it was necessary to create a complicated "ML backend" which is just overkill and too cumbersome. So we discarded LabelStudio as well.

We finally settled for **CVAT**. It does not support free automatic annotation, so we generated annotations with Faster-RCNN is KITTI format, uploaded them, filtered them on the platform, and then exported them to KITTI and Pascal VOC formats.



CVAT does not offer free automated annotation.

We imported our Faster-RCNN-generated annotations in KITTI format.



The fairly unintuitive and slow CVAT GUI allowed us to filter and improve the automatic annotations.

We found all three annotation platforms to be unintuitive, user averse, slow (e.g. when displaying results and uploading annotations), and cumbersome in general.

We think that these platforms may have a place in situations where a large team of annotators must be coordinated, but not when a small group of AI professionals is capable of managing an automated annotator such as Faster R-CNN.

These platforms sell "automated annotation capabilities" that any computer-informed person can obtain from a Keras RCNN tutorial for free...

Task 1.2: Object Detection: Annotation (Team 3) 1/2

RoboFlow:

We tried multiple annotation tools such as CVAT and LabelStudio but we finished using RoboFlow since it doesn't need installation and has a lot of features that you can try for free (training a model with your own labeled data).

For annotating, we did it manually for the first 100th frames using the pre-trained model which had a lot of failed detections.
After that, we trained the model with our labeled data using as data augmentation the horizontal flip and adding random black squares (to make the model more resistant to object occlusions).

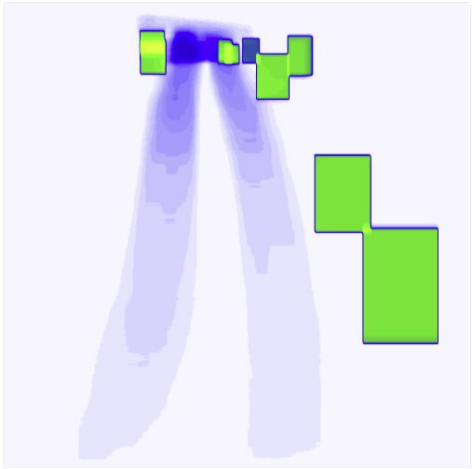
The labeling tool had trouble detecting objects that were really close together, and most of the times it thought they were just one object. It was also hard to detect objects that were occluded by something else, so we had to annotate those manually most of the time.

Failed detection for the label assistant

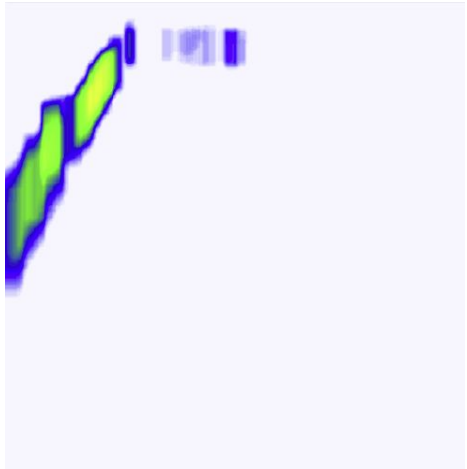
Task 1.2: Object Detection: Annotation (Team 3) 2/2

Our annotation results showed that we labeled approximately 45,000 cars and 1,000 people. However, we can appreciate in the Class Balance graph that the person class is significantly underrepresented, making this dataset less valuable for training purposes and more suitable for testing.

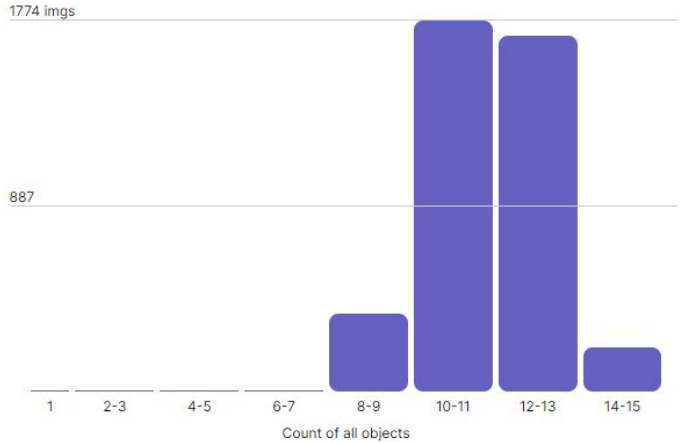
In addition, Roboflow provided us with helpful insights into the dataset, including heatmaps for different classes and a histogram displaying the frequency of objects per image.



Annotation hitmap for cars



Annotation hitmap for people



Histogram of objects count by Image

Task 1.2: Object Detection: Annotation (Team 4) [1/2]

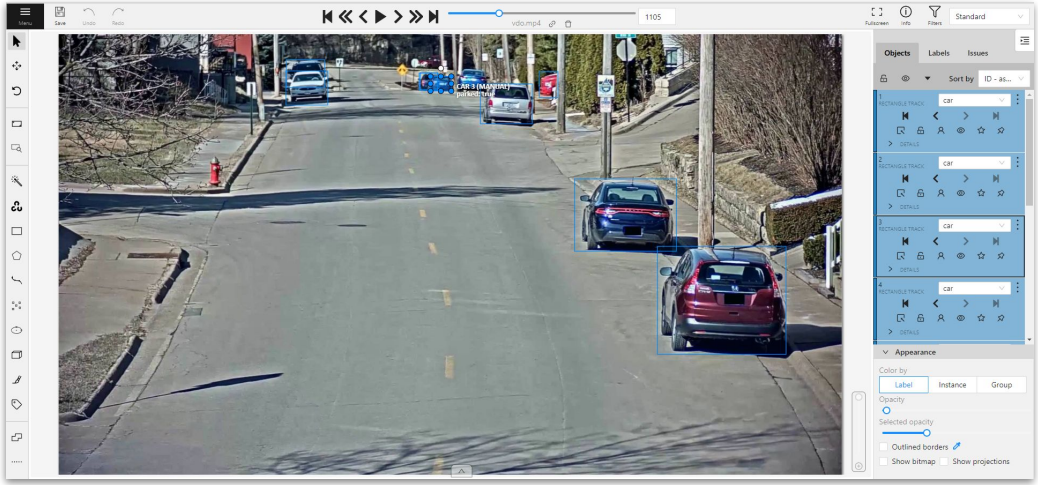
To do this task we decided to use CVAT, which is an open source annotation tool. After some use we notice some good and bad things about it:

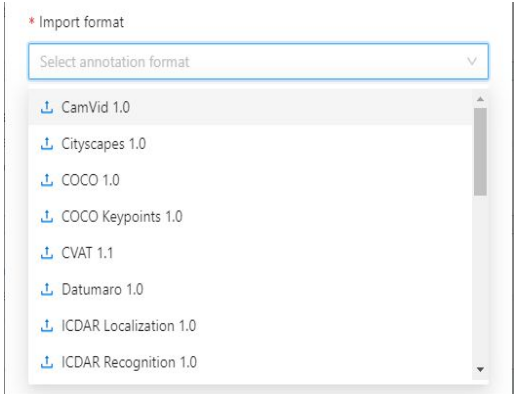
Bad:

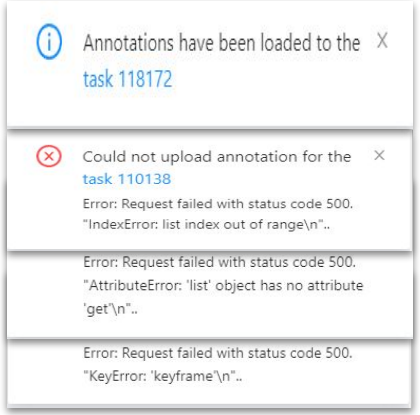
- Is cumbersome to use and the debugging process for uploading some annotations is totally unintuitive
- Its interface sometimes is really slow

Good:

- The interface for annotations is intuitive, it allows you to do several things such as zooming in and out
- It includes nice tools that allow faster annotations such as interpolation
- It allows the use of precomputed annotations in different formats, such as COCO or CVAT







Task 1.2: Object Detection: Annotation (Team 4) [2/2]

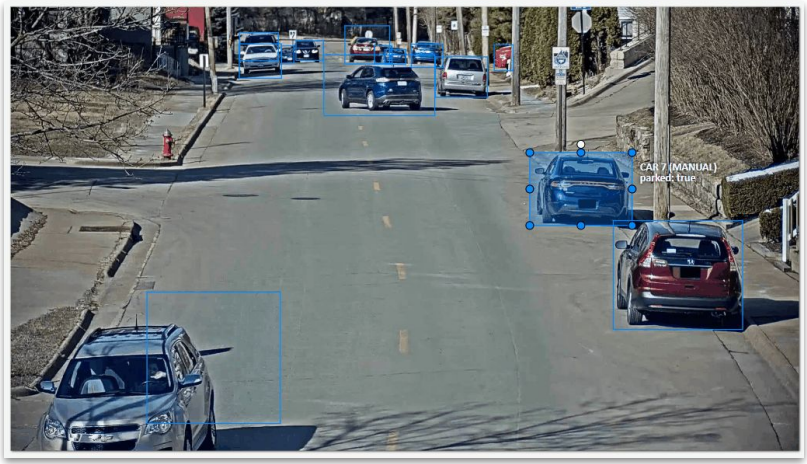
We have noticed that we can create the annotations in an efficient form in two ways:

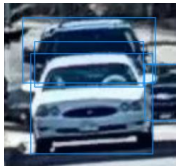
1. Use initial annotations coming from neural network predictions on the video such as Faster-RCNN where we:
   - Create predictions as an initial groundtruth with the NN
   - Upload the annotations to CVAT
   - Manually annotate and remove incorrectly predicted annotations
2. Use only interpolation or OpenCV tracker, which allows the annotation of several frames for the same object at the same time. The reason to not use the initial predictions from a neural network is that sometimes the quality of them is really poor.

