A Comparative Analysis of YOLO models for Traffic Sign detection

COMPUTER VISION & DEEP LEARNING

Authors : Serena De Antoni, Francesco Ghezzo

Academic year: 2022-2023

Master degree in Artificial Intelligence - Department of Computer Science - University of Verona

CONTENTS:

1. Introduction 3

1.1. Motivation and rationale……3

1.2. State of the Art……3

2. Objectives 3

3. Yolo Models 4

3.1. Model Selection……5

3.2. Differences between the models……5

3.2.1. YOLOv3……5

3.2.2. YOLOv5……6

3.2.3. YOLOv8……7

3.2.4. Comparison……8

3.2.5. Metrics evaluation……9

4. Dataset 10

4.1. Dataset choice……10

4.2. Dataset preprocessing……10

4.2.1. Classes……10

4.2.2. Labels……10

4.2.3. Data Augmentation……10

5. Experiments 11

6. Main Results Train 14

6.1. Without Data Augmentation……14

6.1.1. YOLOv3 vs YOLOv5m vs YOLOv8m 30 epochs ……14

6.1.2. YOLOv5n YOLOv5m YOLOv5x ……15

6.1.3. YOLOv8n YOLOv8m YOLOv8x ……16

6.2. With Data Augmentation……17

6.2.1. YOLOv3 vs YOLOv5m vs YOLOv8m……17

6.2.2. YOLOv5n YOLOv5m YOLOv5x ……18

6.2.3. YOLOv8n YOLOv8m YOLOv8x ……19

7. Main Results Predict 20

7.1. Without Data Augmentation……20

7.1.1. YOLOv3 vs YOLOv5m vs YOLOv8m……20

7.1.2. YOLOv5n YOLOv5m YOLOv5x ……21

7.1.3. YOLOv8n YOLOv8m YOLOv8x ……22

7.2. With Data Augmentation……23

7.2.1. YOLOv3 vs YOLOv5m vs YOLOv8m……23

7.2.2. YOLOv5n YOLOv5m YOLOv5x ……24

7.2.3. YOLOv8n YOLOv8m YOLOv8x ……25

8. Conclusions 26

9. Future works 26

Bibliography 27

1. INTRODUCTION

1.1 Motivation and rationale

Object detection, a fundamental task in computer vision, entails identifying and locating objects within an image or video frame. Unlike image classification, which assigns a single label to an entire image, object detection goes a step further by not only recognizing objects but also precisely delineating their spatial extents with bounding boxes.

In recent years, deep learning techniques have propelled remarkable advancements in object detection, enabling robust and accurate detection of objects across diverse domains. One of the most important frameworks in this domain is YOLO (You Only Look Once), renowned for its real-time processing capabilities and high accuracy.

In this report, we embark on a comprehensive analysis of the major differences between YOLO versions 3, 5, and 8 models, focusing on their performance when applied to the task of object detection. Specifically, we leverage a custom dataset comprising traffic signs to evaluate the efficacy of each model iteration in detecting and localizing objects of interest. Through experimentation and evaluation, we want to shed light on the differences between YOLO versions 3, 5, and 8.

1.2 State of the art

In the rapidly evolving landscape of computer vision and deep learning, YOLO has emerged as a groundbreaking framework for real-time object detection. It represents a paradigm shift in object detection methodologies, offering remarkable speed and accuracy. Traditional object detection approaches often involve two-stage methods, where regions of interest are first generated and then classified. In contrast, YOLO revolutionizes object detection by processing images in a single pass through a convolutional neural network (CNN), enabling simultaneous prediction of bounding boxes and class probabilities directly from raw pixels. YOLO divides the input image into a grid, predicting bounding boxes and class probabilities for each grid cell, ensuring accurate localization of objects of various sizes and aspect ratios. Its single-pass architecture facilitates real-time inference, making it ideal for applications requiring fast and efficient object detection. The last version of YOLO is the YOLOv8 model that builds upon the success of previous YOLO versions and introduces new features and improvements to further boost performance and flexibility. YOLOv8 is designed to be fast, accurate, and easy to use, making it an excellent choice for a wide range of object detection and tracking, instance segmentation, image classification and pose estimation tasks**.**

2. OBJECTIVES

The objective of this project is to evaluate and compare three different models: YOLOv3, YOLOv5 and YOLOv8 with two traffic signs dataset. In particular we compare YOLOv3, YOLOv5n, YOLOv5m, YOLOv5x, YOLOv8n, YOLOv8m, YOLOv8x with a traffic sign dataset with and without data augmentation**.**

3. YOLO MODELS

YOLO, which stands for "You Only Look Once," is a deep learning model for real-time object detection. Originally introduced by Joseph Redmon et al. in 2016, YOLO revolutionized the field of computer vision by offering a single unified framework for detecting objects. Main characteristics of YOLO object detection system are:

1. Single-Shot Detection (or one-stage detectors): YOLO operates on the principle of one-stage detection, meaning it performs both object localization (prediction of the bounding boxes) and classification (class probabilities) in a single pass through the neural network.
2. Neural Network Architecture: The architecture of YOLO typically consists of a convolutional neural network (CNN) backbone with 24 conv layers followed by 2 fully connected layers. Alternating 1x1 convolutional layers reduce the features space form preceding layers.

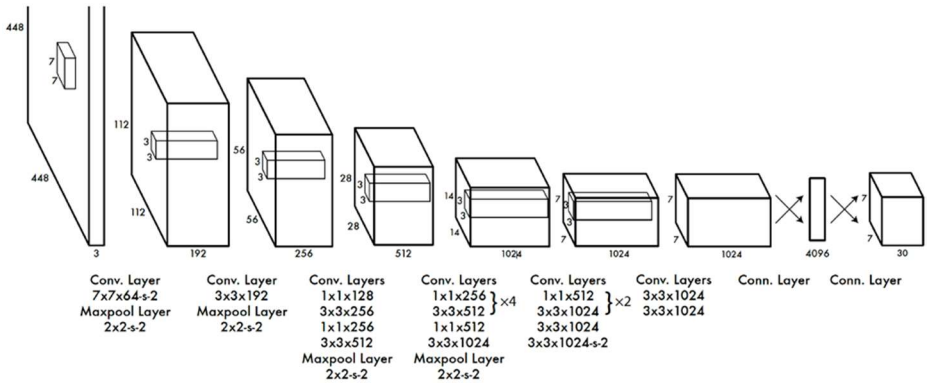


Fig 1: example of YOLO's architecture of the CNN model that forms the backbone

3. Unified Detection: It unifies the separate components of object detection into a single network. It uses the features from the entire image to predict each bounding box and predicts all bounding boxes across all classes from an image simultaneously.
4. Grid-Based and Bounding Box Prediction: YOLO divides the input image into a SxS grid of cells and predicts bounding boxes within each cell. Each grid cell is responsible for detecting objects whose center falls within its spatial location. For each grid cell, YOLO also predicts multiple bounding boxes along with associated confidence scores and class probabilities. Each bounding box consists of four coordinates (x_c, y_c, width, height) representing the location and size of the detected object. The confidence score indicates the model's confidence that the box contains an object, while class probabilities represent the likelihood of belonging to each predefined class. After the prediction of a bounding box, YOLO uses IoU (Intersection Over Union) to choose the right bounding box of an object for the grid cell.
5. Non-Maximum Suppression (NMS): After generating bounding box predictions, YOLO applies non-maximum suppression to eliminate redundant detections and refine the final set of predictions. NMS ensures that each object is detected only once by selecting the most confident bounding boxes and suppressing others that have significant overlap with them.
6. Anchor Boxes: In some versions of YOLO, anchor boxes are used to improve the accuracy of object localization. Anchor boxes are predefined bounding boxes of different shapes and aspect ratios, which serve as reference templates during training. The model adjusts these anchor boxes based on the distribution of object shapes and sizes in the training dataset.
7. Training: YOLO is typically trained using a large dataset of labelled images, where each image is annotated with bounding boxes and class labels. The model is trained to minimize a combination of localization loss and classification loss. To calculate loss, YOLO uses the sum of squared error.

$$
\begin{aligned}
&\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}
$$

8. Real-Time Performance: One of the key advantages of YOLO is its real-time performance, allowing for fast and efficient object detection on live video streams or high-resolution images (45-150 FPS).
9. Disadvantages: YOLO may struggle with accurately localizing small objects and occasionally produces false positives or negatives (the main problem is when the background covers the majority of the photo). Training YOLO often requires significant computational resources and data augmentation techniques to achieve optimal performance.

3.1 Model selection

For our project we decided to evaluate YOLOv3, YOLOv5 and YOLOv8, in particular we compare n, m and xl size of v5 and v8 models, but for YOLOv3 we find only one model that we have compared to YOLOv5m and YOLOv8m.

