

From Cones Detection to Road Segmentation

Dott. Federico Putamorsi
Dott. Leonardo Zanella

Goal

The aim of this project is to identify **cones** and **lines** of the road from images of the **SAE driverless** formula. Followed by the identification of the **drivable** road by segmentation.



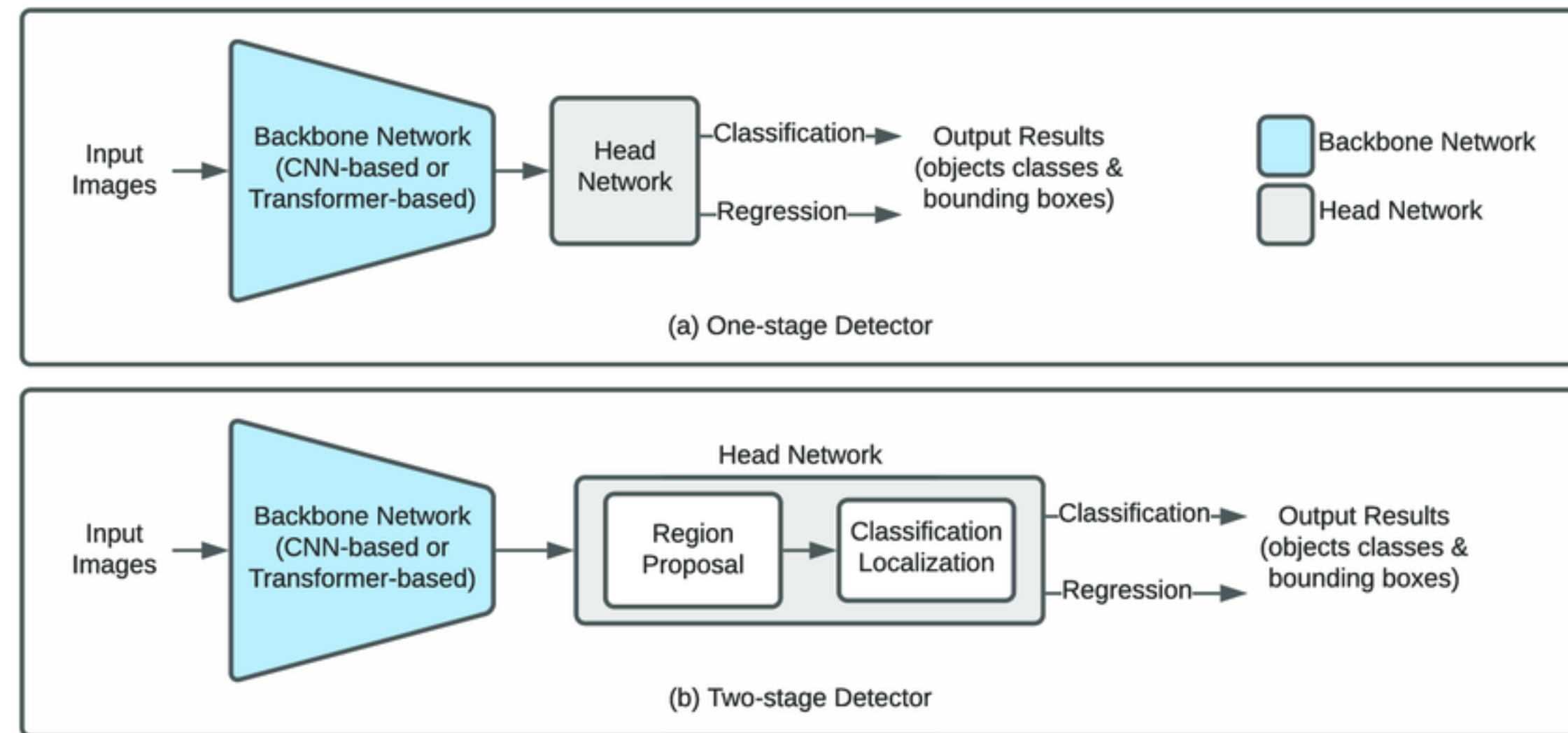
SOTA: State Of The Art

State-of-the-art object detection methods can be categorized into **two** main types:

- **Two-stage object detectors**: this architecture involves object region **proposal** with conventional Computer Vision methods or deep networks, followed by **object classification** based on features extracted from the proposed region with bounding-box regression. This methods achieve the **highest detection accuracy** but are typically **slower**. Because of the **many inference** steps per image, the performance is not as good as one-stage detectors.

SOTA: State Of The Art

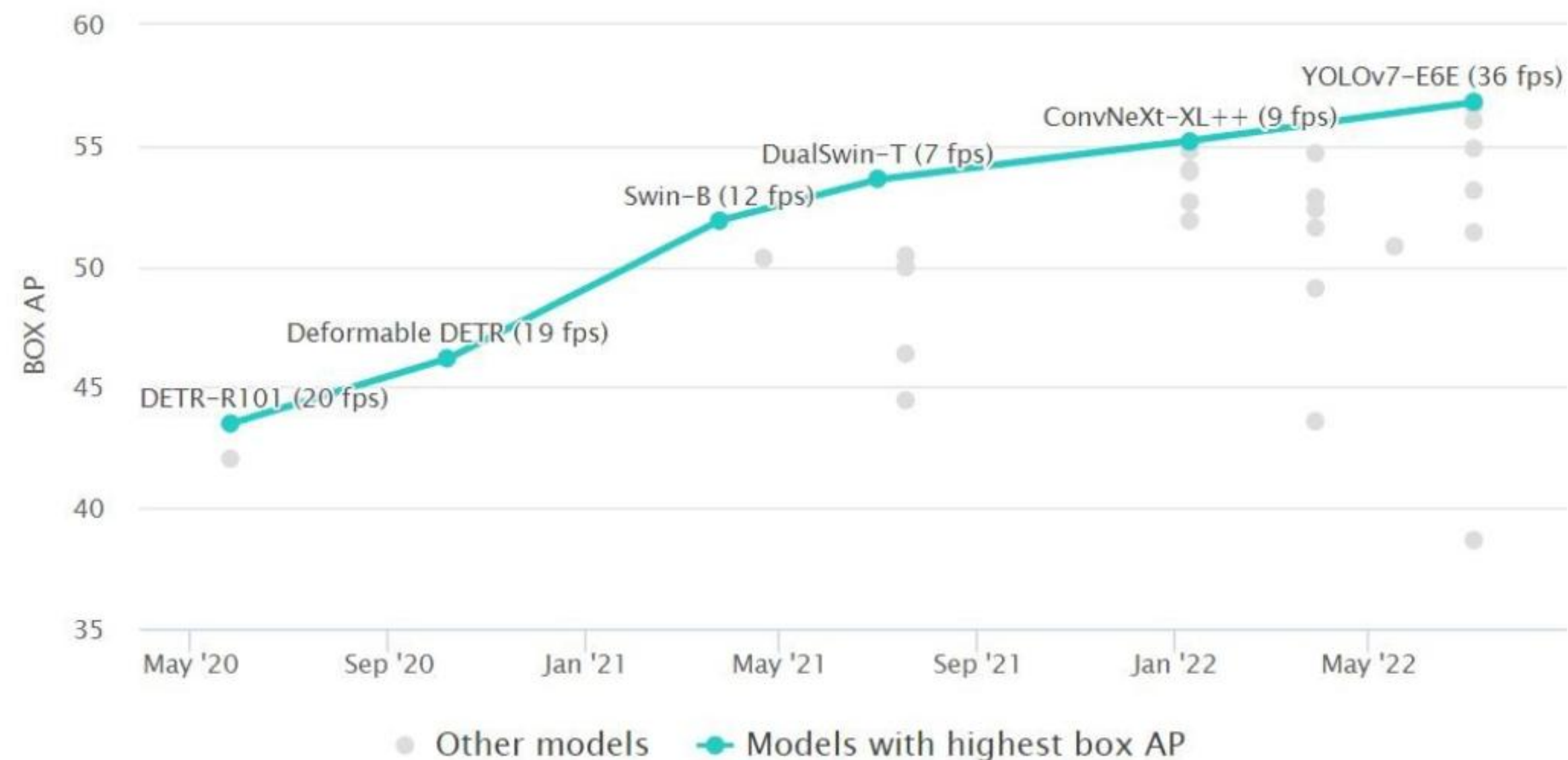
- **One-stage object detectors**: predict bounding boxes over the images **without** the region **proposal** step. This process consumes less time and can therefore be used in **real-time** applications. These detectors prioritize inference speed and are super fast but not as good at recognizing **irregularly** shaped objects or a group of small objects.