Also, we observe that the time spent for the annotations depends on a lot of factors, such as the quality of the initial predictions, number of frames, number of objects…

Overall the time spent on this task was a lot, because of bugs with the used programs and incompatibilities. However, it was also a good learning experience. For only the annotation part with interpolation, we could do it in more or less 3 hours.

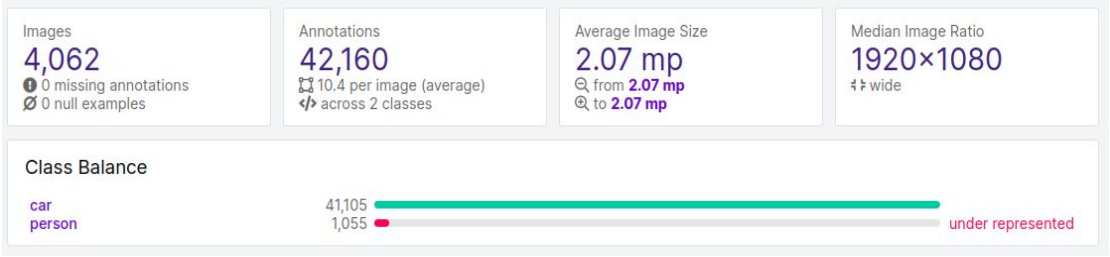
Video with annotations





Sometimes the initial predictions are really poor, as in this case. For parked cars and shadows this is really troublesome because those predictions stay during all the video which makes annotation difficult.

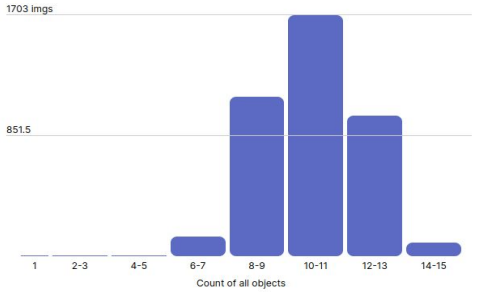
Task 1.2: Object Detection: Annotation (Team 5) 1/2



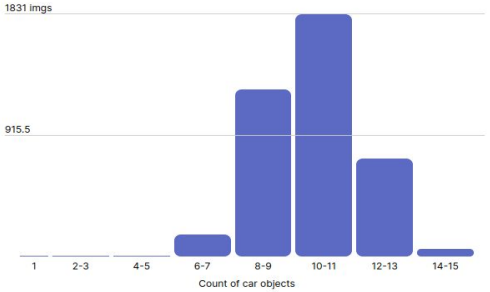
The Detailed analysis of the annotations: The video frames has a huge class imbalance between car and person.

Histogram of Object Count by Image

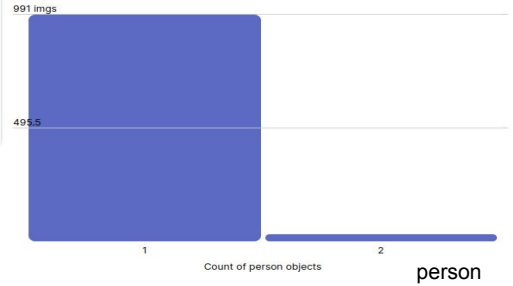
From the histogram counts it has been observed that, there is minimum 6-7 cars in most of the frame, whereas, there is barely 2 person in a single frame. So the dataset is completely biased to the car.



All



car

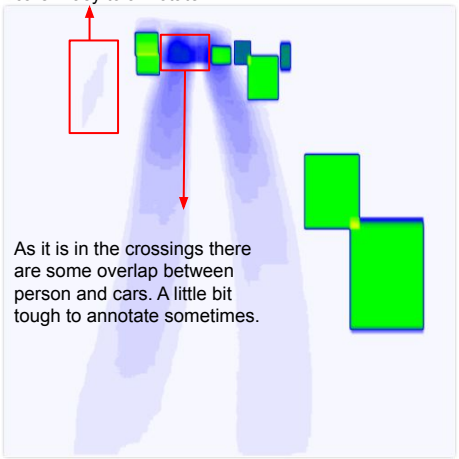


person

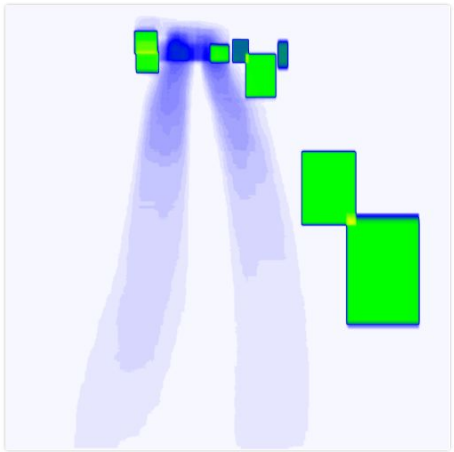
As it is in the footpath there is no overlap between person and cars. Easy to annotate.

As it is in the crossings there are some overlap between person and cars. A little bit tough to annotate sometimes.

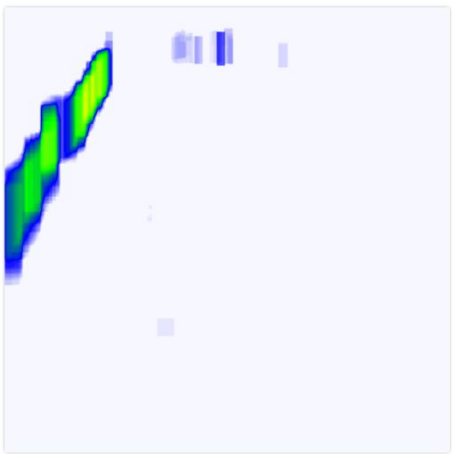
Annotations Heatmap



All classes



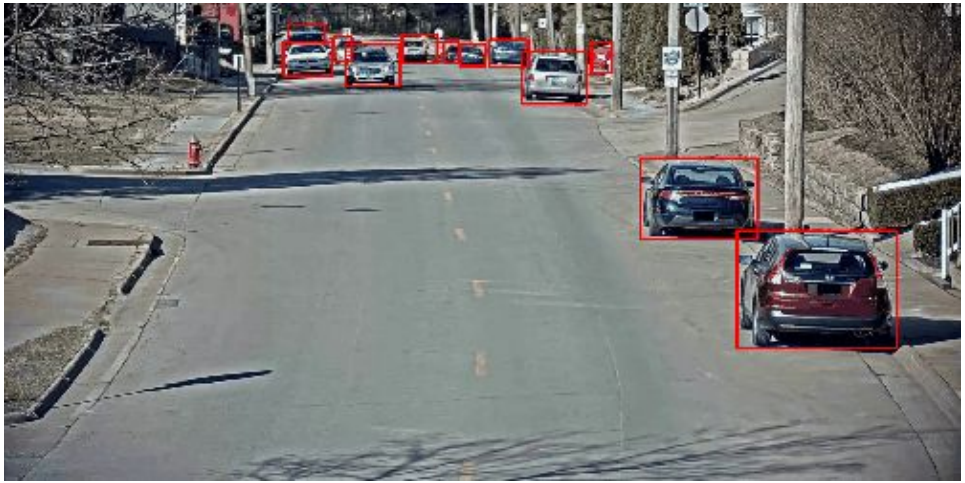
Car



Person

Overall, the annotations are more easy and accurate when the overlap percentage between the two objects is very less. That's why it the parked cars and the person walking in the footpath easy to annotate. Whereas, the more errors is produced in the crossings where we observed a huge number of overlap.

Task 1.2: Object Detection: Annotation (Team 5) 2/2

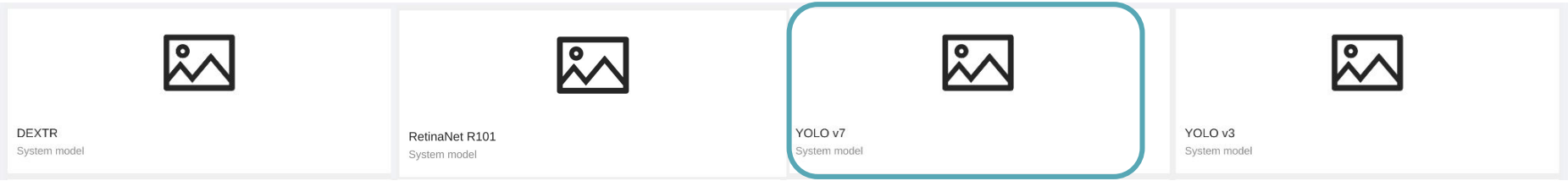


- Roboflow is a very efficient software for the annotations and easy to use (ref).
- For annotations we have plugin an object detection model trained on ms coco dataset (55.8% mAP). Which gives the label assistance first and then we manually corrected the labels.