3.2 Differences between the models

3.2.1 YOLOv3

YOLOv3 (You Only Look Once version 3) was released in 2018. In the previous version of YOLO, the feature extraction backbone is Darknet19, which struggled in detecting small objects, so in YOLOv3 it was changed to Darknet53 to address this problem. In that work, Residual block, skip connections and up-sampling were introduced, which significantly improved the accuracy of the algorithm. YOLOv3 uses feature pyramid network (FPN) as a neck. The neck plays an important role in extracting features maps from different stages which is composed of several bottom-up and top-down paths and the head is composed of a YOLO layer. The role of head in one stage detector is to perform a final prediction which is composed of a vector containing bounding box coordinates: x_c, y_c, width, height, class label, and class probability. So, first the image is fed to Darknet53 for feature extraction, afterwards fed to the feature pyramid network for feature fusion and finally, the YOLO layer generates the results. [2]

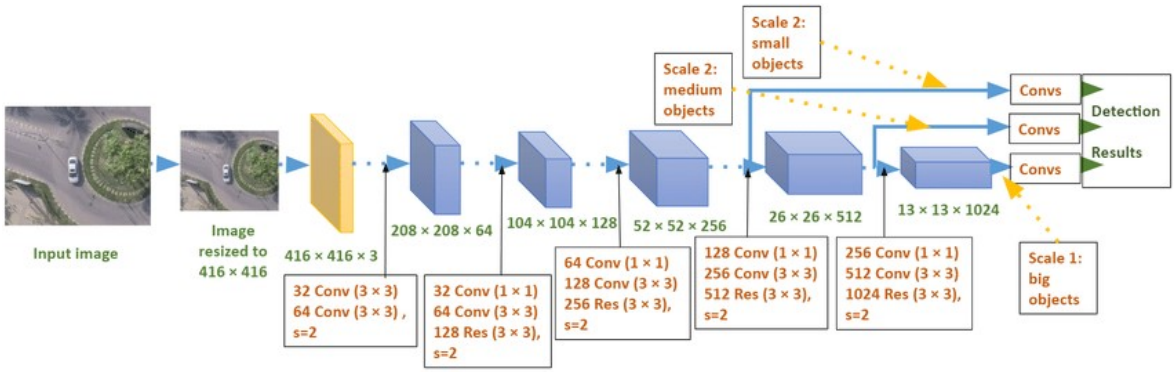


Fig 2: example of YOLOv3 architecture

3.2.2 YOLOv5

YOLOv5 follows the same design concepts as previous versions of YOLO but with an upgraded architecture, EfficientDet, and a PyTorch framework called CSPDarknet53 as its backbone. This refinement tackles repetitive gradient information within large backbones by integrating gradient changes into feature maps, enhancing inference speed, accuracy, and reducing model size. Additionally, YOLOv5's neck incorporates the Path Aggregation Network (PANet) to amplify information flow and introduces a new Feature Pyramid Network (FPN) for improved propagation of low-level features and object localization accuracy. Moreover, YOLOv5's head produces three distinct output feature maps for multi-scale prediction, facilitating efficient detection of objects spanning various sizes. Using a more complex architecture in YOLOv5 allows it to achieve higher accuracy and better generalization to a wider range of object categories [3]. Furthermore, YOLOv5 introduces "dynamic anchor boxes," a novel approach to anchor box generation. This method employs a clustering algorithm to group ground truth bounding boxes into clusters, utilizing the centroids of these clusters as anchor boxes. This innovation ensures that the anchor boxes closely align with the detected objects' sizes and shapes. Additionally, YOLOv5 incorporates "spatial pyramid pooling" (SPP), a specialized pooling layer aimed at reducing the spatial resolution of feature maps. SPP enhances detection performance on small objects by enabling the model to perceive objects across multiple scales. There are various models of YOLOv8 in which change the size and some other parameters as we can see in Figure3. [5]

Model	YAML	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
yolov5nu.pt	yolov5n.yaml	640	34.3	73.6	1.06	2.6	7.7
yolov5su.pt	yolov5s.yaml	640	43.0	120.7	1.27	9.1	24.0
yolov5mu.pt	yolov5m.yaml	640	49.0	233.9	1.86	25.1	64.2
yolov5lu.pt	yolov5l.yaml	640	52.2	408.4	2.50	53.2	135.0
yolov5xu.pt	yolov5x.yaml	640	53.2	763.2	3.81	97.2	246.4

Fig 3: Different YOLOv8 models

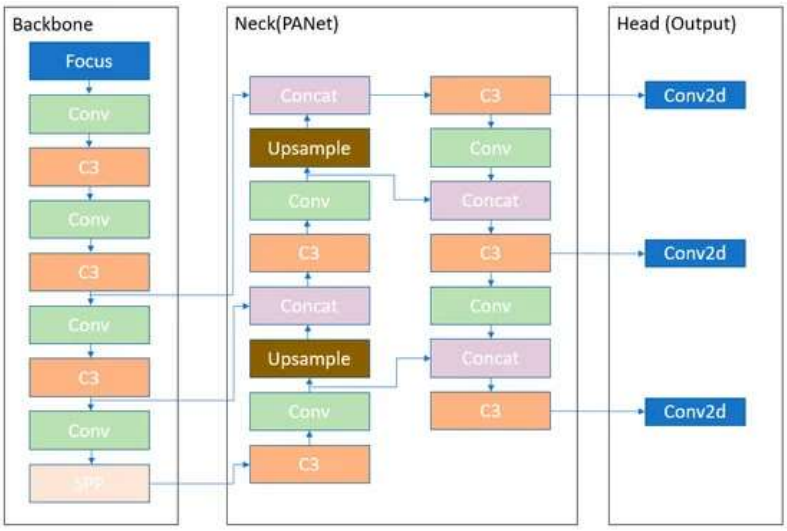


Fig 4: example of YOLOv5 architecture

3.2.2 YOLOv8

YOLOv8 is a state-of-the-art model that builds on the success of previous YOLO and introduces new features and improvements to further boost performance and flexibility. It can be trained on large datasets and is able to run on various hardware platforms, from CPU to GPU. Not only that, but a key feature of YOLOv8 is its scalability. It is designed as a framework so it supports all previous versions of YOLO, making it easy to switch between different versions and compare their performance.

Among the key innovations of YOLOv8 are its advancements in backbone architecture and some modified modules to make it more lightweight. Additionally, it incorporates the Spatial Pyramid Pooling with Fusion (SPPF) module from previous architectures, further optimizing efficiency. In the realm of feature pyramid networks, YOLOv8 refines the PAN (Path Aggregation Network) concept by deleting convolutions in the upsampling stage. The models also abandoned the previous Anchor-Base and used the idea of Anchor-Free, that is more flexible. YOLOv8 uses VFL Loss as classification loss and DFL Loss+CIOU Loss for detection and abandoned the previous IOU matching or unilateral ratio allocation, to use the Task-Aligned Assigner matching method. There are various models of YOLOv8 in which change the size and some other parameters as we can see in Figure 5. [5]

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Fig 5: Different YOLOv8 models

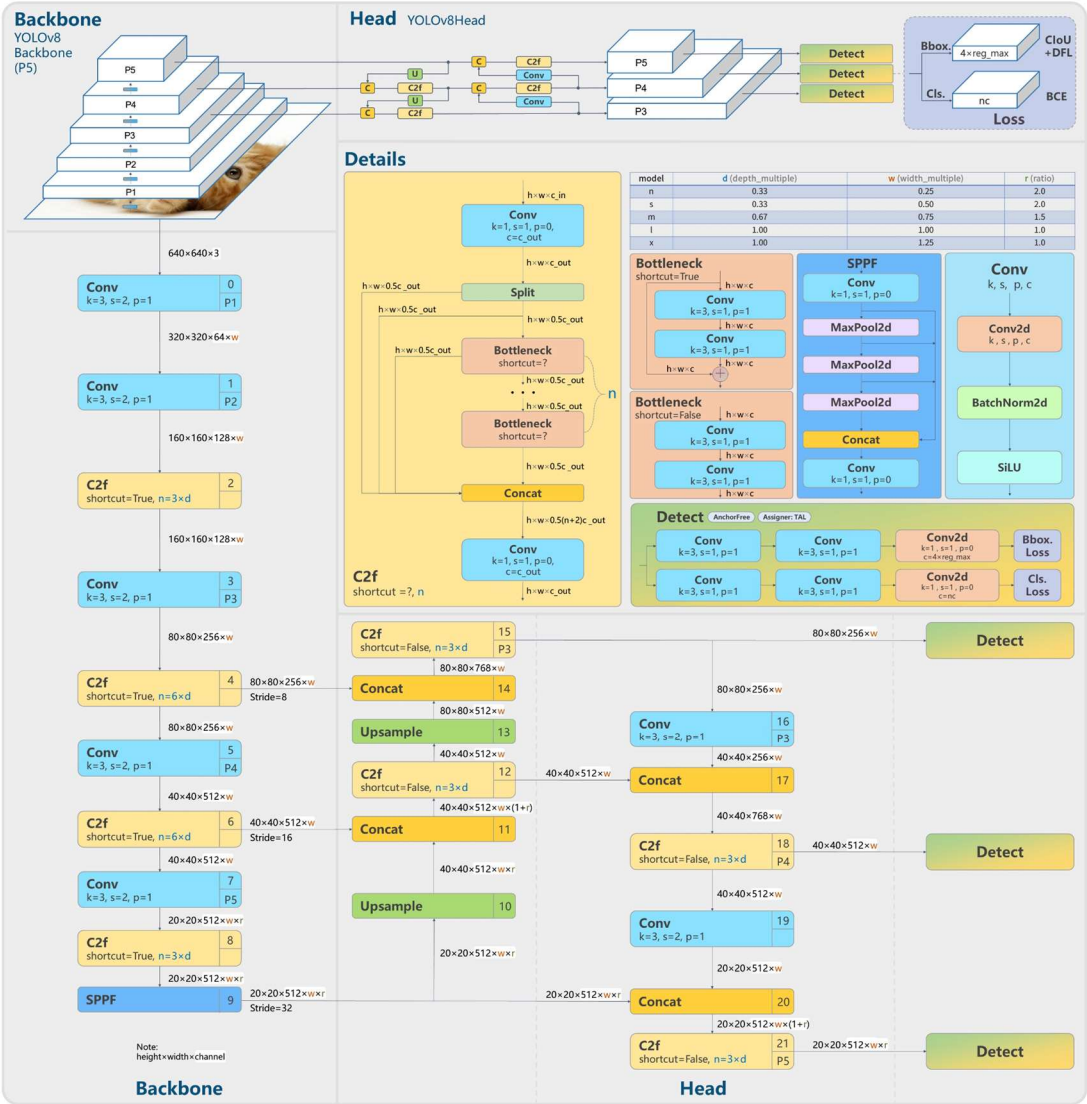


Fig 6: example of YOLOv8 architecture

3.2.4 Comparison

	YOLOv3	**YOLOv5**	**YOLOv8**
Neural Network Type	Fully convolution	Fully convolution	Fully convolution
Backbone Feature Extractor	Darknet53	CSPDarknet53	modified CSPDarknet53
Loss Function	Binary cross entropy	Binary cross entropy + Logits loss function	VFL + DFL Loss + CIOU Loss
Neck	FPN	FPN + PANet	Modified PAN-FPN
Head	YOLO layer	YOLO layer	YOLO, CNN
Activation function	Mish	SiLU	Leaky ReLU
Optimizer	SGD	Adam	Adam, SGD
Momentum	0.937	0.9	0.9
Weight decay	0.0005	0.0005	0.0005
Learning rate	0.001	0.0001	0.001