SOTA: State Of The Art

The most popular benchmark is the Microsoft **COCO dataset**. Different models are typically evaluated according to a **Mean Average Precision** (mAP) metric.

The best real-time object detection algorithm is **YOLOv7**, followed by Vision Transformer (**ViT**) such as Swin and DualSwin, PP-YOLOE, YOLOR, YOLOv4, and EfficientDet.



SOTA: State Of The Art

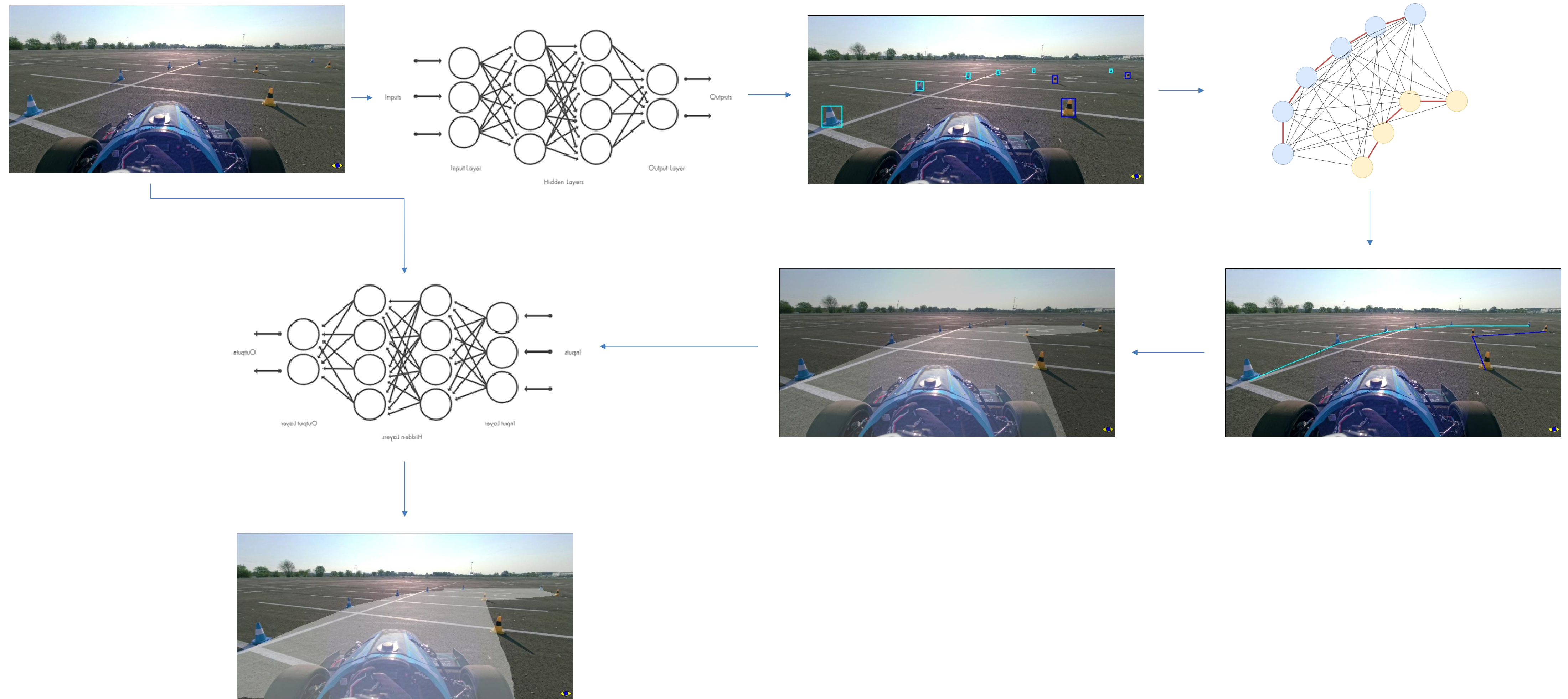
Also, on the MS COCO dataset, an important benchmark metric is **inference time (ms/Frame, lower is better)** or **Frames per Second (FPS, higher is better)**



SOTA: State Of The Art

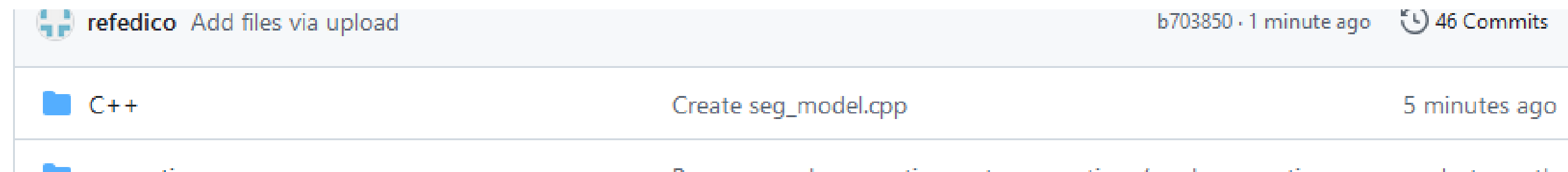
Name	Year	Type	Dataset	mAP	Inference rate (fps)
R-CNN	2014	2D	Pascal VOC	66%	0.02
Fast R-CNN	2015		Pascal VOC	68.80%	0.5
Faster R-CNN	2016		COCO	78.90%	7
YOLOv1	2016		Pascal VOC	63.40%	45
YOLOv2	2016		Pascal VOC	78.60%	67
SSD	2016		Pascal VOC	74.30%	59
RetinaNet	2018		COCO	61.10%	90
YOLOv3	2018		COCO	44.30%	95.2
YOLOv4	2020		COCO	65.70%	62
YOLOv5	2021		COCO	56.40%	140
YOLOR	2021		COCO	74.30%	30
YOLOX	2021		COCO	51.20%	57.8
Complex-YOLO	2018	3D	KITTI	64.00%	50.4
Complexer-YOLO	2019		KITTI	49.44%	100
Wen et al.	2021		KITTI	73.76%	17.8
RAANet	2021		NuScenes	62.00%	

An Overview










Explore our code

- \approx **1000 lines** of Python code (LOC), \approx **1500 LOC** in **C++**
- We also have **generated** C++ code from an **AI code translator**
- Transformer based model, finetuned from **GPT-3.5**
- We haven't tried the code but it **definitely doesn't work**



Explore our code

 conversions	Rename mask_conversion.py to conversions/mask_convertio...	1 hour ago
 documentation	Update doc.html	2 minutes ago
 models	Create yolo.pt	1 hour ago
 README.md	Create README.md	1 minute ago
 configmodel.yaml	Create configmodel.yaml	last week
 detector.py	Update detector.py	1 hour ago
 seg_model.py	Update seg_model.py	1 hour ago

Explore our code

API Documentation

YELLOW_INDEX

BLUE_INDEX

CLASS_NUM

generate_binary_mask()

order_img()

max_box()

lowest_point()

doesIntersect()

getAngle()

list_graphs()

graph_generator()

iou()

inference()

seg_inference()

main()

built with 

```
def generate_binary_mask(img, yellow_points, blue_points, mask_path):
```

[► View Source](#)

Generates a binary mask based on the provided yellow and blue points.