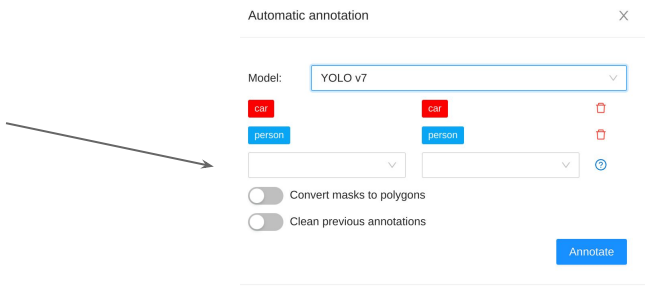
- The labeling tool had trouble distinguishing between objects that were very close to one another, and the majority of the time, it mistook them for one object. We had to manually annotate occluded objects most of the time because it was difficult to detect them.
- It is very hard to keep a track of the sequence during the annotations that's why frames are not properly aligned. After annotations we have to right a script for proper alignment.

Task 1.2: Object Detection: Annotation (Team 6) [1/2]

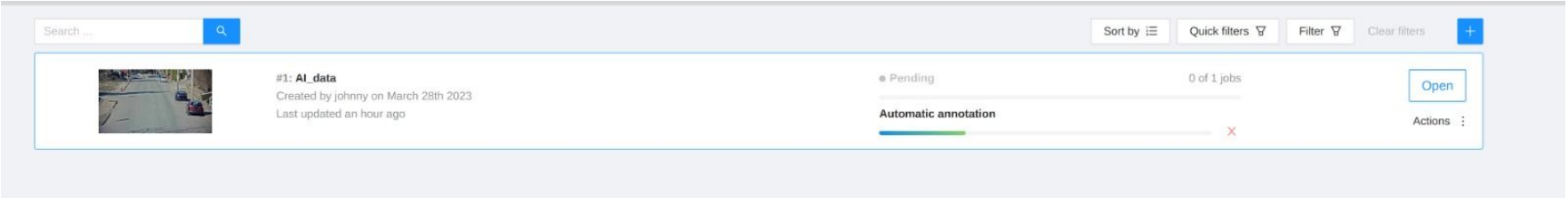
1. We deploy some models to do automatic detection. It can be done by custom model fine-tuning or a model that provides CVAT. [1] We use yolov7.



2. We put two custom classes from coco format. Car to Car and Person to Person.

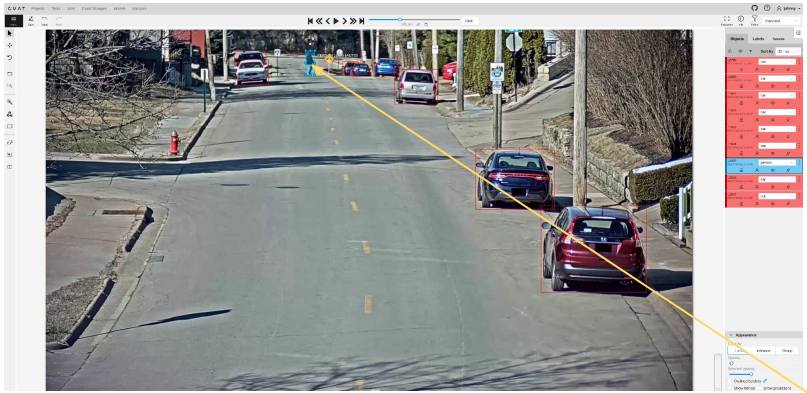


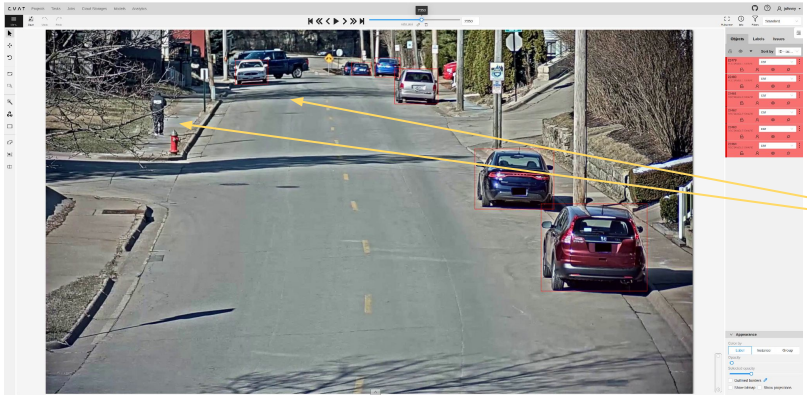
3. Run automatic annotation using RTX 3090.



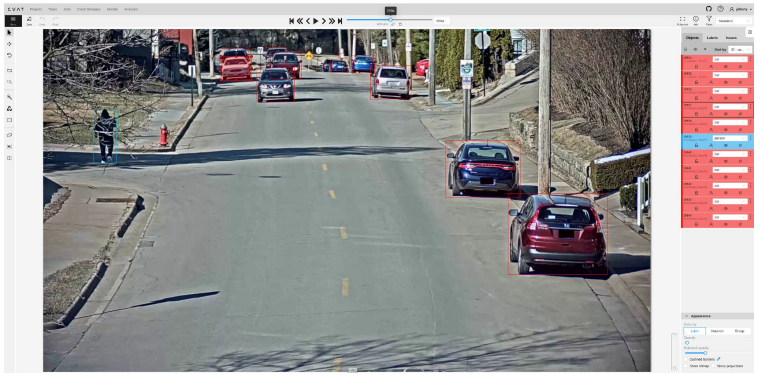
Task 1.2: Object Detection: Annotation (Team 6) [2/2]

4. Visual supervision and manual changes





Perfect Frame



Person detected

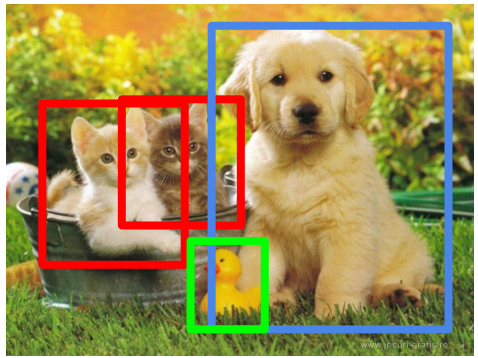
Some frames must be corrected manually

Task 1.2: Feedback

	feedback How long did it take?
Team 1	CVAT: yolov7 slow so did you use manual annotations? Roboflow: completely separated from CVAT? How many bounding boxes?
Team 2	CVAT used. Good explanation of why Roboflow and LabelStudio were discarded.
Team 3	Roboflow Good idea to train on the sequence itself. Bonus point for the analysis of the annotation.
Team 4	CVAT + FasterRCNN. Good analysis of the usability of the tool. Explanation of the workflow followed. Good. Time to annotate 3h. Good to provide this information.
Team 5	Try to organize the slides better. Roboflow is mentioned at the end. Bonus point for the analysis of the annotation. A script? explain? Plugin vs team 1 poor student's alternative? ;)
Team 6	CVAT + Yolov7 Automatic annotation. Other teams claim that this is not a free feature?

Tasks

- Task 1: Object detection
  - Task 1.1: Off-the-shelf
  - Task 1.2: Annotation
  - **Task 1.3: Fine-tune to your data**
  - Task 1.4: K-Fold Cross-validation
- Task 2: Object tracking



Task 1.3: Object Detection: Fine-tune (Team X)

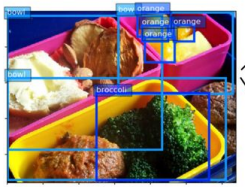
- Copy & paste (max 2 pages)

Task 1.3: Object Detection: Fine-tune (Team 1) - [1/2]

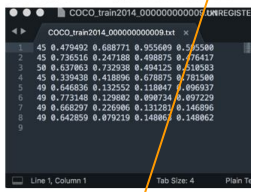
In this task we have opted to finetune YOLOv8, the last YOLO model from the [ultralytics framework](https://docs.ultralytics.com/).

First, it has been necessary to first convert the VOC format datasets to the YOLO format. Convert the Pascal VOC xml file to a series of .txt files that have the annotations for every frame.

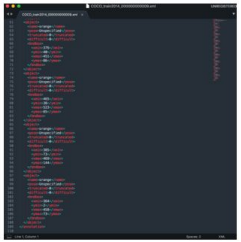
COCO_train2014_000000000009.jpg



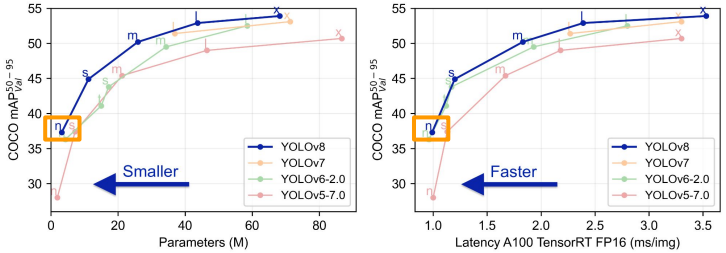
COCO_train2014_000000000009.txt



COCO_train2014_000000000009.xml

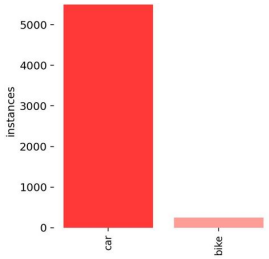


The version selected to finetune is **yolov8n**, the smaller YOLOv8 model, with **3M parameters** approx. The smaller model has been selected in order to have a faster training time and allow us to experiment more and because of hardware limitations.