Table 7: main differences between YOLOv3, YOLOv5 and YOLOv8 architectures

3.2.5 Metrics evaluation

For evaluating all the YOLO's variants for object detection there are some metrics that are commonly used [7]:

1. Average precision (AP): Average precision measures the model's accuracy in detecting objects at different levels of precision. It calculates the area under the precision–recall curve (AUC-PR) for different thresholds. The formula for calculating AP is:

$$AP = \sum_{i=1}^{n} (R_i - R_{i-1})P_i$$

2. Intersection over Union (IoU): Intersection over Union is a metric that measures the overlap between the predicted bounding box and the ground-truth bounding box. It is calculated as the ratio of the intersection area to the union area of the two boxes. The formula for calculating IoU is:

$$IoU = \frac{Area\ of\ Overlap}{Area\ of\ Union}$$

3. Mean Average Precision (mAP): Mean Average Precision is the average of the AP's calculated at different levels of precision. It is used to measure the model's overall performance across all classes. The mAP can be calculated as:

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i$$

4. False-Positive Rate (FPR): The false Positive Rate is the proportion of negative samples incorrectly classified as positive. It is used to measure the model's performance in detecting false positives. The formula for calculating FPR is:

$$FPR = \frac{False\ Positive}{False\ Positive\ +\ True\ Negative}$$

5. Recall: The recall is the proportion of positive samples model correctly identifies. It is used to measure the model's performance in detecting true positives. The formula for calculating Recall is:

$$FPR = \frac{True\ Positive}{True\ Positive\ +\ False\ Negative}$$

4. DATASET

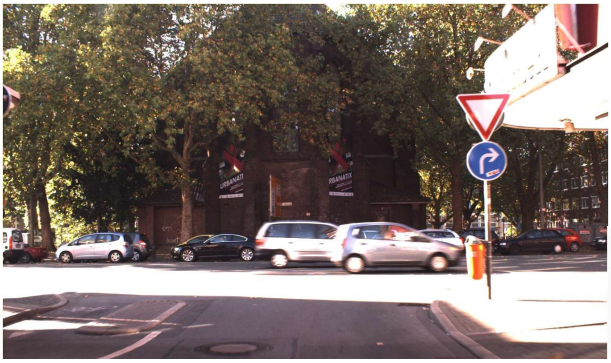
In this section we will explain the dataset selection for this project.

4.1 Dataset choice

We select a dataset that contains 741 labelled photos of roads with different traffic signs.[X] The dataset is divided into 444 train images , 148 validation images and 149 test images. Dataset consists of images in *.jpg format and *.txt files next to every image that have the same names as images files have. These *.txt files include annotations of bounding boxes of Traffic Signs in the YOLO format:

[Class Number] [Center in x] [Center in y] [Width] [Height]

In which center in x, center in y, width and height are normalized by the size of the image.
Traffic signs in this dataset are grouped into four categories: "prohibitory", "danger", "mandatory" and "other". [1]



```
2 0.7977941176470589 0.44375 0.0661764705882353 0.1125
3 0.7933823529411764 0.321875 0.09117647058823529 0.13875
```

Fig 8: example of an image (left) and is associated label (right)

4.2 Dataset preprocessing

We noticed that the classes were few and reductive so we increased the number of classes. Additionally, upon realizing that no zebra crossing signs were labelled, we manually labelled some of them to address the issue.

4.2.1 Classes

We have increase the number of classes from 4 to 10 that are: "prohibitory", "danger", "directional signs", "speed limit", "pedestrian and bicycle", "End of ..." , "stop" , "give way", "other", "zebra crossing".

4.2.2 Labels

At this point, we needed to modify each .txt file to adjust to align with the correct label class we added. Then we needed to add the labels for the zebra crossing signs and the respective bounding box in the *.txt files with the new classes, all done by hand each image by image.

```
1 0.5727941176470588 0.534375 0.020588235294117647 0.03125
9 0.372794 0.47875 0.0102941 0.0175
9 0.477942 0.44625 0.0110294 0.01875
```

Fig 9: example of an image (left) in which there are some pedestrian crossing signs and we add in the associated label the corresponding values (right)

4.2.3 Data Augmentation

To enhance dataset diversity, initially, we tried to implement an exhaustive augmentation algorithm, incorporating all conceivable modifications. However, this approach presented a significant challenge: rotation and flipping resulted in label alterations across all *.txt files. Consequently, we opted not to pursue this avenue and instead focused on augmentation techniques that only manipulate image characteristics such as contrast and brightness. Then we apply data augmentation for changing randomly the brightness and the contrast across the entire train set. Our process entailed duplicating both the images and labels folders of the original training set, applying data augmentation only to the replicated folder. We merge the first training set folder with the augmented one. Consequently, now we have two dataset, the ‘normal’ one that contains 444 train

images, 148 validation images and 149 test images and the new 'augmented' dataset that contains 888 train images, 148 validation images and 149 test images.

Fig 10: example of an augmented image : the first is the original image, the second is the augmented one

5. EXPERIMENTS

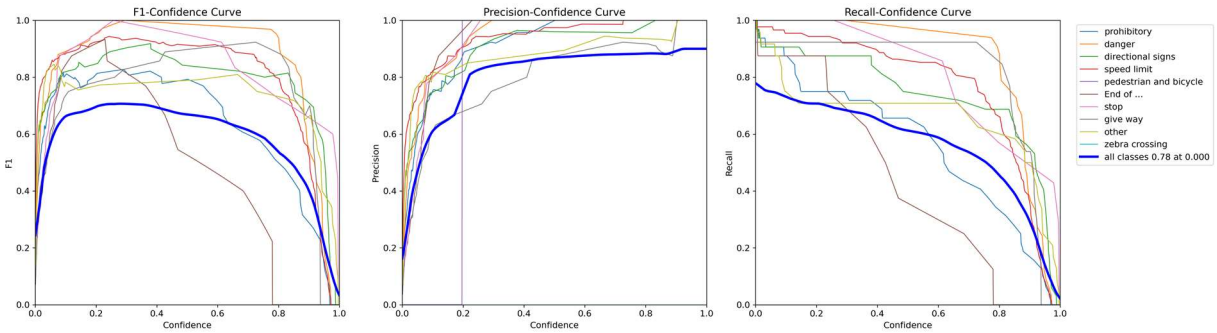
We decided to compare:

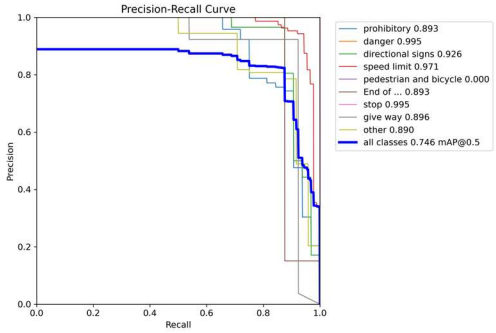
- YOLOv3 with YOLOv5m and YOLOv8m
- YOLOv5n with YOLOv8n
- YOLOv5m with YOLOv8m
- YOLOv5x with YOLOv8x

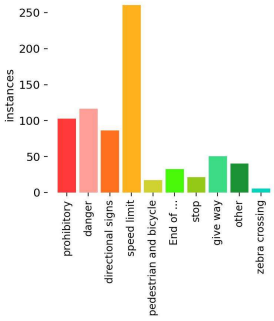
So we took the different models and we did fine tuning with our custom dataset.
We train all the models for 5 epochs, and then we decide to train YOLOv3, YOLOv5m, YOLOv8m with not augmented dataset also for 15 and 30 epochs (we decided to report only the results of 30 epochs train). The motivation behind the fact of training only the "m" models with 30 epochs but without the augmentation dataset falls into the computational problem. We encountered some challenges while training the V3 and V5 models due to their considerable size and inefficiency in training. Despite attempting to train them with an augmentation dataset across 30 epochs, we faced constraints with GPU power and memory.

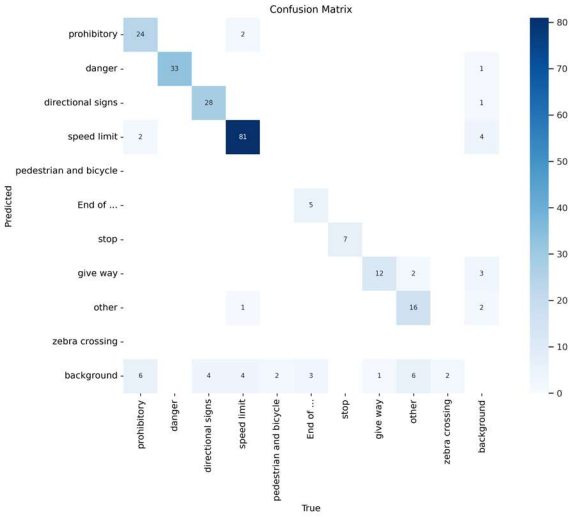
For each model's fine tuning we get out:

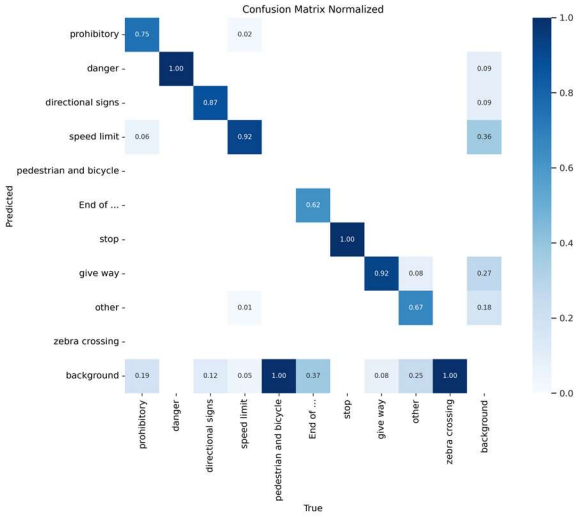
- best.pt: The weights of the best epoch of the model
- last.pt: The weights of the last epoch of the model
- F1_curve.png: graph of the F1-Confidence Curve
- PR_curve.png: graph of the Precision-Recall Curve
- P_curve.png: graph of the Precision_Confidence Curve
- R_curve.png: graph of the Recall_Confidence Curve
- confusion_matrix.png: Confusion Matrix
- confusion_matrix_normalized.png: Confusion Matrix Normalized
- Labels.png: the labels distribution
- results.csv: file that contains all the scores
- results.png: graphs of the scores











epoch	train/box_loss	train/cls_loss	train/dfl_loss	metrics/precision(B)	metrics/recall(B)	metrics/mAP50(B)	metrics/mAP50-95(B)	val/box_loss	val/cls_loss	val/dfl_loss	lr/pg0	lr/pg1	lr/pg2
1	1.003	4.2062	0.87035	0.68547	0.45896	0.4744	0.36144	0.91433	1.7281	0.81961	0.00019278	0.00019278	0.00019278
2	0.86121	1.5344	0.8432	0.6208	0.53244	0.55794	0.39755	0.97795	1.1683	0.82679	0.00031495	0.00031495	0.00031495
3	0.78949	1.2068	0.83625	0.86913	0.59118	0.67613	0.51157	0.88494	0.81616	0.81777	0.00035794	0.00035794	0.00035794
4	0.79222	0.9445	0.82869	0.90433	0.57926	0.70776	0.54429	0.83947	0.70476	0.80823	0.00028988	0.00028988	0.00028988
5	0.77562	0.81817	0.81998	0.83393	0.69013	0.74593	0.5804	0.77821	0.59775	0.80149	0.00028988	0.00028988	0.00028988

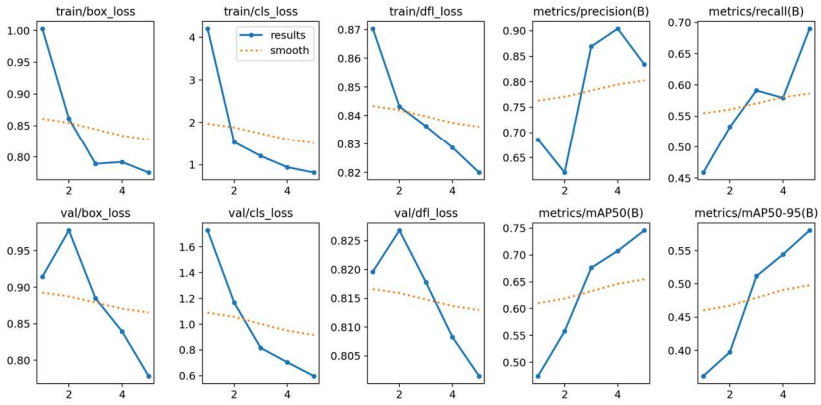


Fig 11: examples of all the information that we receive from the fine tuning

And for each model's prediction we get out:

- All the images with bounding boxes and precision
- A *.txt file for each image that contains the labels associated
- F1_curve.png: graph of the F1-Confidence Curve
- PR_curve.png: graph of the Precision-Recall Curve
- P_curve.png: graph of the Precision_Confidence Curve
- R_curve.png: graph of the Recall_Confidence Curve
- confusion_matrix.png: Confusion Matrix
- confusion_matrix_normalized.png: Confusion Matrix Normalized

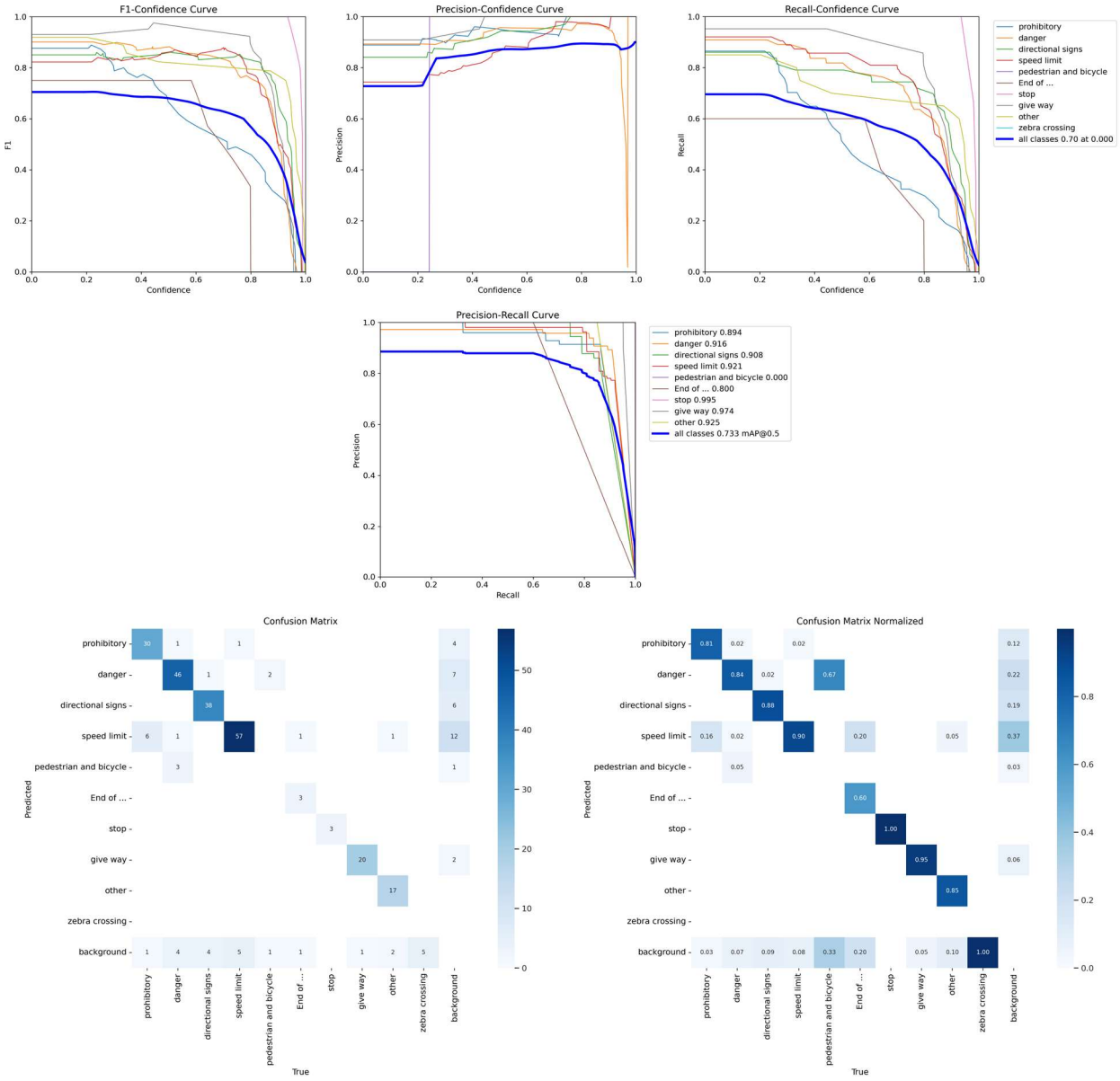


Fig 12: example of all the information that we receive from the prediction

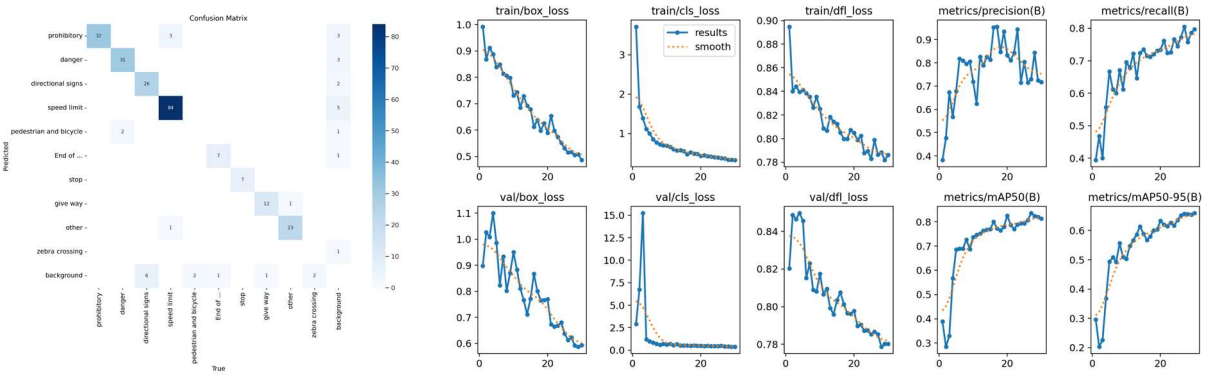
6. MAIN RESULTS TRAIN

In all our main training results, we aim to highlight the distinctions between the models' performances by presenting the confusion matrix and the respective graphs showing train and validation losses, precision, and recall.