Parameters: `img` (PIL.Image.Image): The input image. `yellow_points` (list): List of tuples representing the yellow points. `blue_points` (list): List of tuples representing the blue points. `mask_path` (str): Path to save the generated binary mask.

Returns: `numpy.ndarray`: The generated binary mask.

```
def order_img(img_dir):
```

[► View Source](#)

Orders the images in the specified directory by their names.

Args: `img_dir` (str): The directory containing the images to be ordered.

Returns: `list`: A list of image names sorted in ascending order.

```
def max_box(boxes):
```

[► View Source](#)

Find the index of the box with the maximum height.

Args: `boxes` (list of list of int): A list of boxes, where each box is represented by a list of four integers `[x1, y1, x2, y2]` representing the coordinates of the box.

Returns: `int`: The index of the box with the maximum height.

```
def lowest_point(points):
```

[► View Source](#)

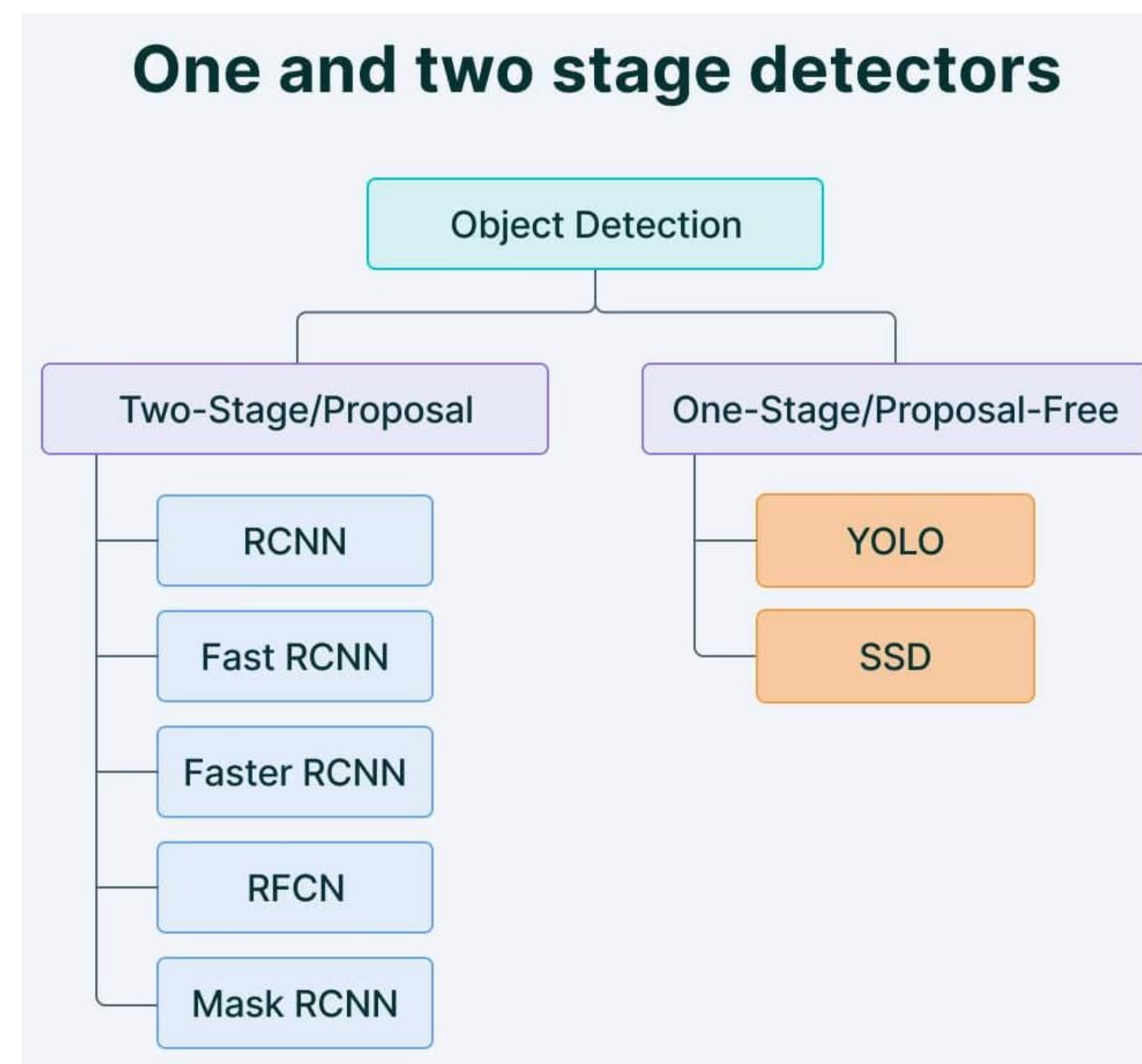
Finds the index of the point with the lowest y-coordinate from a list of points.

Args: `points` (list of tuple): A list of tuples where each tuple represents a point `(x, y)`.

