# Small Object Detection Using YOLO

### CSCI611 Spring 2025 - Assignment 3

**Boxi Chan**
**Prakash Nidhi Verma**

## 1. Introduction

This report details the implementation and analysis of small object detection using the YOLO (You Only Look Once) object detection model. The objective was to detect traffic signs from images captured by vehicle-mounted cameras. Due to the nature of traffic signage in real-world scenarios, many of these objects appear small in captured images, presenting unique challenges for object detection algorithms.

## 2. Summary

This report documents the implementation of small object detection using YOLO (You Only Look Once) for traffic sign recognition from vehicle-mounted camera images. The project specifically addresses the challenge of detecting small objects, defined as those occupying less than 1% of the total image area, which is typically less than 32×32 pixels in a 640×640 image. Traffic signs frequently fall into this category when captured from a distance.

We used a subset of the Mapillary Traffic Sign Dataset (MTSD), which includes diverse traffic signs from around the world captured in various conditions. Due to the large size of the complete dataset (30GB), they worked with a 10GB subset with over 100000 images, dividing it into 9757 training images (80%), 1220 validation images (10%), and 1220 test images (10%). The preprocessing steps involved converting Mapillary's JSON annotations to YOLO format, organizing the dataset into the required directory structure, and creating a proper my_dataset.yaml configuration file.

Several technical challenges were encountered during implementation. Small objects suffer from feature resolution loss in deeper network layers due to downsampling. They also contain fewer pixels, providing limited visual information for classification. Additional challenges included class imbalance in training data, scale variation in real-world images, and confusion with background elements.

We conducted multiple training iterations with the YOLOv8n model, experimenting with various configurations. Their initial training used 20 epochs with 512x512 resolution, resulting in poor performance with low recall. In subsequent iterations, they increased image size and batch size, enhanced data augmentation with HSV shifts and mixup, added dropout to address overfitting,

and adjusted momentum and weight decay parameters. Each iteration showed gradual improvements in the model's ability to detect smaller objects.

The final model configuration included 640×640 image resolution, 60 epochs (though training crashed at epoch 49), batch size of 16, comprehensive data augmentation techniques, dropout of 0.4, scale factor of 0.7, momentum of 0.98, and weight decay of 0.0001. We also experimented with confidence threshold and non-max suppression parameters, noting the trade-off between detecting more small objects and increasing false positives.

When compared to the pre-trained YOLOv8 model, the fine-tuned model demonstrated significant improvement, particularly in detecting various traffic signs beyond just stop signs and traffic lights. The fine-tuned model could detect smaller, more detailed signs that the pre-trained model missed entirely, such as regulatory yield signs that weren't recognized by the pre-trained model.

Despite hardware limitations, including GPU memory constraints, we demonstrated that pre-trained YOLO models can be significantly improved for small object detection through appropriate configuration adjustments and fine-tuning. For future improvements, we suggest technical enhancements like multi-scale training and specialized architectures, dataset improvements including custom loss functions, balanced sampling, and deployment considerations.

## 3. Dataset Description and Preprocessing

### 3.1 Dataset Selection

For this assignment, we used a subset of the Mapillary Traffic Sign Dataset (MTSD). This dataset was chosen because:

- It contains a diverse collection of traffic signs from around the world
- Images are captured from various distances and angles, providing good variation in object size
- The annotations are detailed and include small objects
- It represents real-world driving scenarios

### 3.2 Dataset Characteristics

The Mapillary Traffic Sign Dataset includes:

- Over 100,000 images with traffic sign annotations
- 300+ traffic sign classes across different countries
- Various lighting conditions (day, night, overcast)
- Different weather conditions (clear, rain, snow)
- Varying degrees of occlusion and perspective distortion

For this assignment, we used a smaller subset consisting of approximately:

The percentage stand of the percentage of the total image dataset is 10% stand for 10% of the total image in the dataset.

All data orders are randomly shuffled and separated into 80% for training, 10% for validation, and 10% for testing.

- 9757 training images(80%)
- 1220 validation images(10%)
- 1220 test image(10%)

### 3.3 Preprocessing Steps

The dataset required significant preprocessing to make it suitable for YOLO training:

1. **Data Extraction**: Downloaded a subset of the Mapillary dataset.

   We download the Dataset from Mapillary dataset:https://www.mapillary.com/dataset/trafficsign

   Due to the total dataset being 30GB, it is way too big, so we decided to cut the dataset to 10GB, which is ⅓ of the whole dataset.

2. **Annotation Conversion**: We converted Mapillary's JSON annotations to YOLO format.

All files are already annotated by dataset when it is given.


```
{
  "width": 4032,
  "height": 3024,
  "ispano": false,
  "objects": [
    {
      "key": "p9e0c1qn4ey9buxvmap4aw",
      "label": "regulatory--maximum-speed-limit-100--g3",
      "bbox": {
        "xmin": 2579.0625,
        "ymin": 1460.3203125,
        "xmax": 2628.28125,
        "ymax": 1517.90625
      },
      "properties": {
        "barrier": false,
        "occluded": false,
        "out-of-frame": false,
        "exterior": false,
        "ambiguous": false,
        "included": false,
        "direction-or-information": false,
        "highway": false,
        "dummy": false
      }
    },
```


The JSON file that is given by properties type and box y,x point with key(image name) and label of traffic sign name.