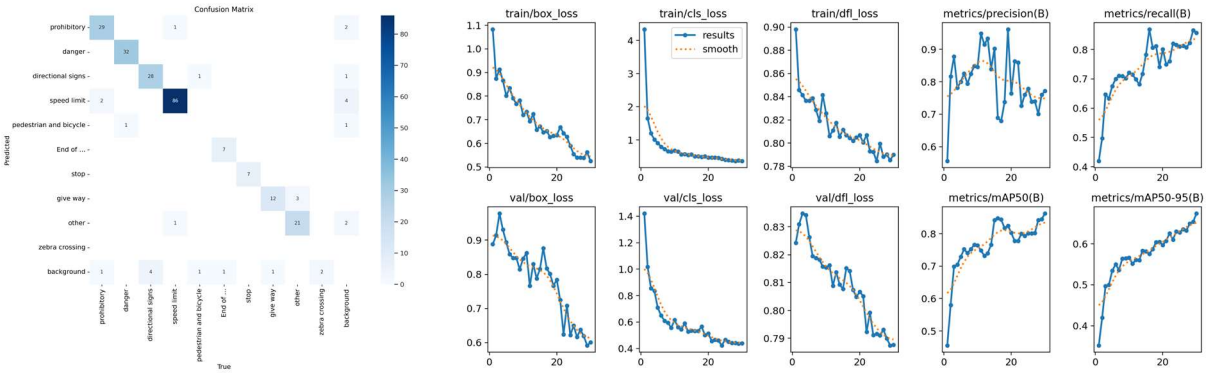
6.1 Without Data Augmentation

6.1.1 YOLOv3 vs YOLOv5m vs YOLOv8m 30 epochs

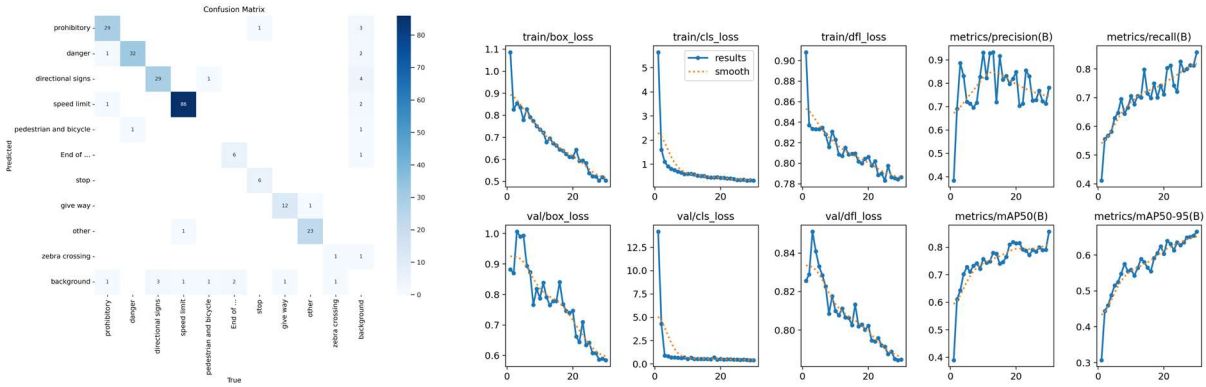
<u>YOLOv3m</u>



<u>YOLOv5m</u>

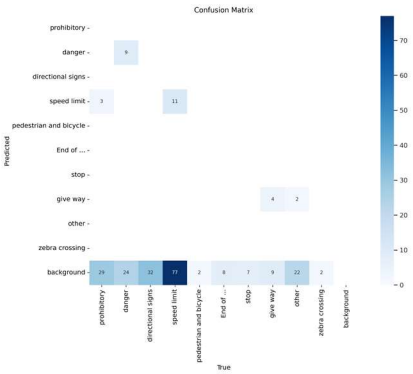


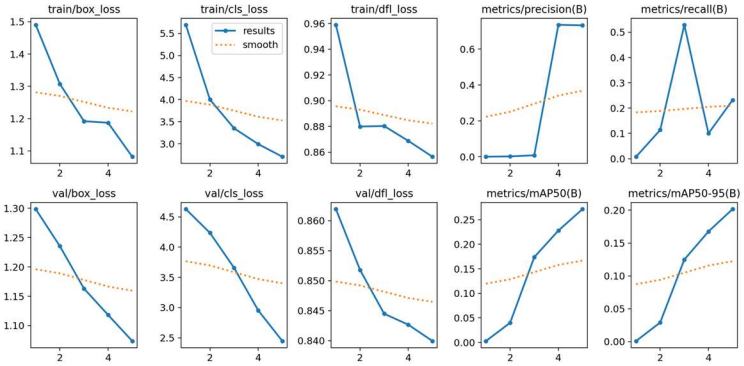
<u>YOLOv8m</u>



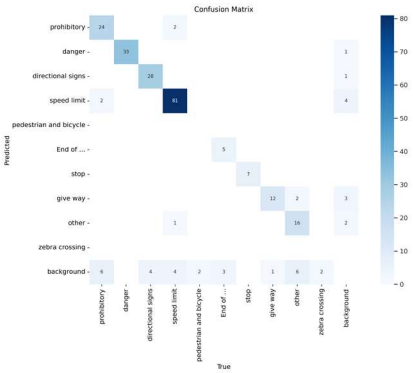
6.1.2 YOLOv5n YOLOv5m YOLOv5x

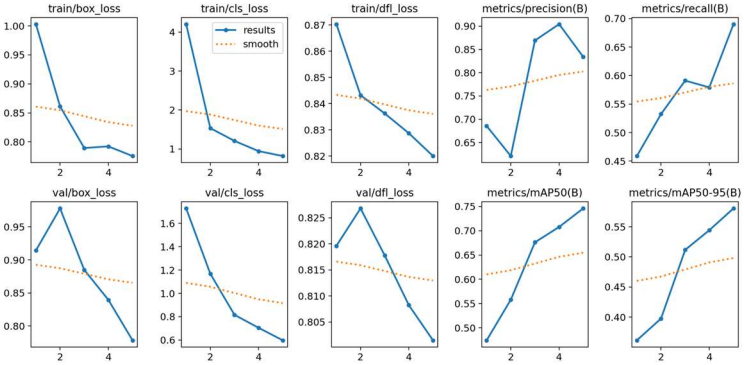
YOLOv5n



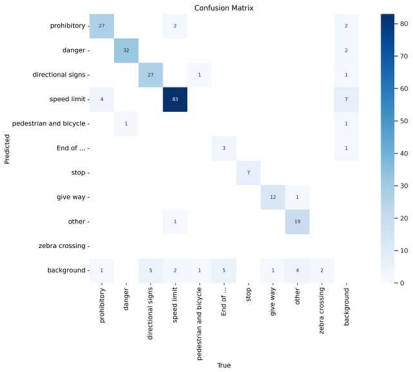


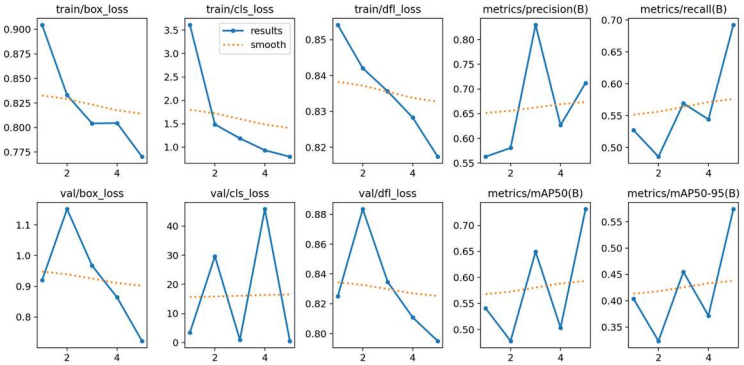
YOLOv5m





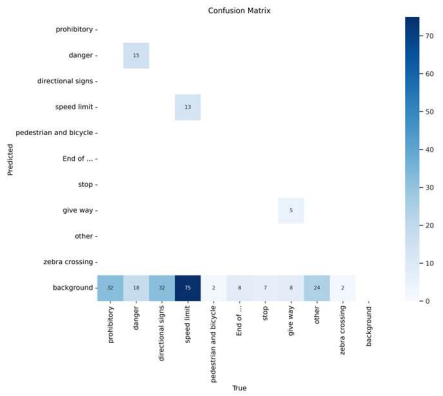
YOLOv5x

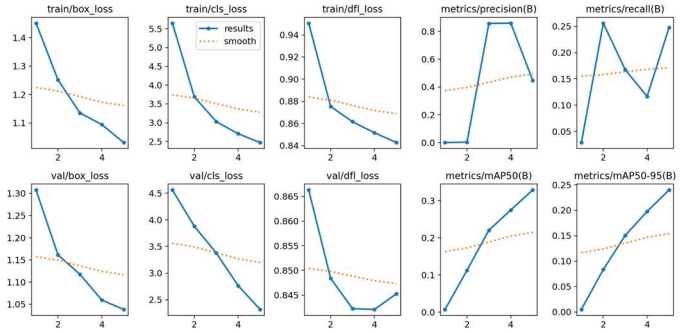




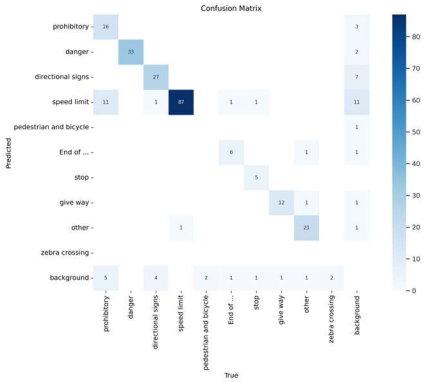
6.1.3 YOLOv8n YOLOv8m YOLOv8x