YOLO: You Only Look Once

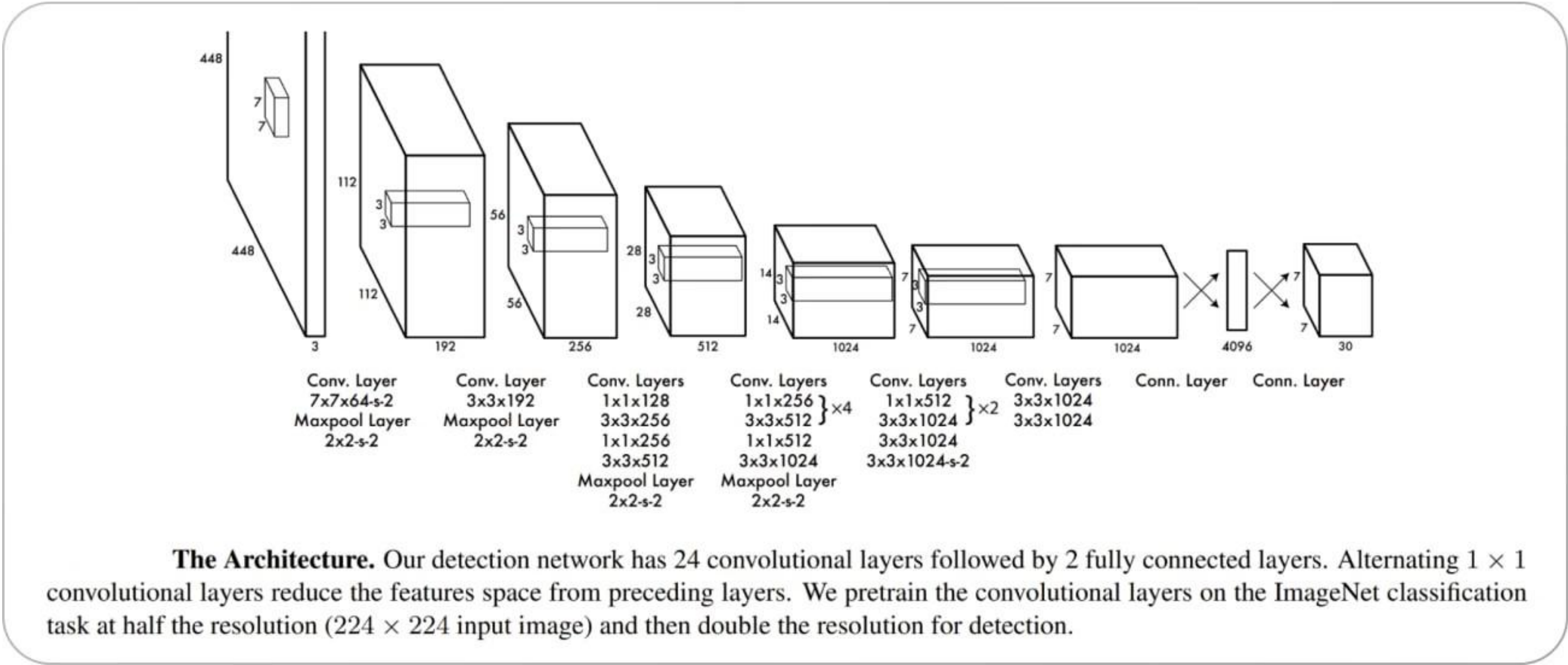


You Only Look Once (YOLO) proposes using an **end-to-end** deep convolutional neural network (CNN) that makes predictions of bounding boxes and class probabilities all at **once**.



YOLO: You Only Look Once

The **architecture** of the CNN model that forms the backbone of YOLO is shown below.



YOLO: You Only Look Once

- **Pre-trained Backbone:**

The first 20 convolution layers of the CNN are pre-trained on **ImageNet**. Additional convolution and fully connected layers are added to convert the network for object detection tasks.

- **Image Grid Division:**

The input image is divided into an $S \times S$ **grid**. Each grid cell detects objects whose centers fall within the cell.

- **Bounding Box Prediction:**

Each grid cell predicts B **bounding boxes** and **confidence scores**, which indicate how likely the box contains an object and the accuracy of the box's dimensions.

YOLO: You Only Look Once

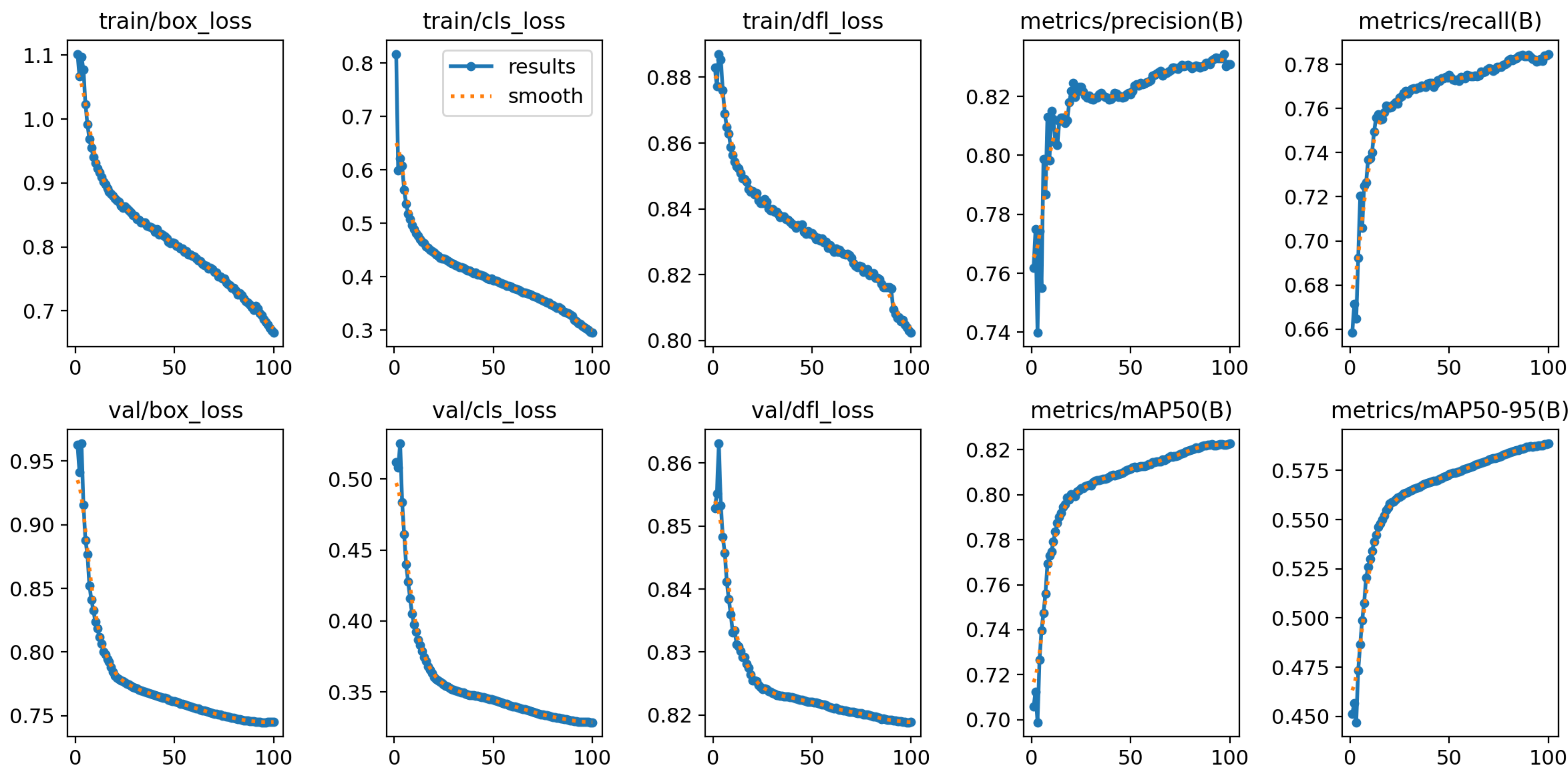
- **IOU:**

Most of the time, a single object in an image can have **multiple** grid box candidates for prediction. The goal of the IOU ($0 < IOU < 1$) is to **discard** such grid boxes to only keep those that are relevant defining a **threshold**.