Comparison between YOLO family models

In this task the first **25% frames** will be selected for **training**, and **the rest for validation**.
The sequence consists mostly of car instances, so we can expect a worse performance detecting bikes.



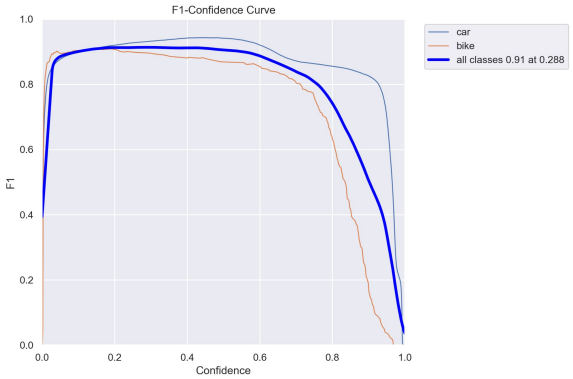
Ultralytics documentation: https://docs.ultralytics.com/

Task 1.3: Object Detection: Fine-tune (Team 1) - [2/2]

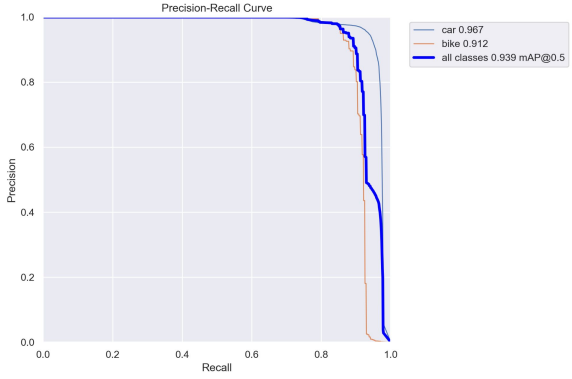
YOLOv8 performs a very interesting augmentation which is the mosaic augmentation. However, this augmentation is empirically shown to degrade performance if performed through the whole training routine. It is advantageous to turn it off for the last ten training epochs.

Augmented batch with mosaic augmentation

Model	Optimizer	validation mAP50		
		car	bike	all
YOLOv8	SGD (momentum =0.937)	0.967	**0.912**	**0.939**
	Adam	**0.972**	0.858	0.915
	AdamW	0.972	0.830	0.901
	RMSProp	0.864	0.421	0.642



F1-Confidence curve, we see that with a confidence around 0.3 it achieves the best score



PR curve for the SGD experiment

Tested all the available optimizers for the YOLOv8 model with an initial learning rate of 0.01.
As expected, bike detection is worse than car. The best performing model is the one with SGD, although Adam performs better for car detection.

Task 1.3: Object Detection: Fine-tune (Team 2) [1/2]

Model	Implementation	Confidence threshold	$AP_{0.50}$	$AP_{0.75}$	$AP_{0.90}$
Faster R-CNN Pretrained	FAIR/Detectron2	.50	.5316	.4229	.2428
		.75	.5316	.4229	.2428
		.90	.4542	.4229	.2428
Faster R-CNN Finetuned (first 25%)	FAIR/Detectron2	.50	.529	.512	.414
		.75	.529	.512	.414
		.90	.529	.512	.414
Faster R-CNN Finetuned (random 25%)	FAIR/Detectron2	.50	**0.560**	**0.560**	**0.560**
		.75	0.548	0.547	0.548
		.90	0.544	0.544	0.544

From this results we want to highlight 3 key observations:

1. Under the fine-tunning conditions, the results are almost equal no matter what confidence threshold we set. In the posterior visualization we realized that the confidence for the positive classes was always around the 99% and **therefore was no "uncertainty"** for the results to vary.

2. In lower IoU thresholds there's almost no improvement (AP50 and Ap75) while in AP90 there is a huge leap. This is because the pretrained model works **well enough in the first two metrics**, there is no room to improvement from the generalization of the model perspective. In the AP90 metric, we set such a **fine-grained task** that it's crucial to adapt the model to the new data.

3. When we use a random 25% for training means that for a given a validation frame there is around 37% of probabilities that the **previous or next frame** has been used for training. Therefore results in this set are not reliable as they are the most **overfitted.**

Task 1.3: Object Detection: Fine-tune (Team 2) [2/2]

Random 25% train selection

Initial 25% train selection

In both scenarios the confidence per class varies from 99% to 100% in almost every case. This is the reason why the previous results were almost invariant to the confidence of the retrieved predictions.

Task 1.3: Object Detection: Fine-tune (Team 3) [1/2]

Fine tunning: Pre-trained model is adapted to a new task by further training it on a new dataset. The main idea behind fine-tuning is to leverage the knowledge that a model has already learned from a large dataset (usually a generic one) and transfer that knowledge to a new, task-specific dataset. This approach can save a significant amount of time and computational resources compared to training a model from scratch.

Process:

1. **Select a pre-trained YOLOv8 Ultralytics model:** Obtain a pre-trained YOLOv8 model, which has already learned features from a large dataset such as COCO. This model will serve as the starting point for fine-tuning.
2. **Prepare the new dataset:** Collect a new dataset specific to the object detection task you want the model to perform. This dataset should contain images and corresponding annotations (bounding boxes and class labels) for the objects of interest.
3. **Modify the configuration file:** Adjust the YOLOv8 configuration file (.yaml) to accommodate the new dataset.
4. **Initialize the model with pre-trained weights:** Load the pre-trained weights into the YOLOv8 model. These weights act as a starting point for the fine-tuning process, allowing the model to adapt more quickly to the new dataset.
5. **Train the model on the new dataset:** Fine-tune the YOLOv8 model on the new dataset using an optimizer like ['SGD', 'Adam', 'AdamW', 'RMSProp']. During this process, the model's weights will be updated to better adapt to the new dataset, improving its object detection performance on the specific task. To find the optimal hyperparameters for your specific task, we have performed a **grid search**, exploring various combinations of epochs, image sizes, batch sizes, learning rates, and optimizers.
6. **Evaluate and adjust:** After fine-tuning, evaluate the model's performance on a validation set from the new dataset. If the performance is not satisfactory, we may need to adjust the model architecture, learning rate, or other hyperparameters and fine-tune again.

Task 1.3: Object Detection: Fine-tune (Team 3) [2/2]

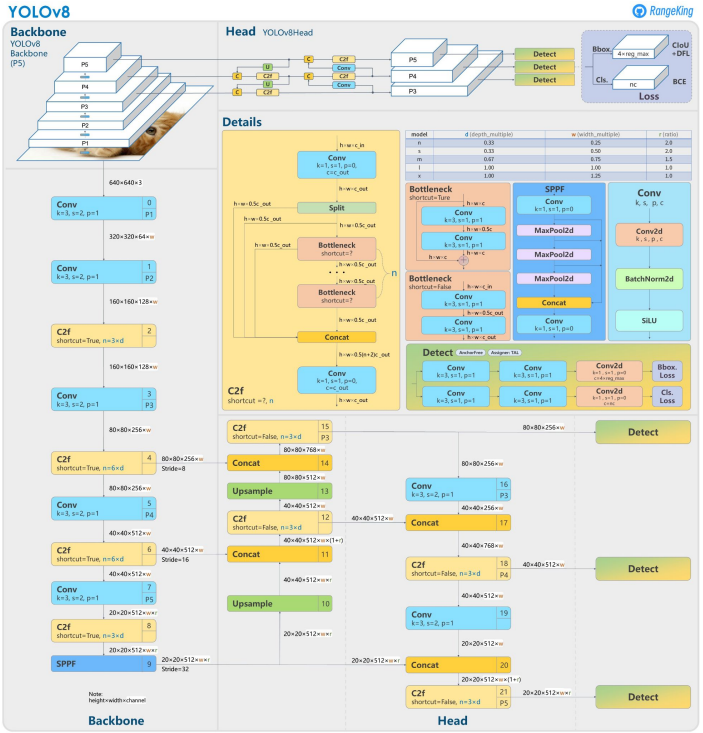
Fine-tuning a YOLOv8 model allows you to leverage the power of this state-of-the-art object detection model and adapt it to your specific task. This process can save time and resources compared to training a new model from scratch while still achieving excellent detection performance on your dataset. The grid search performed during the fine-tuning process helps identify the best combination of hyperparameters for your specific task, further improving the model's performance.