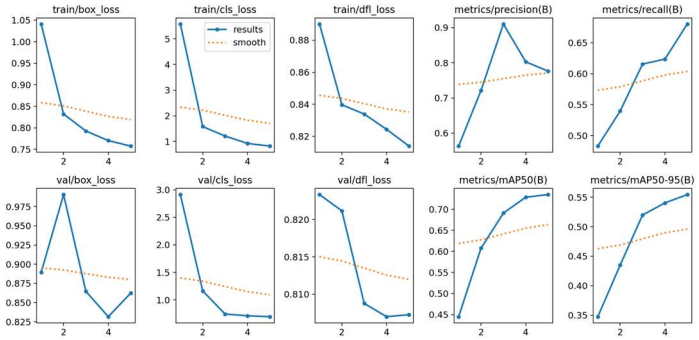
YOLOv8n



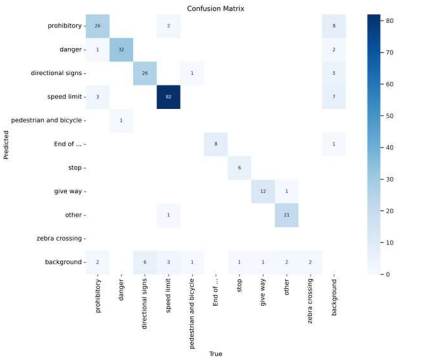


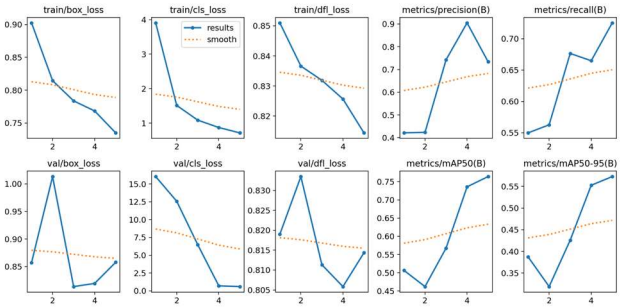
YOLOv8m





YOLOv8x

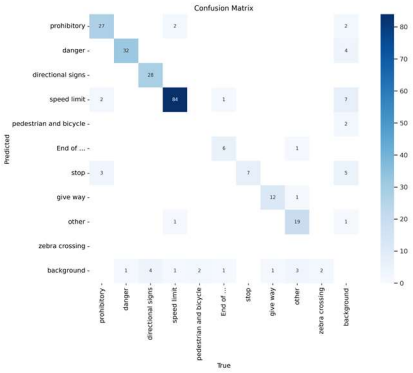


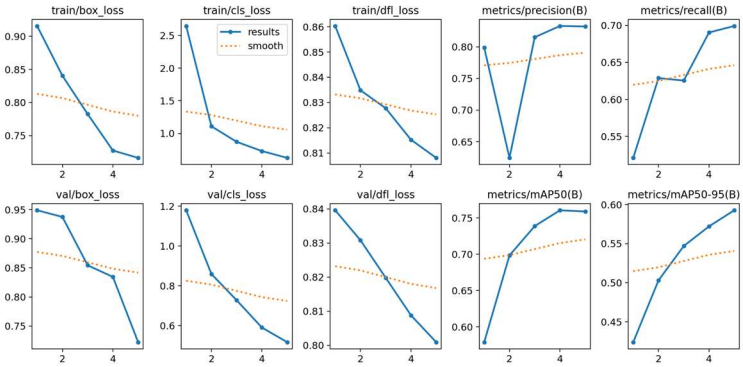


6.2 With Data Augmentation

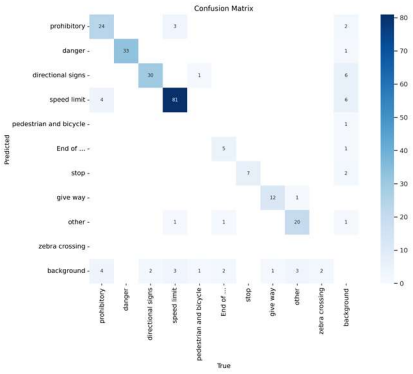
6.2.1 YOLOv3 vs YOLOv5m vs YOLOv8m

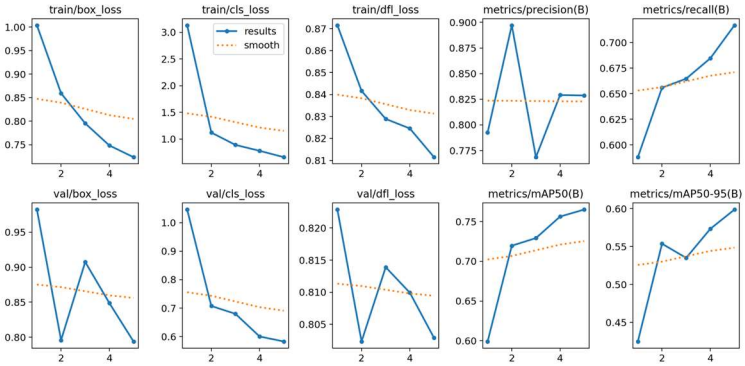
YOLOv3



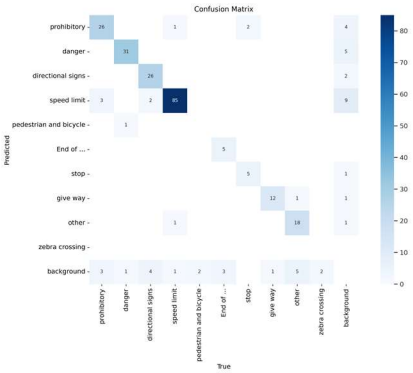


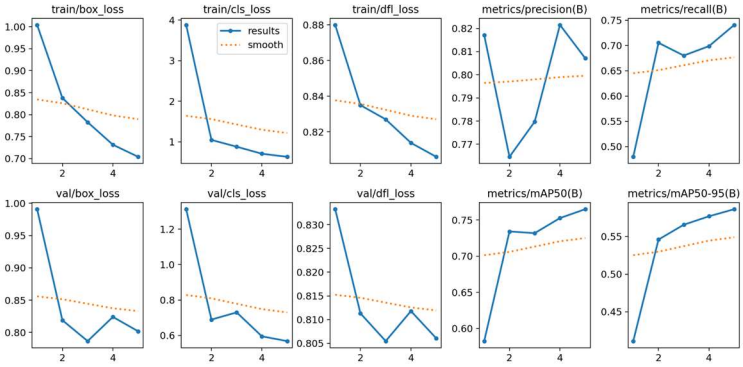
YOLOv5m





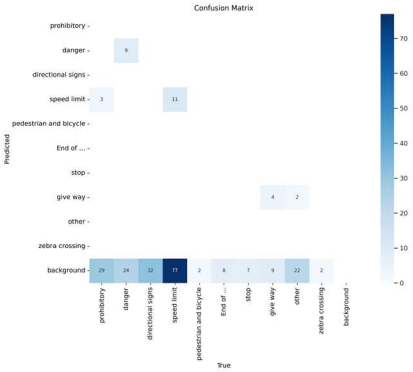
YOLOv8m

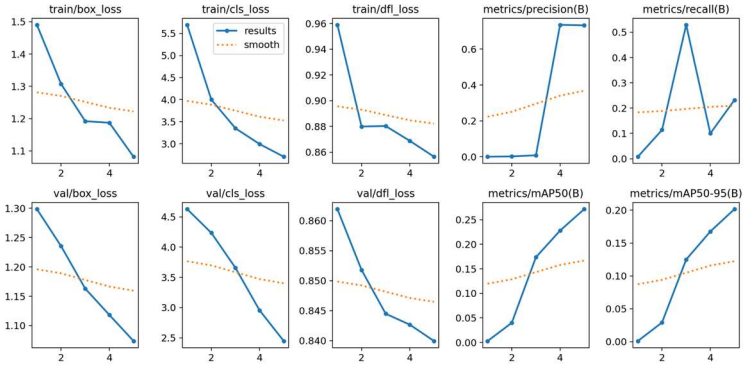




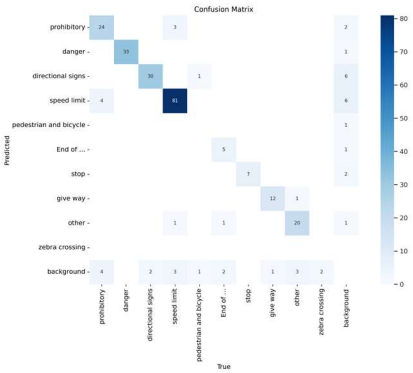
6.2.2 YOLOv5n YOLOv5m YOLOv5x

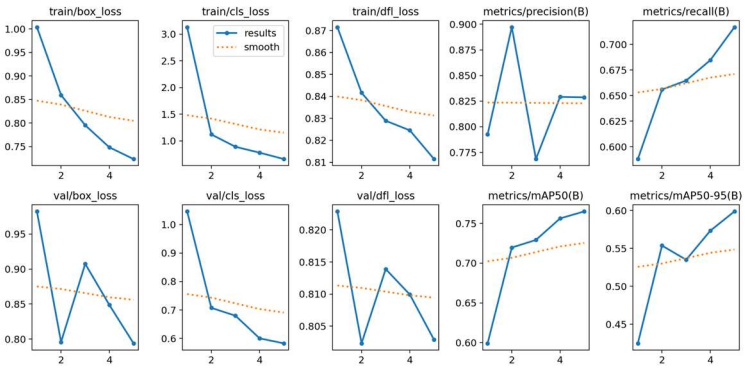