- **Non-Maximum Suppression (NMS):**

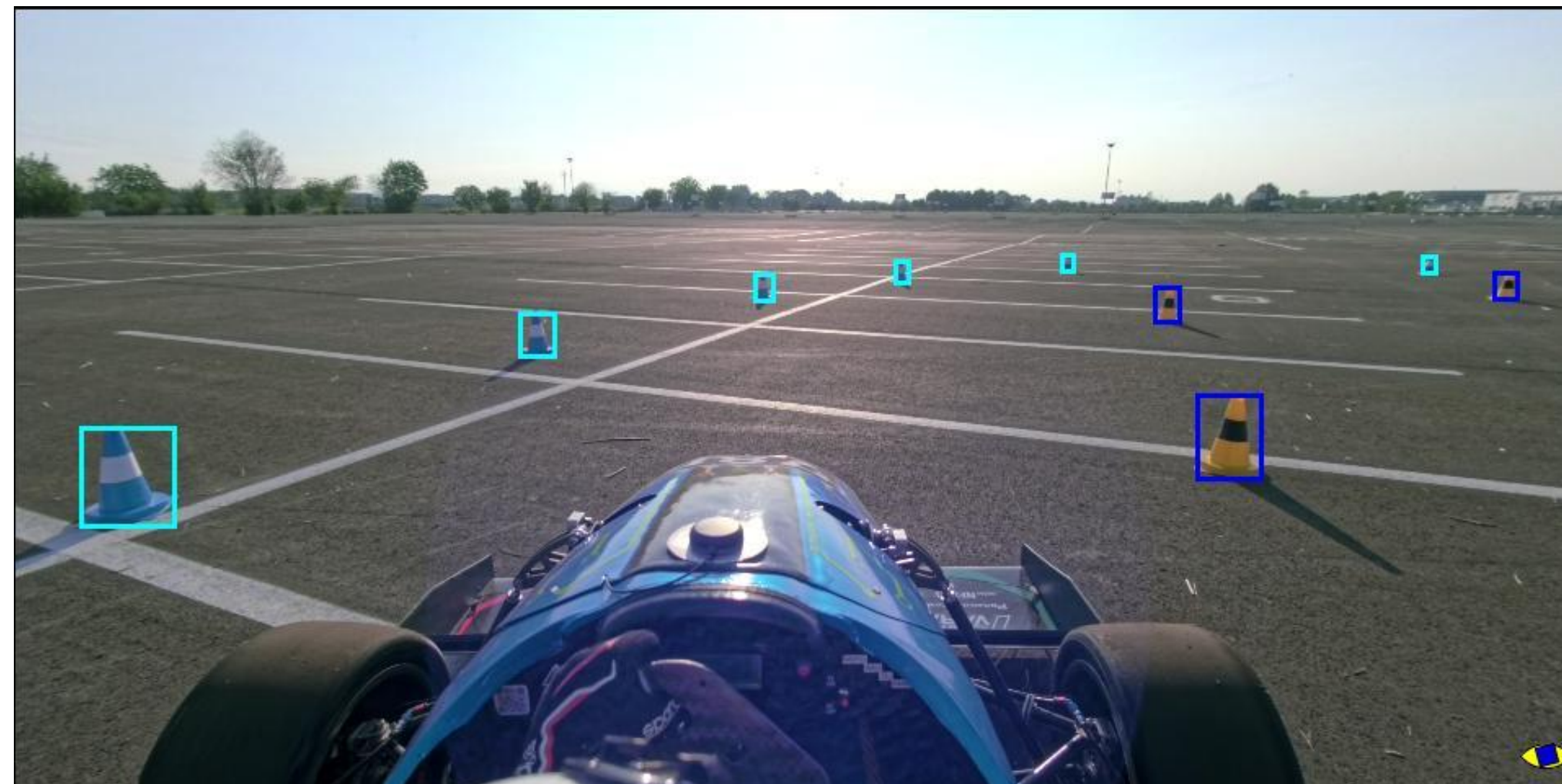
NMS is applied as a post-processing step to **eliminate redundant or overlapping bounding boxes**. This ensures only one bounding box is retained for each detected object.

YOLO11_m: Our Model Training



Our Deterministic Algorithm

- We **removed** small bounding boxes from the list of boxes found to avoid drawing lines too far away, where details are not as clear and is easier to **find a wrong path**.
- For each bounding box, we defined the **center** on it as $\left(0, \frac{w}{2}\right)$.