Hyper parameters that we used for grid search:
epochs_options = [50, 200]
imgsz_options = [640, 1024]
batch_size_options = [8, 16]
learning_rate_options = [0.001, 0.01]
optimizer_options = ['SGD', 'Adam', 'AdamW', 'RMSProp']

In conclusion, the fine-tuning process with YOLOv8 has enabled us to adapt a pre-trained object detection model to a new dataset and object classes. This process has been effective in leveraging the power of a state-of-the-art object detection model, while also reducing the time and resources required compared to training a new model from scratch.

The **grid search** we performed during the fine-tuning process has allowed us to explore various combinations of hyperparameters such as epochs, image sizes, batch sizes, learning rates, and optimizers. This comprehensive search has helped us identify the best combination of hyperparameters for our specific task, resulting in a more accurate and reliable model.

BUT! Using the same video with different consecutive frames, there is a high probability that the frames used for the train are also used for the test, so we have an overfitted model.



Model	Epochs	Img size	Batch size	Optim	mAP 95
YOLOv8	138	640	16	SGD	0'743

Optimal parameters after grid search

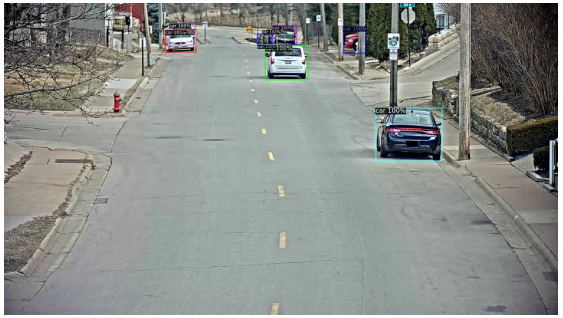
Task 1.3: Object Detection: Fine-tune (Team 4) [1/2]

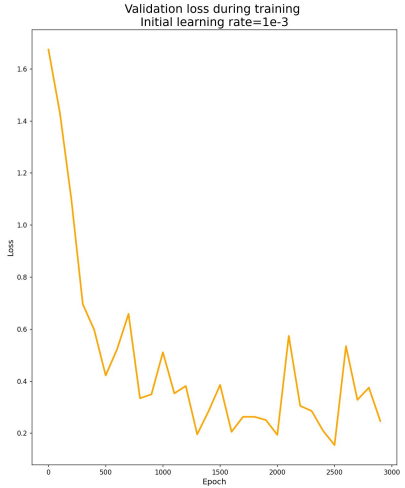
We opted to finetune the Faster-RCNN since we do not have segmentation mask GT to train the Mask-RCNN, and other networks are quite bigger and we have low computational resources.

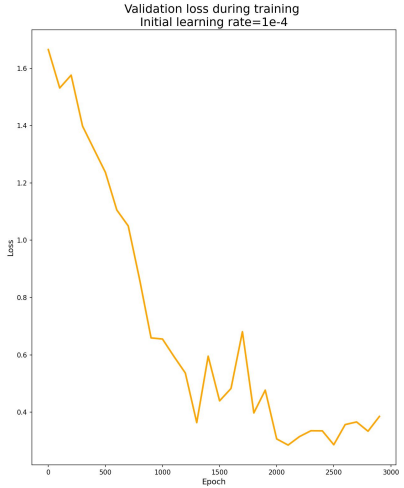
Since we used Detectron2, it was fast to adapt the COCO dataset to our new dataset of the video sequences, and to use just car class, by registering a new dataset in Detectron2.

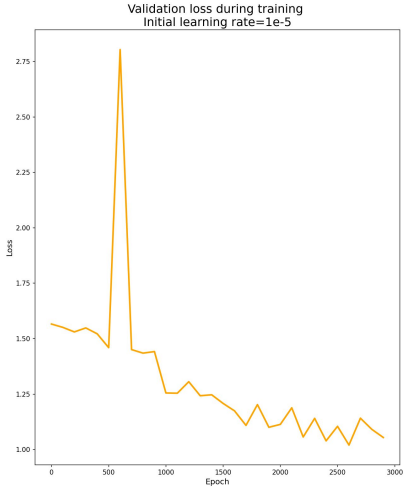
We have tried 3 different values of learning rate, and we trained always for 3000 epochs. Training subset consists of the first 25% frames of the sequence, and validation subset the last 75% frames.

Validation prediction output for trained model with lr=1e-3:









Task 1.3: Object Detection: Fine-tune (Team 4) [2/2]

Parameters for training:
Optimizer: SGD with momentum
Batch size: 128

Comparison between the trained models:

Model	Initial learning rate	Validation mAP	Training epochs	Training time (hh:mm:ss)
Faster-RCNN	1e-3	0.8865	3000	02:08:26
	1e-4	0.7545	3000	02:08:35
	1e-5	0.3168	3000	02:10:42

The best validation mAP is obtained using an initial learning rate of 1e-3. As we can see in the table with the validation mAP, and also in the plot of the previous slide, the learning rate of 1e-5 does not reduce the loss enough, so after training mAP is too low.

Comparison between the best trained model in terms of mAP and the pre-trained model:

We executed again the inference of the pre-trained model of Task 1.1, but now with just the validation subset (last 75% part of the video sequence), to compare it with the results of the best trained model. Of course, that model was trained with the first 25% frames, but validated with the last 75% frames.

Model	Fine-tuned	Initial learning rate	Validation mAP
Faster-RCNN	False	-	0.4043
	True	1e-3	0.8865

After fine-tuning the validation mAP rises from 0.40 to 0.89.

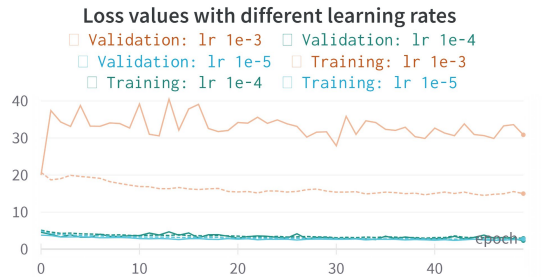
Task 1.3: Object Detection: Fine-tune (Team 5) [1/2]

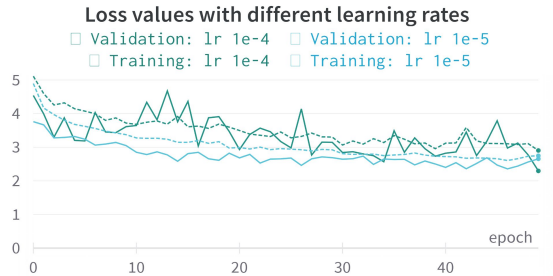
As shown in the slides for task 1.1, our best performing model was the DETR. As it was not too time-consuming, we attempted to fine-tune it using the video frames.

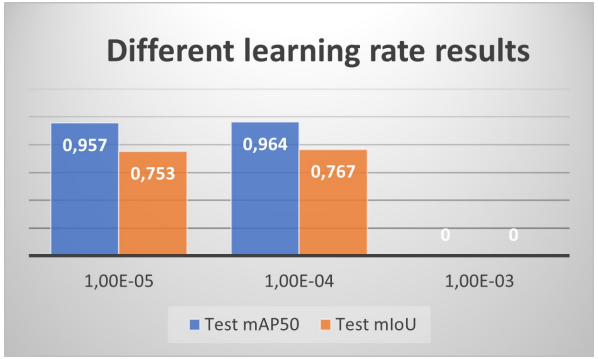
First 25% of frames were used for training and the remaining frames were used for testing.
Hyperparameters used:

- Data augmentation techniques were applied to prevent overfitting, which included:
  - Random horizontal flip
  - Random resize (with a list of scales ranging from 480 to 800)
  - Random crop
- AdawW optimizer was used.
- Model was trained for 50 epochs.
- Batch size of 8 was used.

- The official training script was used as a reference, and our videos were parsed into COCO format.
- A single RTX 3090 was used for training.
- Each training session lasted for 3 hours and 37 minutes.
- Starting point weights were taken from the COCO dataset.
- Multiple learning rates were tested during the training process.





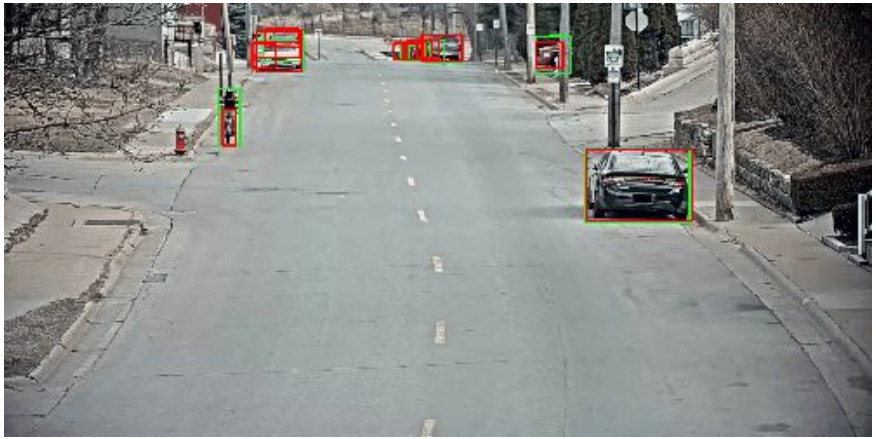


- The learning rate of 1e-3 was found to be too high, causing the loss to explode during the initial epochs.
- The learning rates of 1e-4 and 1e-5 had similar performance. During the initial epochs, the loss was reduced significantly and then stabilized.
- A learning rate of 1e-4 was selected for the next round of training.