YOLOv5n



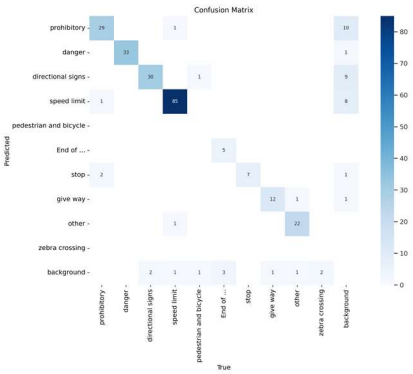


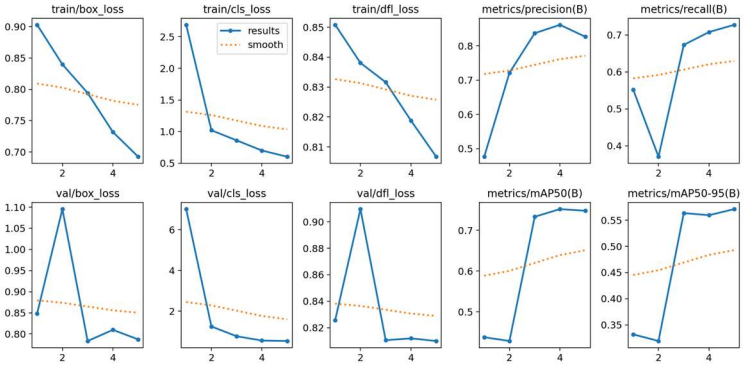
YOLOv5m





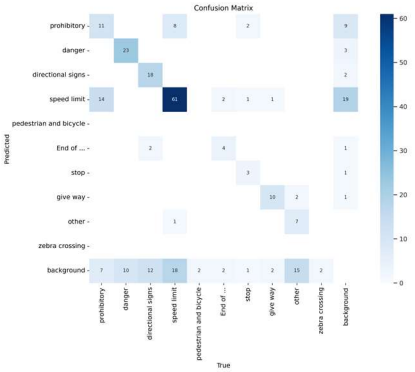
YOLOv5x

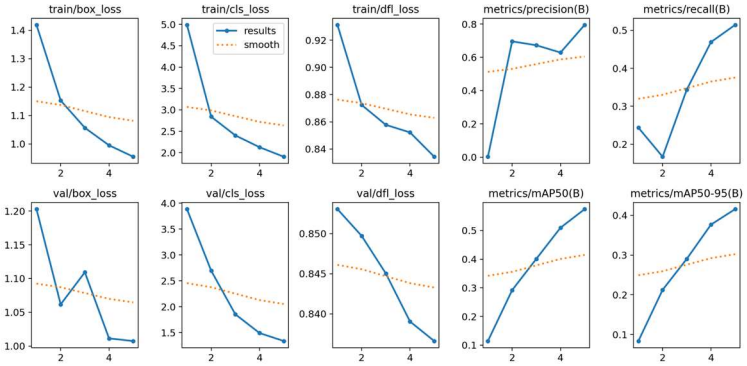




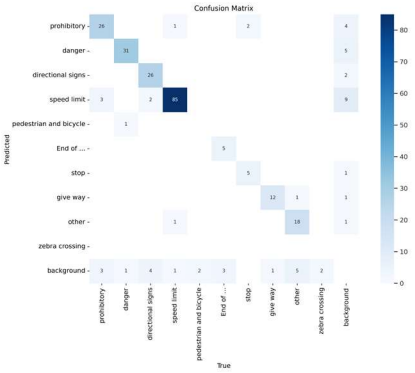
6.2.3 YOLOv8n YOLOv8m YOLOv8x

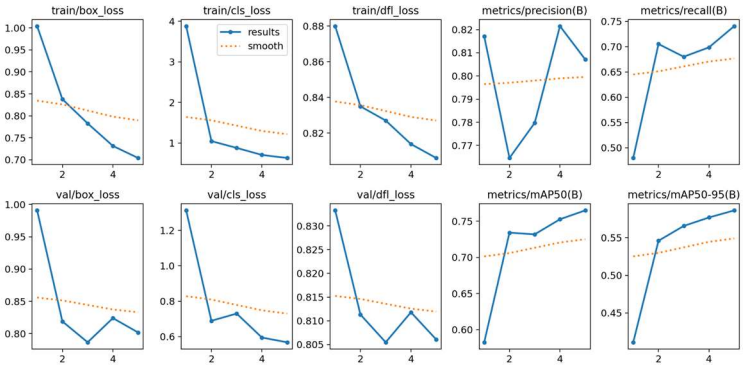
YOLOv8n



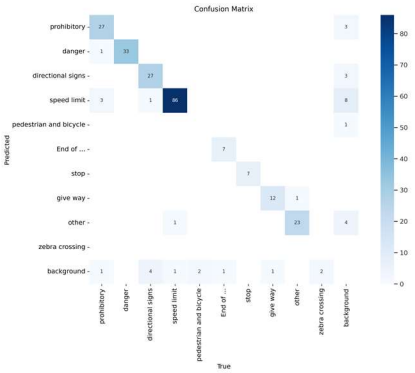


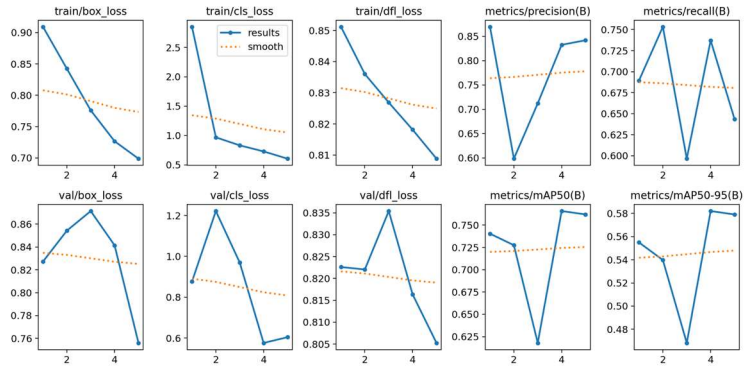
YOLOv8m





YOLOv8x





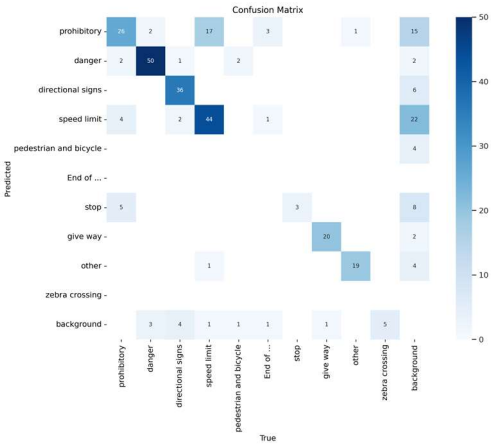
7. MAIN RESULTS PREDICT

In all our main predicted results, we aim to highlight the distinctions between the models' performances by presenting the confusion matrix and one example of the output of the net.