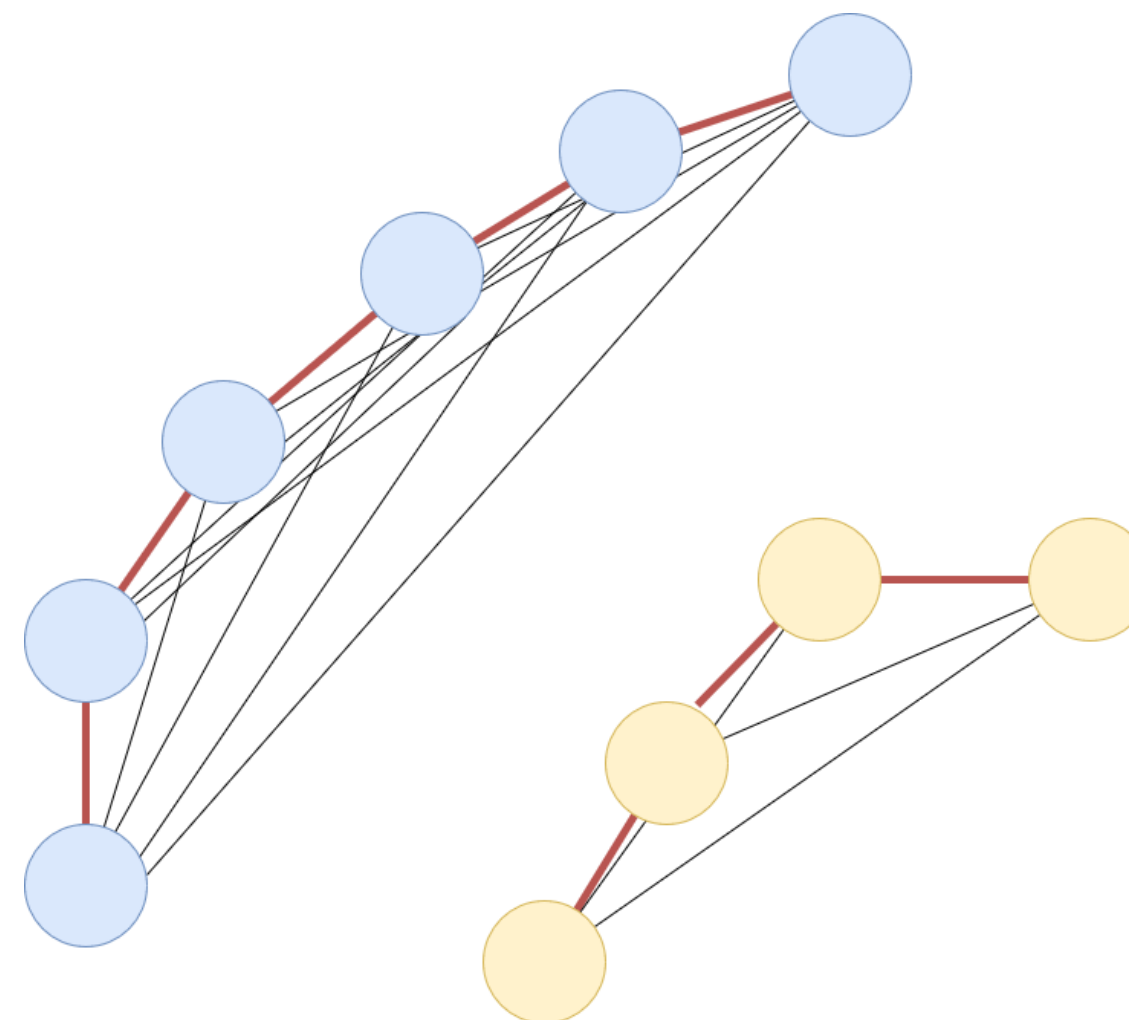


Our Deterministic Algorithm

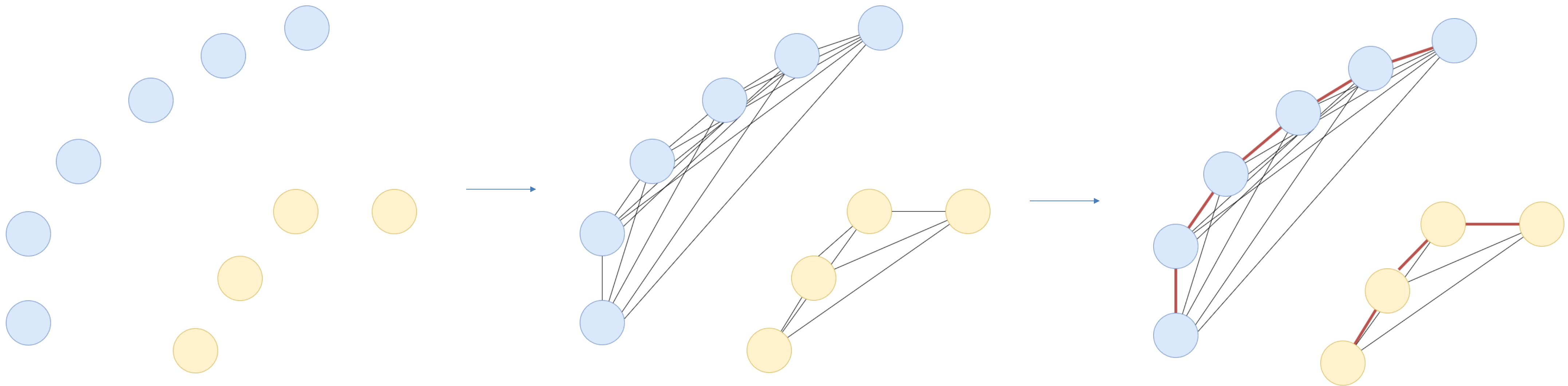
- Then we calculated the **distance** between each cone of the same color using the following empirical **formula**:

$$d(x_1, x_2, y_1, y_2, h_1, h_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^3} * \left(\frac{1}{h_1} + \frac{1}{h_2} \right)$$

- We created a **graph** where nodes are cones and links are distances.



Our Deterministic Algorithm



Our Deterministic Algorithm

- The algorithm **starts** from the cone with the **bigger height**, and sequentially connects all the cones of the same color with the **nearest** unconnected cone
- We introduced some **stop rules** to avoid clear errors in connecting cones:
 - If the distance between two cones is more than **3 times bigger** than the previous distance, we stop
 - If the angle between the new segment and the previous is bigger than **160 degrees**, we stop
 - If the lines from two different colors **intersect**, we stop

Mask Generation Algorithm

To **generate** the mask:

- We connected the **two last cones** together
- We extended the first segment of each color downward



Mask Generation Algorithm

When **only one cone type** was present in the image, we connected by default the last node to the point in the **bottom right or left opposed** to it.



Semantic Segmentation

Semantic segmentation is a crucial task in computer vision of **assigning** a **class** label to every single pixel of an input image.



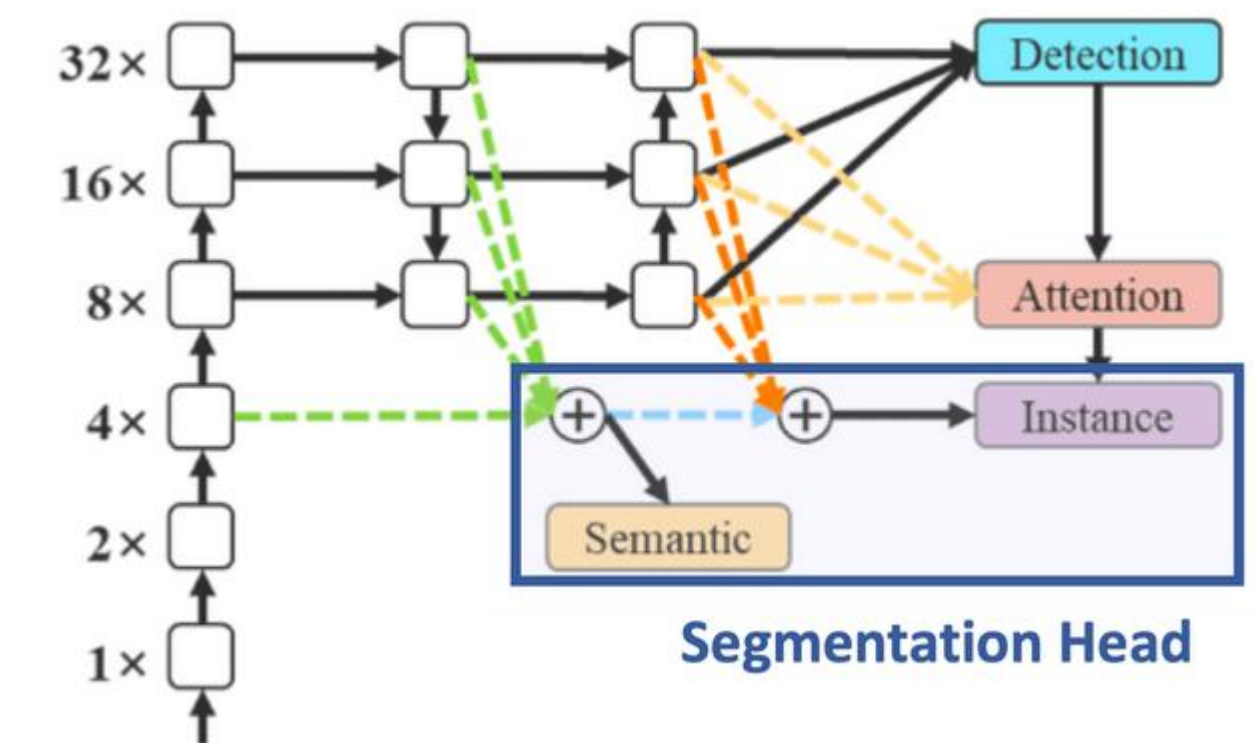
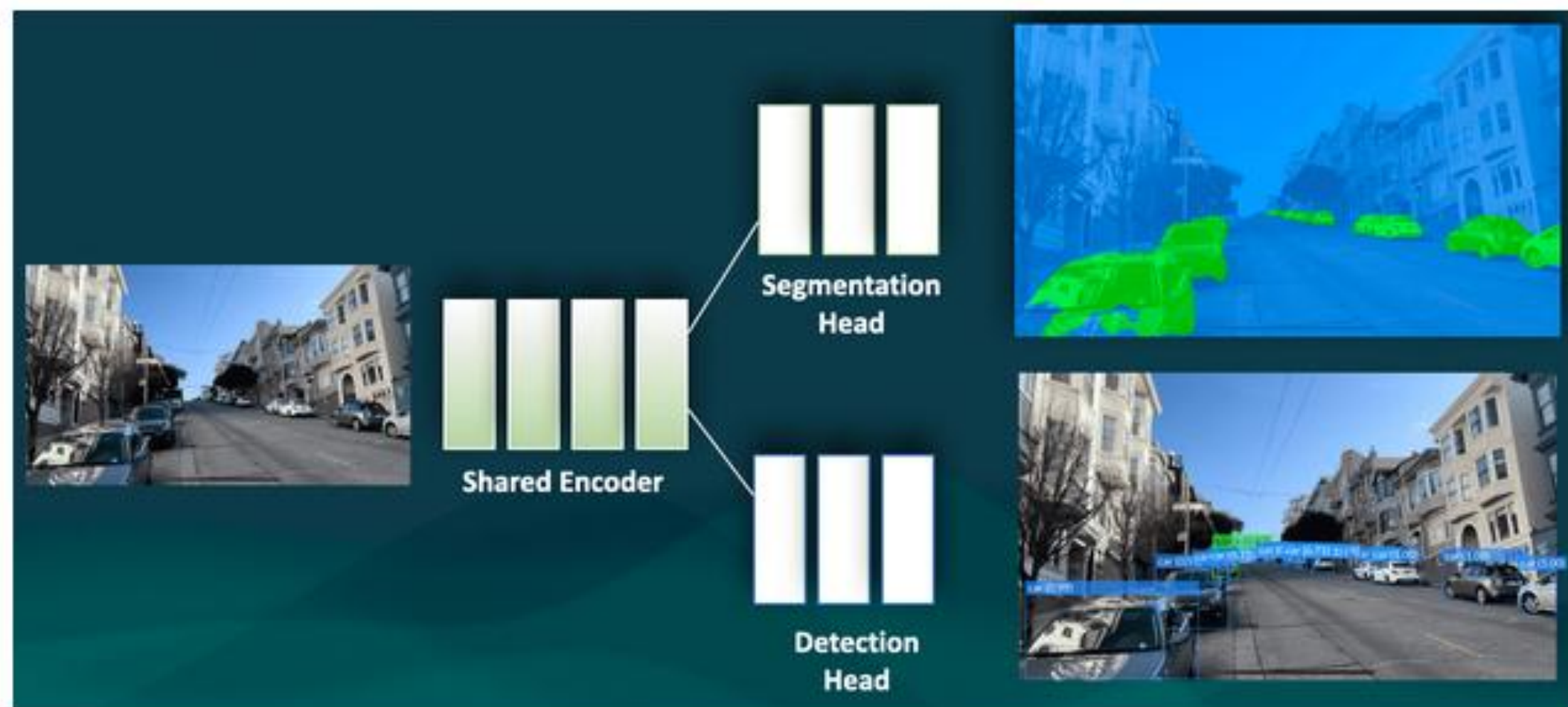
YOLO Segmentation