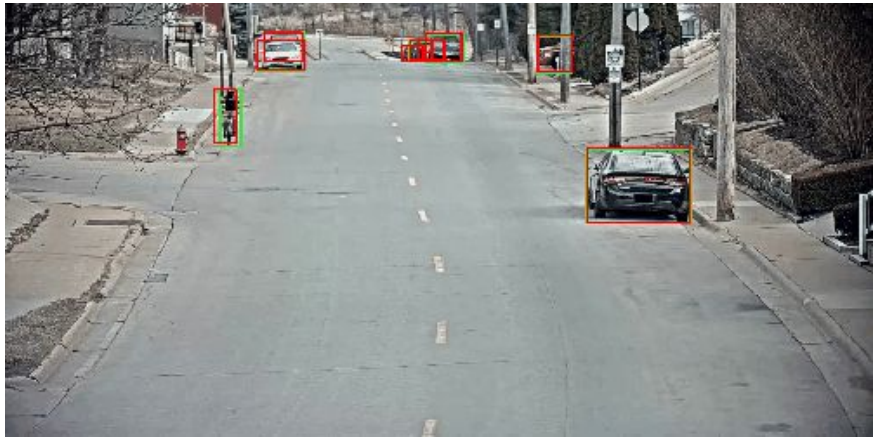
Task 1.3: Object Detection: Fine-tune (Team 5) [2/2]

Ground truth box

Detected box



Pretrained DETR



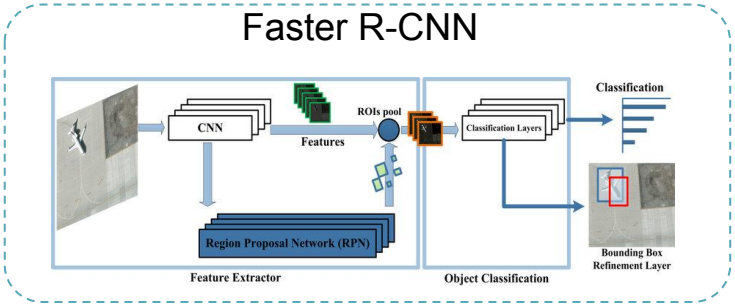
Fine-tuned DETR

The fine-tuned model:

- Generally produces more stable and precise detections compared to the pre-trained model.
- The bicycle detections now include the person, improving their IoU with GT.
- Vans and pick-ups are consistently detected as cars.
- Parked car detections are very stable, possibly due to the high level of overfitting on these static objects.

	Percentage of improvement comparing pretrained
mAP50	**+48.3%**
mIoU	**+47.5%**

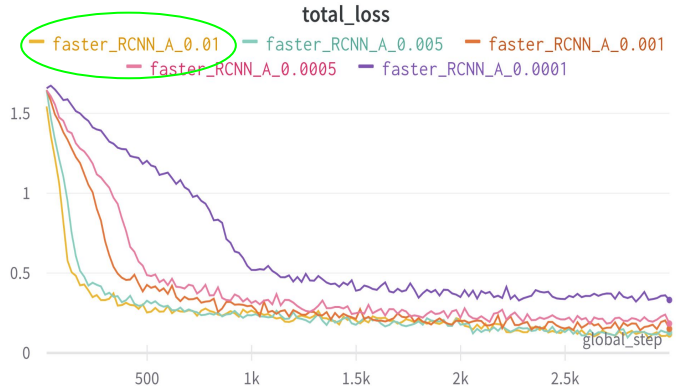
Task 1.3: Object Detection: Fine-tune (Team 6) [1/2]

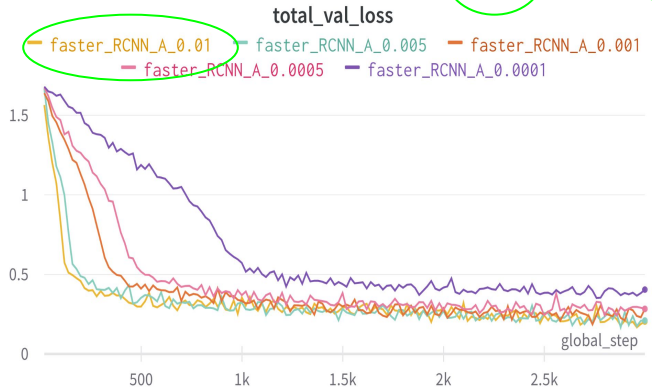


Config:

- INITIAL WEIGHTS → `faster_rcnn_X_101_32x8d_FPN_3x`
- NUM_STEPS (batch iter) → 3000
- BASE_LR → **grid search**
- LR_WARM_UP → 1000
- LR_SCHEDULER → (2000, 2500)
- GAMMA_SCHEDULER → 0.5
- IMS_PER_BATCH → 2

- We randomly select a subset of 10% samples from the total validation dataset to compute the validation loss during training.
- Total loss is the sum of the **loss regression box** and the **classification loss**
- We perform a grid search on the BASE_LR hyperparameter, evaluating values of **0.01**, 0.005, 0.001, 0.0005, 0.0001.



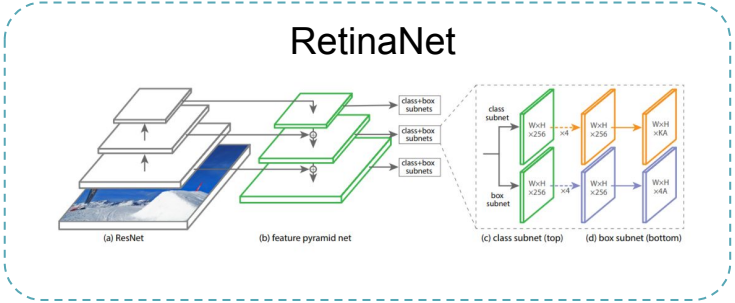


Best loss curves and higher AP50

Model	AP_{50}
Pretrained Faster R-CNN	0.48
Fine-tuned Faster R-CNN	**0.85**

https://www.researchgate.net/figure/Faster-RCNN-framework-for-object-detection-and-classification-new-network-model-to-form_fig5_346764538

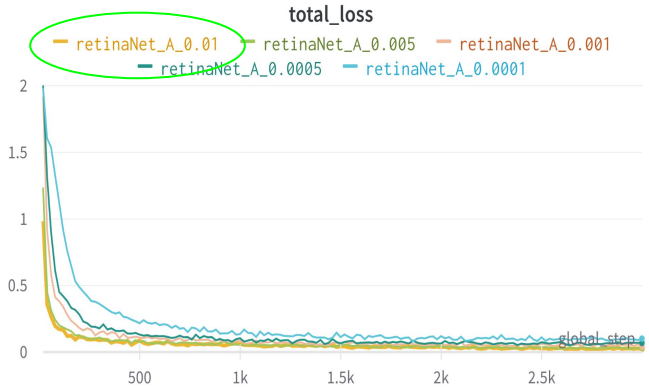
Task 1.3: Object Detection: Fine-tune (Team 6) [2/2]

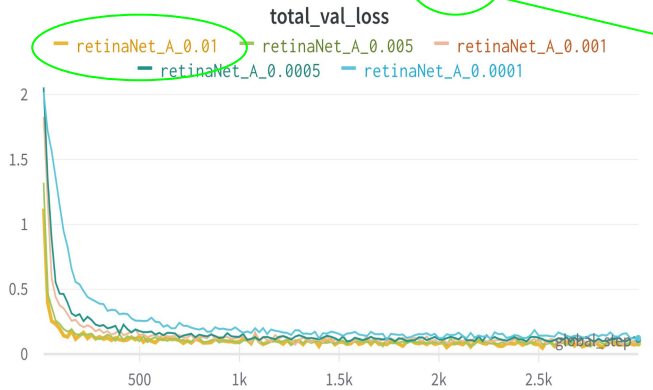


Config:

- INITIAL WEIGHTS → `retinanet_R_101_FPN_3x`
- NUM_STEPS (batch iter) → 3000
- BASE_LR → **grid search**
- LR_WARM_UP → 1000
- LR_SCHEDULER → (2000, 2500)
- GAMMA_SCHEDULER → 0.5
- IMS_PER_BATCH → 2

- We randomly select a subset of 10% samples from the total validation dataset to compute the validation loss during training.
- Total loss is the sum of the **loss regression box** and the **classification loss**
- We perform a grid search on the BASE_LR hyperparameter, evaluating values of **0.01**, 0.005, 0.001, 0.0005, 0.0001.