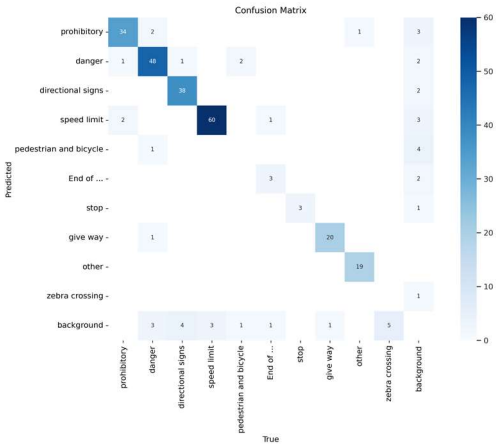
7.1 Without Data Augmentation

7.1.1 YOLOv3 vs YOLOv5m vs YOLOv8m

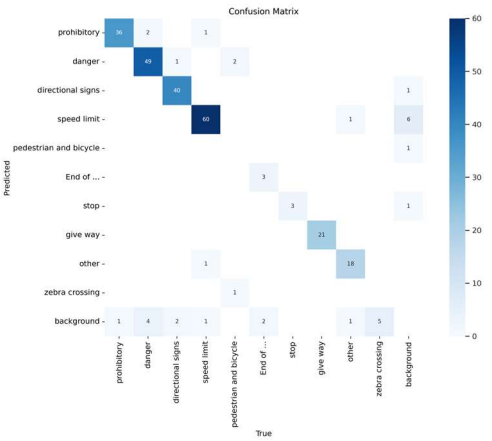
YOLOv3m



YOLOv5m

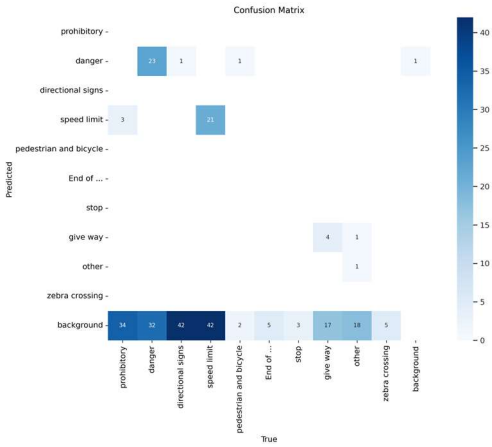


YOLOv8m

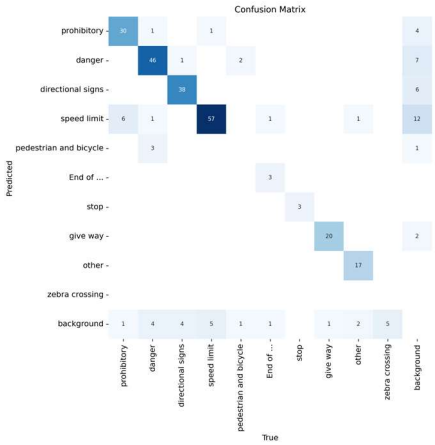


7.1.2 YOLOv5n YOLOv5m YOLOv5x

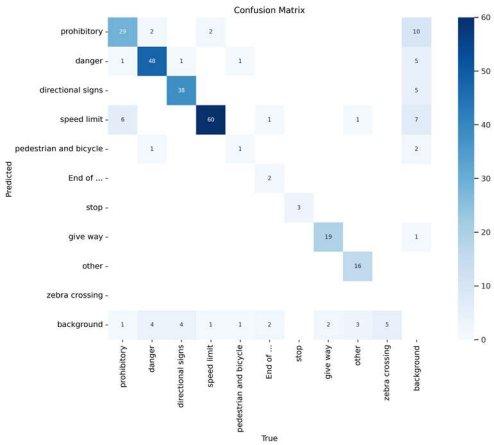
YOLOv5n



YOLOv5m

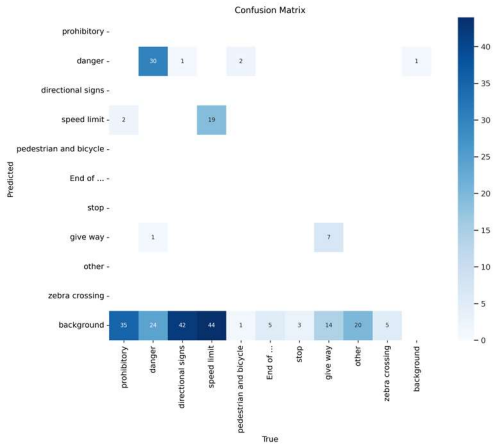


YOLOv5x

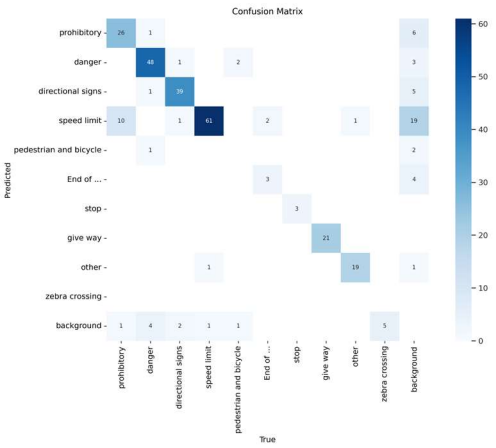


7.1.3 YOLOv8n YOLOv8m YOLOv8x

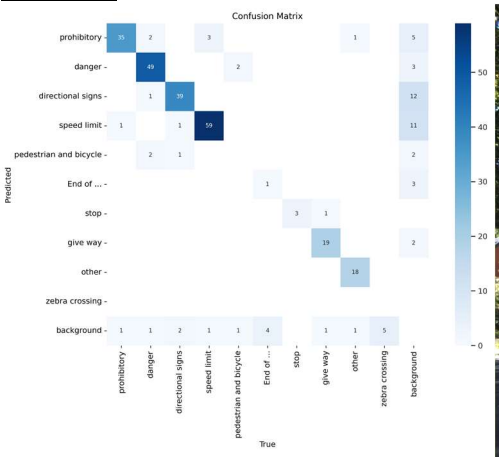
YOLOv8n



YOLOv8m



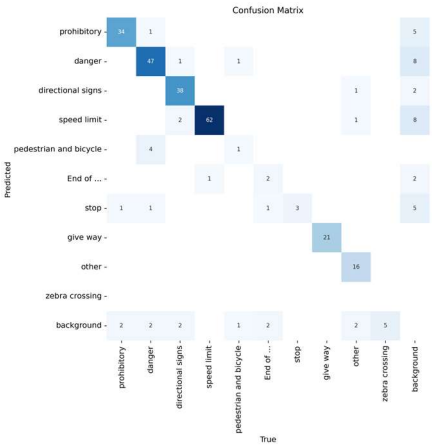
YOLOv8x



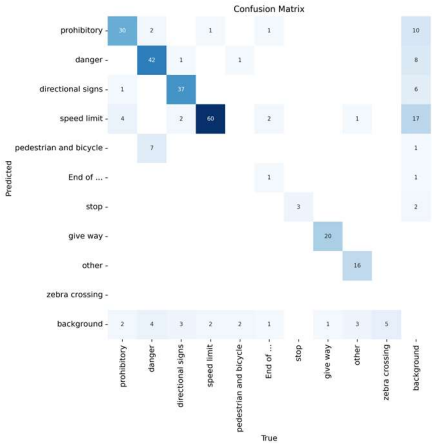
7.2 With Data Augmentation

7.2.1 YOLOv3 vs YOLOv5m vs YOLOv8m 5 epochs

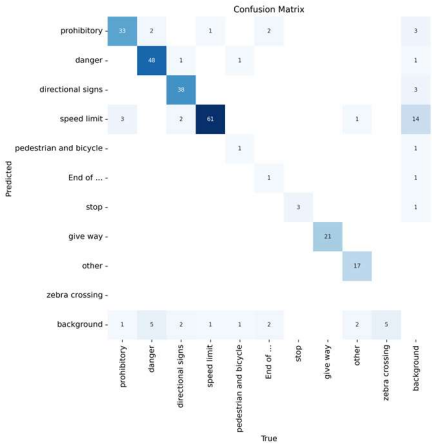
YOLOv3



YOLOv5m

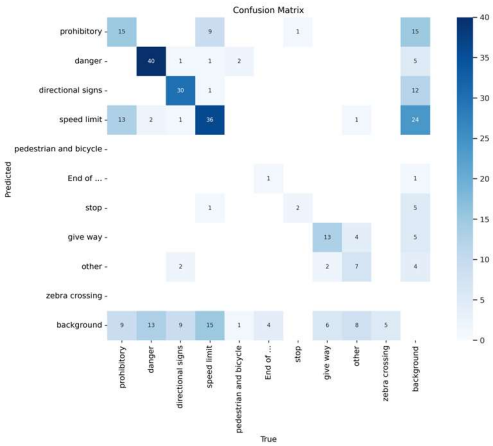


YOLOv8m

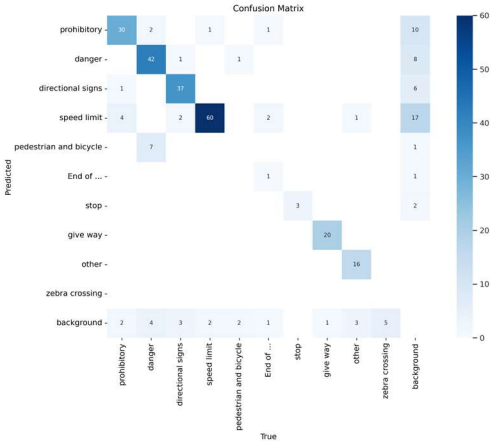


7.2.2 YOLOv5n YOLOv5m YOLOv5x

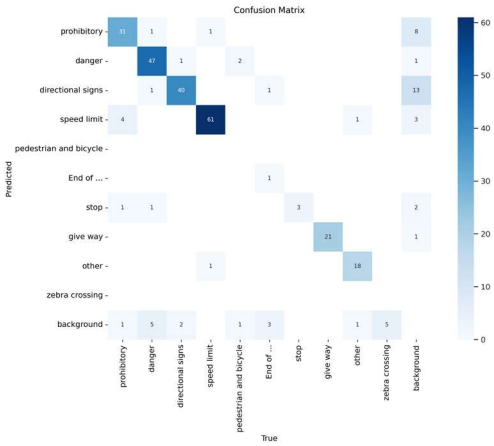
YOLOv5n



YOLOv5m

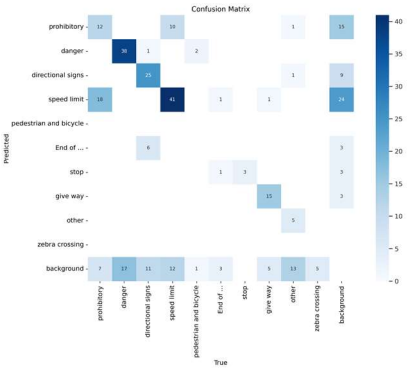


YOLOv5x

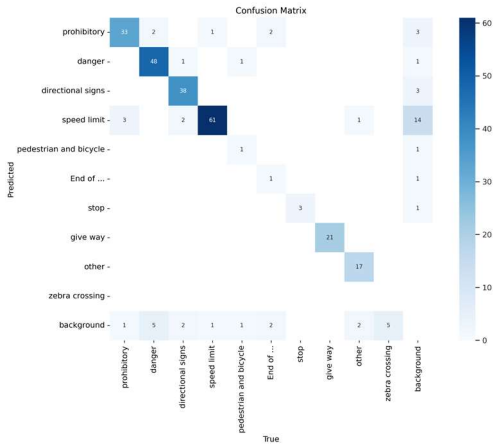


7.2.3 YOLOv8n YOLOv8m YOLOv8x

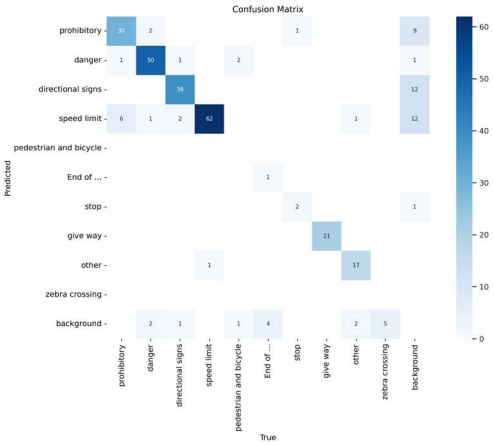
YOLOv8n



YOLOv8m



YOLOv8x



8. CONCLUSIONS

This study offers an in-depth exploration of the YOLO architecture and its variants, delineating their respective strengths and weaknesses. We can conclude that:

1. Newer the version of YOLO exhibit superior performance across various metrics:
    - Faster Training phase;
    - Faster inference time;
    - Diminished model dimensions (MB), comparing different YOLO versions (e.g., v3-v5m-v8m), leading to more efficient GPU memory utilization;
    - Improved accuracy metrics such as Mean Average Precision (MaP), Average Precision (AP), etc;
    - Capability to achieve comparable or even better performance with smaller datasets
2. The addition of an augmentation dataset significantly enhances model performance, particularly with smaller models like "nano."
3. YOLOv3 can perform a little bit better on the localitazion task for smaller objects. We can reconduct it to a possible smaller default threshold of showing predictions.
4. Adding epochs (our GPU constraint was up to 30 epochs with the bought colab GPU) can significantly improve the performance of the model, specifically in the smaller models.
5. Changing hyperparameters can ultimately make the performance worse, so we left them unchanged. (They are already “optimezed” by Ultralytics)
6. The zebra crossing class is one of the most uncorrect localization ones, but there is a catch. After we trained the various models and used them to make predictions, we realized that we incorrectly calculated the bounding box of this class. We put in the labels not the real width and height but half of them. Noneless, the larger models can still predict this class accurately.

9. FUTURE WORKS

As we conclude this study, several avenues emerge for future exploration of the comparison between the models:

- We have incorporated all the “speed limit” traffic signs in one big class. A future work could be to divide this class into multiple sub-classes, to predict the exact speed velocity of each traffic sign. It should be an “easy” work but tedious, due the the changing of each label of each image. The same thing could be done with direction signs.
- Next approach to achieve a better MaP would be to perfect the data augmentation algorithm including both rotation and flipping. The flipping approach would be applied only for those images that don't include traffic signs that with mirroring would change the semantic (class); however in the other classes the labels must also be changed, accordingly to a function that invert the Xc and Yc of the bounding box (such as “1-Xc” and “1-Yc”, because the values are already normalized) . The rotation instead it’s a little bit more tricky, we need to remember the rotation factor and apply it to the labels, probably increasing the width and height of the bounding box and slightly changing the center of it.

BIBLIOGRAPHY

[1] https://www.kaggle.com/datasets/valentynsichkar/traffic-signs-dataset-in-yolo-format
[2] https://docs.ultralytics.com/models/yolov3/
[3] Nepal, U.; Eslamiat, H. Comparing YOLOv3, YOLOv4 and YOLOv5 for Autonomous Landing Spot Detection in Faulty UAVs. *Sensors* **2022**, *22*, 464. https://doi.org/10.3390/s22020464
[4] https://docs.ultralytics.com/models/yolov5/
[5] https://docs.ultralytics.com/models/yolov8/
[6] https://github.com/ultralytics/ultralytics
[7] https://link.springer.com/article/10.1007/s44196-023-00302-w