Steps for Segmentation using YOLO-Based Approach:

1. Detect objects in the image using **YOLO** to develop bounding boxes around each detected object.
2. For every bounding box, use **a segmentation algorithm** to classify each pixel within the box.
3. Semantic segmentation can be applied directly, while for **instance segmentation**, additional steps will be needed to differentiate instances.

YOLO Segmentation

In this case we will have a **segmentation head** so we will have the usual **backbone** with a different head:



SOTA: State Of The Art for Segmentation

To provide a thorough evaluation, we categorized the models based on specific use cases and assessed them using several key metrics:

- **Real-Time** Segmentation, these models are optimized for high-speed segmentation tasks that require quick processing.
- **High-Precision** Segmentation, these models focus on achieving the highest possible accuracy in segmentation tasks.
- **Promptable** Segmentation, these models are designed for interactive segmentation, allowing users to provide initial guidance through graphical prompts or text descriptions.

SOTA: State Of The Art for Segmentation

From the paper **Evaluating the Evolution of YOLO (You Only Look Once) Models: A Comprehensive Benchmark Study of YOLO11 and Its Predecessors** of 31 Oct 2024.

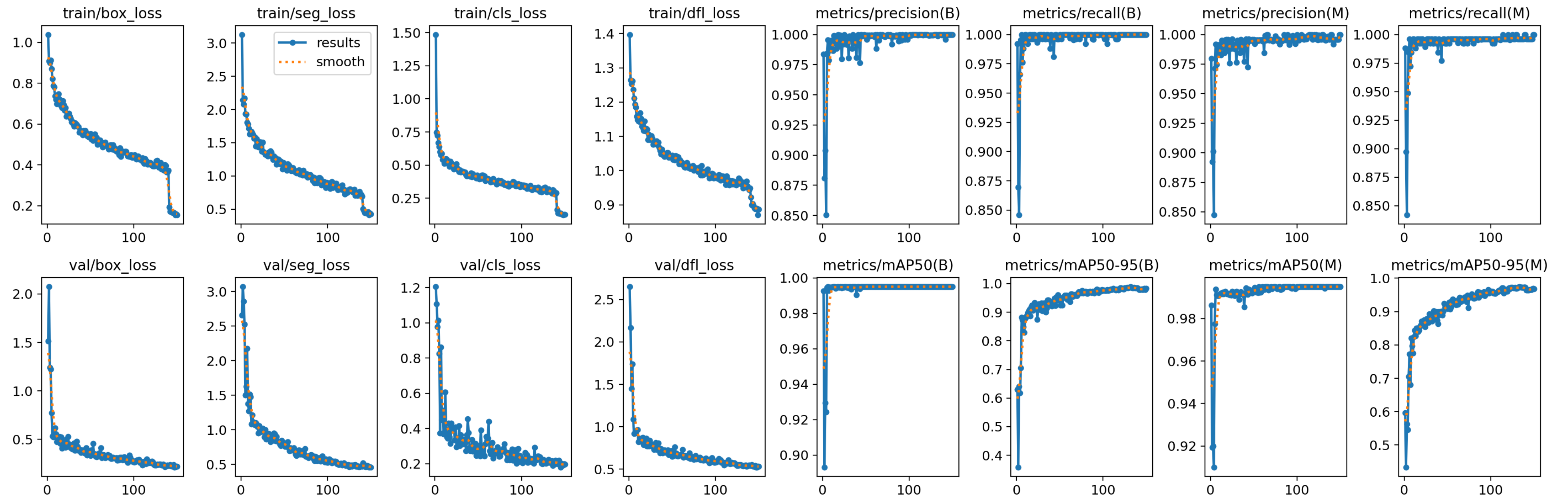
It evaluates their performance on three diverse datasets: **Traffic Signs** (with varying object sizes), African Wildlife (with diverse aspect ratios), and Ships and Vessels (with small-sized objects of a single class), ensuring a comprehensive assessment across datasets with distinct challenges.