Best loss curves and higher AP50

Model	AP_{50}
Pre-trained RetinaNet	0.52
Fine-tuned RetinaNet	**0.92**

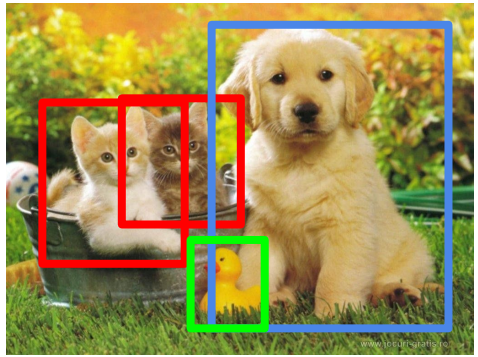
https://developers.arcgis.com/python/guide/how-retinanet-works/

Task 1.3: Feedback

	feedback
Team 1	ultralytics framework, more teams use it, why? check wording, it seems that you changed now to yolov8 but if we want to compare off-the-shelf vs fine-tunned same object detector needs to be used. Good explanation (e.g. augmentation). Missing comparison with off-the-shelf. Missing training curves. Conclusion: much better mAP.
Team 2	Training curves not provided. Not many details about the finetuning process Did you use data augmentation? Conclusions: possibility of overfitting with random sampling mAP similar to off-the-shelf
Team 3	Space is limited, try not to repeat (fine-tunning leverages the knowledge…) Missing comparison with off-the-shelf. Missing training curves. Conclusion: much better mAP (with reservation on overfitting). Could you test with less frames (25% at the end?)
Team 4	Validation loss curves provided but no training loss. Analysis of the hyperparameters used. Good! Much better mAP Data augmentation?
Team 5	Bonus point for comparison with off-the-shelf and training curves. Nice discussion of qualitative results. Conclusion: much better mAP with reservations (parked cars)
Team 6	Details about the training process given. Training/validation loss curves. Data augmentation? Grid search for the LR. Good Much better mAP

Tasks

- Task 1: Object detection
  - Task 1.1: Off-the-shelf
  - Task 1.2: Annotation
  - Task 1.3: Fine-tune to your data
  - **Task 1.4: K-Fold Cross-validation**
- Task 2: Object tracking



Task 1.4: K-Fold Cross-validation

Try different data partitions on your sequence:

Strategy A (same as week 2):

- First 25% frames for training
- Second 75% for test.

Strategy B:

- K-Fold cross-validation (use K=4).
- Fixed folds. Split sequence into 4 folds (25% each). One fold is the same as strategy A

Strategy C:

- K-Fold cross-validation (use K=4)
- Random 25% Train - rest for Test

Task 1.4: K-Fold Cross-validation (Team X)

- Copy & paste (max 2, pages)

Task 1.4: K-Fold Cross-validation (Team 1) - [1/1]

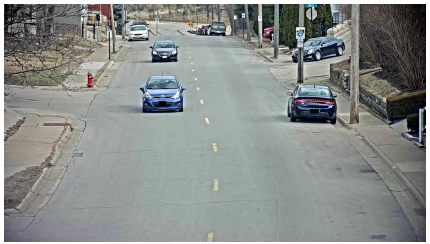
For this task we will use the same model we fine-tuned in Task 1.3, YOLOv8.

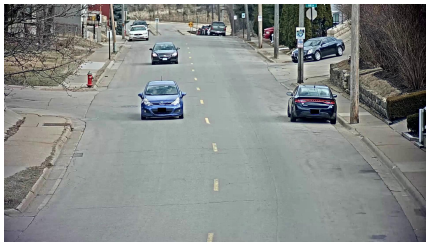
Three different strategies have been followed:

- Strategy A: the strategy used in the previous task, first 25% frames for training and the rest for validation
- Strategy B: 4-fold cross validation. Fixed folds.
- Strategy C: 4-fold cross validation. Random folds.

For both strategy B and C the KFold method from sklearn has been used. For strategy C it was necessary to set the parameter shuffle to True, which shuffles data before building the different folds in order to have random folds that **do not overlap**.

Strategy C results are considerably better than Strategy B and A. That is due to using random folds with a dataset that consists of a sequence in time, which is **not a very good idea** as it will overfit the validation set, as the validation set and the training will have very similar frames.





frame_0524.jpg in validation set and frame_0527.jpg in training set in the first random fold. Due to random shuffling, both sets have very similar frames and it overfits the validation set.

Model	Strategy	validation mAP50 (all) for K fold				Mean mAP50
		0	1	2	3	
YOLOv8	A	0.939	-	-	-	0.939
	B	0.936	0.968	0.964	0.968	0.959
	C	0.977	0.982	0.973	0.976	**0.977**

Fine-tuning with K-Fold Cross-validation. Using SGD optimizer with momentum=0.937 and initial learning rate of 0.01.

Task 1.4: K-Fold Cross-validation (Team 2) [1/1]

Model	Fold*	Confidence threshold	$AP_{0.50}$	$AP_{0.75}$	$AP_{0.90}$
Faster R-CNN	XOX	.50	0.67	0.59	0.58
		.75	0.67	0.59	0.58
		.90	0.67	0.59	0.58
Faster R-CNN	XXO	.50	0.81	0.80	**0.65**
		.75	0.81	0.80	0.65
		.90	0.81	0.80	0.65
Faster R-CNN	OXX	.50	0.55	0.55	0.44
		.75	0.55	0.55	0.44
		.90	0.55	0.55	0.44
distribution	-	-	0.67+-0.13	0.68+-0.10	0.55 +-0.10

* fold:
  - **x** - test (33%)
  - **o** - train (33%)

Some important aspect to consider under this setup is the surprising fact that the maximum AP is achieved by using the last 33% fold for training.

We decided to address the folds in a sequential way as the random approach creates the maximum overfitting possible.

As there should not be prevalence for any fold in the experimental setup, we selected the mean and variance for all folds as the final results.

Task 1.4: K-Fold Cross-validation (Team 3) [1/2]

Strategy A:

In this strategy, we split the dataset into a fixed train-test set, with the first 25% of the data for training and the remaining 75% for testing.

1. Split the dataset into train (first 25%) and test (remaining 75%) sets.
2. Train the YOLO model on the training set.
3. Evaluate the model on the test set.
4. Save the model.

Strategy B:

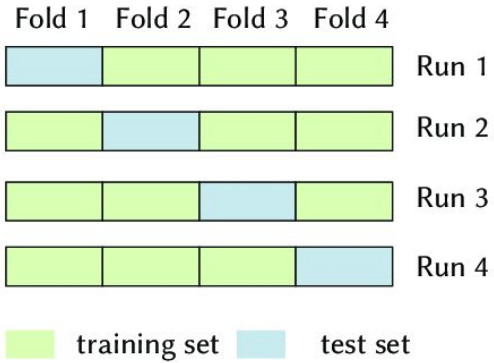
In this strategy, we use K-Fold cross-validation with K=4 and fixed folds. The dataset is divided into 4 equal-sized folds (25% each). One of the folds corresponds to the same fold used in Strategy A.

1. Divide the dataset into 4 equal-sized folds.
2. Perform 4 iterations, where each iteration: a. Uses one fold as the validation set and the remaining 3 folds as the training set. b. Trains the YOLO model on the training set. c. Evaluates the model on the validation set. d. Saves the model for the specific fold.
3. Calculate the average performance across all 4 folds

Strategy C:

In this strategy, we use K-Fold cross-validation with K=4 and random 25% train-test splits.
In each iteration, we randomly select 25% of the dataset for training and the rest for testing.

1. Perform 4 iterations, where each iteration:
   a. Randomly selects 25% of the dataset for training and the rest for testing.
   b. Trains the YOLO model on the training set.
   c. Evaluates the model on the test set.
   d. Saves the model for the specific iteration.
2. Calculate the average performance across all 4 iterations.



Task 1.4: K-Fold Cross-validation (Team 3) [2/2]

We can conclude:

- **Strategy A**, with a fixed train-test split, may have resulted in either overfitting or underfitting, depending on the distribution of the data within the split. In this case it overfitted because we used frames of the same video randomly which the next is nearly the same as the before.
- **Strategy B**, with K-Fold cross-validation and fixed folds, will provide a more robust evaluation of the model's performance by averaging the results across different folds. This will help to reduce the impact of overfitting and provided a better estimate of the model's ability to generalize.
- **Strategy C,** with K-Fold cross-validation and random train-test splits, offered the most flexibility and adaptability to different data distributions. By using random splits, this strategy further reduced the impact of overfitting and provided an even better estimate of the model's performance on unseen data.

BUT! Using the same video with different consecutive frames, there is a high probability that the frames used for the train are also used for the test, so we have an overfitted model.

Model	Strategy	validation mAP95 (all) for K fold				Mean mAP95
		0	1	2	3	
YOLOv8 With standard HP from Ultralytics	A	0'743	-	-	-	0'743
	B	0.765	0'789	0'756	0'739	0'762
	C	0'759	0'739	0'799	0'790	**0'771**

Task 1.4: K-Fold Cross-validation (Team 4) [1/1]

The parameters for batch size and optimizer were the same as for Task 1.3. The training epochs were 3000

			Validation mAP (0.5) for each k				
Model	**Strategy**	**Initial learning rate**	**1**	**2**	**3**	**4**	**Mean AP (0.5)**
Faster-RCNN	A	1e-3	0.8189	-	-	-	0.8189
	B	1e-3	0.8189	0.8363	0.7996	0.7845	0.8091
		1e-4	0.7157	0.7166	0.6645	0.6326	0.6823
		1e-5	0.1724	0.1740	0.1724	0.1691	0.1718
	C	1e-3	0.8501	0.8303	0.8435	0.8402	0.8410
		1e-4	0.7066	0.7079	0.6858	0.6804	0.6951
		1e-5	0.1703	0.1717	0.1721	0.1725	0.1761

We perform cross-validation with strategies A (no cross-validation), B (cross-validation with k=4) and C (cross-validation randomizing dataset and k=4). We notice that the strategy C is the one that gives best results (in our runs for cross-validation). We think that this happens because of the distribution of the frames, with randomized frames we may allow the model to have access for training into far more objects than just those that it could see in contiguous frames. However, having access to almost all the objects of the dataset for training may involve overfitting.