Version	Accuracy Rank	Speed Rank	GFLOPs Rank	Size Rank
YOLOv3u-tiny	28	1	6	11
YOLOv3u	20	24	28	28
YOLOv5un	27	6	2	4
YOLOv5us	24	7	8	9
YOLOv5um	17	15	13	18
YOLOv5ul	14	19	21	23
YOLOv5ux	17	27	26	27
YOLOv8n	26	5	4	5
YOLOv8s	23	9	11	10
YOLOv8m	15	17	16	20
YOLOv8l	13	22	22	22
YOLOv8x	8	26	27	26
YOLOv9t	20	12	3	1
YOLOv9s	7	15	10	6
YOLOv9m	4	21	15	15
YOLOv9c	9	25	19	19
YOLOv9e	12	28	24	25
YOLOv10n	25	2	5	3
YOLOv10s	19	3	9	7
YOLOv10m	5	10	12	12
YOLOv10b	9	12	18	14
YOLOv10l	11	17	20	17
YOLOv10x	2	22	23	21
YOLO11n	22	3	1	2
YOLO11s	16	8	7	8
YOLO11m	1	11	14	13
YOLO11l	3	14	17	16
YOLO11x	5	19	25	24

SOTA: State Of The Art for Segmentation

Versions	Precision	Recall	mAP50	mAP50-95	Preprocess Time	Inference Time	Postprocess Time	Total Time	GFLOPs	Size
YOLOv3u	0.75	0.849	0.874	0.781	0.7	8.5	0.4	9.6	207.86	282.4
YOLOV3u tiny	0.845	0.667	0.772	0.682	1.4	0.7	0.3	2.4	24.44	19
YOLOv5un	0.805	0.679	0.749	0.665	0.6	6.6	0.4	7.6	5.65	7.1
YOLOv5us	0.85	0.777	0.827	0.744	0.5	7.8	0.4	8.7	18.58	23.9
YOLOv5um	0.849	0.701	0.83	0.744	1.1	9.5	0.4	11	50.54	64.1
YOLOv5ul	0.831	0.836	0.886	0.799	0.6	9.7	0.4	10.7	106.85	134.9
YOLOv5ux	0.863	0.795	0.867	0.777	1.1	9.8	0.4	11.3	195.2	246.3
YOLOv8n	0.749	0.688	0.777	0.689	0.6	6.8	0.4	7.8	6.55	8.1
YOLOv8s	0.766	0.788	0.806	0.718	0.6	7.8	0.4	8.8	22.59	28.6
YOLOv8m	0.838	0.805	0.845	0.763	1.6	9.1	0.4	11.1	52.12	78.9
YOLOv8l	0.771	0.789	0.853	0.767	0.6	9.2	0.4	10.2	87.77	165
YOLOv8x	0.902	0.744	0.874	0.78	0.6	9.4	0.4	10.4	136.9	257.7
YOLOv9t	0.792	0.748	0.812	0.731	0.5	10	0.4	10.9	4.93	7.7
YOLOv9s	0.763	0.81	0.828	0.75	0.6	11.1	0.4	12.1	15.33	26.8
YOLOv9m	0.864	0.796	0.864	0.784	1	12.1	0.4	13.5	40.98	76.7
YOLOv9c	0.827	0.807	0.852	0.769	1.3	11.6	0.4	13.3	51.8	102.6
YOLOv9e	0.819	0.824	0.854	0.764	0.8	16.1	0.4	17.3	117.5	189.4
YOLOv10n	0.722	0.602	0.722	0.64	1	0.8	0.2	2	5.59	8.3
YOLOv10s	0.823	0.742	0.834	0.744	1.2	1.1	0.2	2.5	15.9	24.7
YOLOv10m	0.834	0.843	0.88	0.781	1.2	2.4	0.2	3.8	32.1	63.8
YOLOv10b	0.836	0.764	0.859	0.765	1	3.1	0.2	4.3	39.7	98.4
YOLOv10l	0.873	0.807	0.866	0.771	1.1	3.8	0.2	5.1	50	126.8
YOLOv10x	0.773	0.854	0.88	0.787	1	6.3	0.2	7.5	61.4	170.4
YOLO11n	0.768	0.695	0.757	0.668	1.2	0.6	0.4	2.2	5.35	6.4
YOLO11s	0.819	0.758	0.838	0.742	1.2	1	0.4	2.6	18.4	21.4
YOLO11m	0.898	0.826	0.893	0.795	1.2	2.4	0.4	4	38.8	67.9
YOLO11l	0.862	0.839	0.889	0.794	1.2	3	0.4	4.6	49	86.8
YOLO11x	0.819	0.816	0.885	0.784	0.9	6.1	0.4	7.4	109	194.8

YOLO Segmentation



Our Demo



Test by yourself

To test and launch the source code we refer to **GitHub** where you can find a **step-by-step guide** to install the requirements and test the developed model.

<https://github.com/leokx6/ADAS>

From Cones Detection to Road Segmentation

Description

This project uses various Python packages to manage YOLO models, process images and videos, and apply analysis and drawing techniques to images. Below are the instructions for setting up the required environment.

System Requirements

Make sure you have installed:

- Python 3.8 or higher
- pip (Python package manager)



Installing Requirements

To run this project, install the Python dependencies listed in requirements.txt using the following command:

```
pip install -r requirements.txt
```



Conclusions & Results

Both the deterministic system and the segmentation one have reached **good performances** in a short period of time, especially on clear tracks. Our track detection system could be helpful, in conjunction with other sensors, during **path and trajectory planning**. The YOLOv11m segmentation model is fast and accurate and is fit to be used on an embedded (but dedicated) processor.

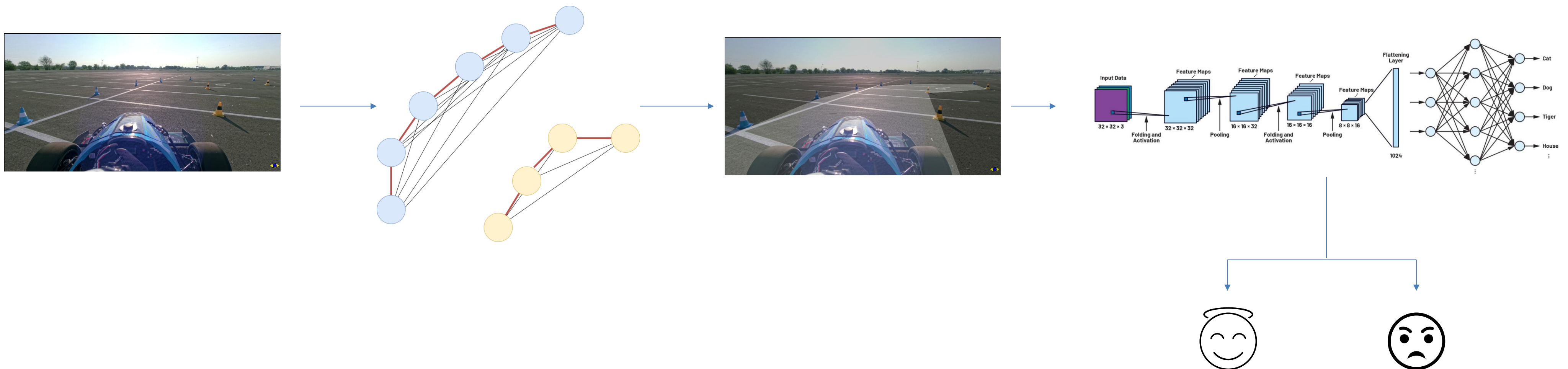
	Inference (ms)	Training time (h)	Tt HPC (h)
Deterministic Algorithm	28	-	-
Object Detection Model (m)	27	80	6
Road Segmentation Model (s)	12	1	0

Future works

- We had to deal with a dataset of **limited size** and scope.
- To generate the segmentation ground truth, we took images from the cone detection dataset, **limiting the amount of images effectively available** for this task.
- Because we generated the mask using a deterministic algorithm, we had to **remove part** of the images from the segmentation dataset due to **errors** during the generation. Around **20%** of the track representing images was removed due to this.
- A team with the capability to film its own car driving around its own designed track could easily overcome the first problem taking only relevant shots.

Future works

- It could be useful to design a **convolutional network** trained to spot **good masks** on the ground truth and remove bad ones automatically. This should allow for a much bigger segmentation dataset that could be automatically created from videos taken from the car during testing events.



Thank you
for
your attention!