Task 1.4: K-Fold Cross-validation (Team 5) [1/2]

We tried to do cross validation with the dataset splits using the sklearn implementation (shuffle True for random).

Strategy B:

Fold splits were made sequentially, which may lead to overfitting of the model as the train and test frames are similar:

	FOLD 1	FOLD 2	FOLD 3	FOLD 4	**MEAN**
Test mAP50 (11 point implementation)*	0,906	0.907	0.907	0.908	**0.907**
Test mAP50	0,965	0,961	0,944	0,955	**0.956**
Test mIoU	0,767	0,786	0,758	0,733	**0.761**

The results are quite similar with every fold.

*The 11-point implementation of mAP calculation does not provide precise values for mAP when it is greater than 0.91. This is because it does not include the last point when recall is not equal to 100%. In the next experiments, we will use the common implementation of mAP calculation instead of the 11-point method.

Task 1.4: K-Fold Cross-validation (Team 5) [1/2]

Strategy C:

Fold splits were made randomly:

	FOLD 1	FOLD 2	FOLD 3	FOLD 4	**MEAN**
Test mAP50	0,976	0,982	0,982	0,978	**0.98**
Test mIoU	0,78	0,871	0,863	0,797	**0.828**

The results are even better using random splits. This may be due to the fact that the difference between temporally close frames is small, and random splitting ensures that the training frames are evenly spread, thus reducing the risk of overfitting to a specific part of the sequence. However, this may also lead to high overfitting of the entire sequence during training, which could be considered as cheating despite better test results.

In our opinion, the results of this experiment may not be entirely fair. We believe that a different sequence should be used for testing to assess the network's robustness, as using similar train and test frames could lead to overfitting.

	Mean mAP	Mean mIoU
Strategy A	0.965	0.767
Strategy B	0.956	0.761
Strategy C	**0.98**	**0.828**

Task 1.4: K-Fold Cross-validation (Team 6) [1/2]

		AP_{50}	
	# Split	Faster R-CNN	RetinaNet
Strategy A	1	0.85	0.92
Strategy B	2	0.84	0.84
	3	0.86	0.88
	4	0.87	0.88
Strategy C	1	0.87	0.97
	2	0.88	0.88
	3	0.88	0.96
	4	0.88	0.88
Mean		0.87	0.90

There are 2141 frames in total. We split them in 4 parts and perform 4-fold cross validation with each fold containing approximately 540 frames.

- In strategy A we keep the first 25% frames to train (first part) and the rest 75% to validate. In strategy B2, B3 and B4 we take the second, third and fourth part,respectively, to train. 3
- In strategy C, we take randomly 4 splits which train and test distribution is also 25% and 75% but now without keeping any sequential order. In this case, we use seed parameters for reproducibility.

All models were trained with the same hyperparameters, stated on the previous slides.

RetinaNet is a more complex model than Faster R-CNN due to the addition of a feature pyramid network and a novel focal loss function. This added complexity can make RetinaNet more sensitive to changes in the data and harder to train consistently. In contrast, Faster R-CNN is a simpler model with a more straightforward loss function, which may lead to more stable training.

Task 1.4: K-Fold Cross-validation (Team 6) [2/2]

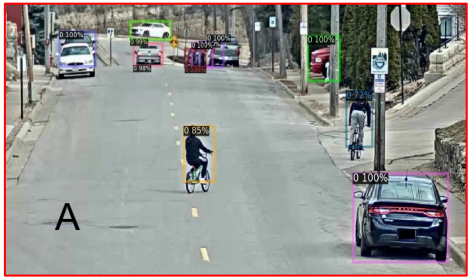
In both strategy A and B, the model is trained on a single ordered split of 25% of the frames and validated on the remaining 75%. However, this approach may lead to overfitting because the model might only learn patterns specific to that particular subset and fail to generalize well to other parts of the dataset.

On the other hand, strategy C involves randomly selecting four splits, each with 25% frames for training and 75% for validation. This approach may result in better generalization because the model is trained on different parts of the dataset. Nevertheless, since the data selection is random, we can obtain more variations in the model's performance across different splits.

For example when detecting the objects in frame 954, in strategy A, B3 and B4, when fitting the model, the model misclassify bicycles as cars when testing in the second 25% of the video, since it has not been exposed to bicycles crossing the road during training. Nor has it seen bicycles with their backs turned.

In strategy A, the model has been trained with only one bike coming towards the camera and the trajectory is always straight the street, away from the middle of the road. It seems for that reason the confidence value is lower than the other splits, that barely has not seen any bicycle.

Otherwise, in strategy C1, the model has been trained with frame 953, so it has learned to avoid classifying the bicycle as a car.



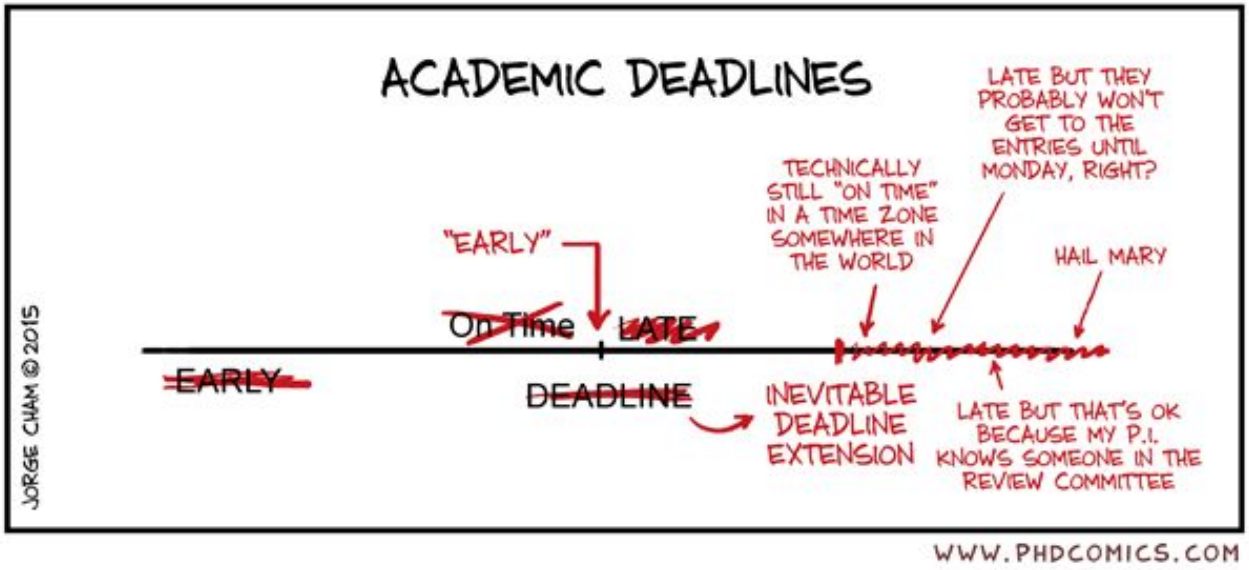
Task 1: Object Detection

Team ID	Model (choose one)	T1,1 Off-the-shelf (mAP_{50}) (Random 4-Fold cross-validation)	T1,2 Fine-tuned (mAP_{50}) (Random 4-Fold cross-validation)
Team 1	YOLOv8	0.5422	0.977
Team 2	Faster-RCNN	0.51	0.67+-0.13
Team 3	YOLOv8	0.521	0.901
Team 4	Faster-RCNN	0.278	0.841
Team 5	DETR	0.647	0.98
Team 6	RetinaNet R101	0.52	0.92

Task 1.4: Feedback

	feedback
Team 1	Implemented all strategies Explains why random sampling for the 25% train (Strategy C) provides the best results and why it is not a good idea in our scenario. Both quantitative and qualitative evaluation is provided.
Team 2	Implemented all strategies Both quantitative and qualitative evaluation is provided. Explains why random sampling for the 25% train (Strategy C) provides the best results and why it is not a good idea in our scenario.
Team 3	Implemented all strategies implementation? Explains why random sampling for the 25% train (Strategy C) provides the best results and why it is not a good idea in our scenario. Both quantitative and qualitative evaluation is provided.
Team 4	Implemented all strategies Both quantitative and qualitative evaluation is provided. Analysis of strategy C (more objects analyzed) not related to spatial redundancy. Overfitting mentioned, however.
Team 5	Implemented all strategies Explains why random sampling for the 25% train (Strategy C) provides the best results and why it is not a good idea in our scenario. Both quantitative and qualitative evaluation is provided.
Team 6	Results provided for two models. Implemented all strategies Both quantitative and qualitative evaluation is provided. Analysis of strategy C not correct. No more generalization.

Deliverables



- Deadline: **Wednesday March 29th at 3pm**
- Deliverables:
  - Submit your report by editing these slides: task1 and task2
  - Provide feedback regarding the teamwork (email)