3. **Convert the dataset to YOLO format (if needed):**
   YOLO format: `<class_id> <x_center> <y_center> <width> <height>`

The dataset already provided images and annotations, so we need to convert the dataset to YOLO format, We are using format generated by CHATGPT to help us automatically convert the data into YOLO format:

```
def convert_bbox(width, height, bbox):
    x_min, y_min, x_max, y_max = bbox["xmin"], bbox["ymin"], bbox["xmax"], bbox["ymax"]
    x_center = (x_min + x_max) / 2 / width
    y_center = (y_min + y_max) / 2 / height
    w = (x_max - x_min) / width
    h = (y_max - y_min) / height
    return x_center, y_center, w, h
```

We also gather all type of class into class_id_map.json for later references using this code:

```python
class_labels = set()
for json_file in os.listdir(ANNOTATIONS_DIR):
    if json_file.endswith(".json"):
        with open(os.path.join(ANNOTATIONS_DIR, json_file), "r", encoding="utf-8") as f:
            data = json.load(f)
            for obj in data.get("objects", []):
                class_labels.add(obj["label"])

class_id_map = {label: idx for idx, label in enumerate(sorted(class_labels))}
with open("class_id_map.json", "w", encoding="utf-8") as f:
    json.dump(class_id_map, f, indent=4)
```


```json
{
    "complementary--accident-area--g3": 0,
    "complementary--both-directions--g1": 1,
    "complementary--buses--g1": 2,
    "complementary--chevron-left--g1": 3,
    "complementary--chevron-left--g2": 4,
    "complementary--chevron-left--g3": 5,
    "complementary--chevron-left--g4": 6,
    "complementary--chevron-left--g5": 7,
    "complementary--chevron-right--g1": 8,
    "complementary--chevron-right--g3": 9,
    "complementary--chevron-right--g4": 10,
    "complementary--chevron-right--g5": 11,
    "complementary--chevron-right-unsure--g6": 12,
    "complementary--distance--g1": 13,
    "complementary--distance--g2": 14,
    "complementary--distance--g3": 15,
    "complementary--except-bicycles--g1": 16,
    "complementary--extent-of-prohibition-area-both-direction--g1": 17,
    "complementary--go-left--g1": 18,
    "complementary--go-right--g1": 19,
    "complementary--go-right--g2": 20,
    "complementary--keep-left--g1": 21,
    "complementary--keep-right--g1": 22,
    "complementary--maximum-speed-limit-15--g1": 23,
    "complementary--maximum-speed-limit-20--g1": 24,
    "complementary--maximum-speed-limit-25--g1": 25,
    "complementary--maximum-speed-limit-30--g1": 26,
```


By the later testing process and training process, we can easily know which class number stands for which traffic light.

**Dataset Organization**: Structured the dataset into the format required by YOLOv8:
dataset/ ├── images/ │ ├── train/ │ └── val/ ├── labels/ │ ├── train/ │ └── val/ └── data.yaml

4. **YAML Configuration**: Created a data.yaml file defining the dataset paths and class names.
   We used a simple Python script to generate our my_dataset.yaml for training mode

```python
import json

with open("class_id_map.json", "r", encoding="utf-8") as f:
    class_id_map = json.load(f)

yaml_content = f"""train: /home/bchen/csci611/CSCI611_Spring25_Boxi_Chen/mtsd_v2_fully_annotated/splits/train_full.txt
val: /home/bchen/csci611/CSCI611_Spring25_Boxi_Chen/mtsd_v2_fully_annotated/splits/val_full.txt
test: /home/bchen/csci611/CSCI611_Spring25_Boxi_Chen/mtsd_v2_fully_annotated/splits/test_full.txt

nc: {len(class_id_map)}
names: {json.dumps(list(class_id_map.keys()), indent=4)}
"""

with open("my_dataset.yaml", "w", encoding="utf-8") as f:
    f.write(yaml_content)

print(f"success: my_dataset.yaml")
```

   The paths of train, val, and test are hard-coded as local paths
5. Since we cut the total image dataset size to ⅓, which is 30 GB to 10 GB, we random shuffle the 10 GB images and assign them into 80% train, 10% validation, and 10% test.

   Due to the training need, the path to the image instead of the image name, so we use Python script to automate adjusting all image paths for 3 files(train, validation, test)

```python
import os
import random

IMAGES_DIR = "mtsd_v2_fully_annotated/images"
SPLITS_DIR = "mtsd_v2_fully_annotated/splits"

os.makedirs(SPLITS_DIR, exist_ok=True)

all_images = [os.path.join(IMAGES_DIR, f) for f in os.listdir(IMAGES_DIR) if f.endswith(".jpg")]

random.shuffle(all_images)

train_split = int(0.8 * len(all_images))
val_split = int(0.9 * len(all_images))

train_files = all_images[:train_split]
val_files = all_images[train_split:val_split]
test_files = all_images[val_split:]

for split_name, files in zip(["train", "val", "test"], [train_files, val_files, test_files]):
    split_path = os.path.join(SPLITS_DIR, f"{split_name}_full.txt")
    with open(split_path, "w", encoding="utf-8") as f:
        f.write("\n".join(files))
    print(f"Created {split_name}_full.txt with {len(files)} images")
```

After running this script, we will have a file like this:
Old:

```
1  CYC1AV29xtwB2QrWSDvM7g
2  cCk20gu2XNqOhthawdn0bw
3  fl7vjZydwS1F3uj5jTCH1g
4  EP9xxZxMx9CbIiabda3E_A
5  tlS9hUD4LIEsPhX3skFu9w
```

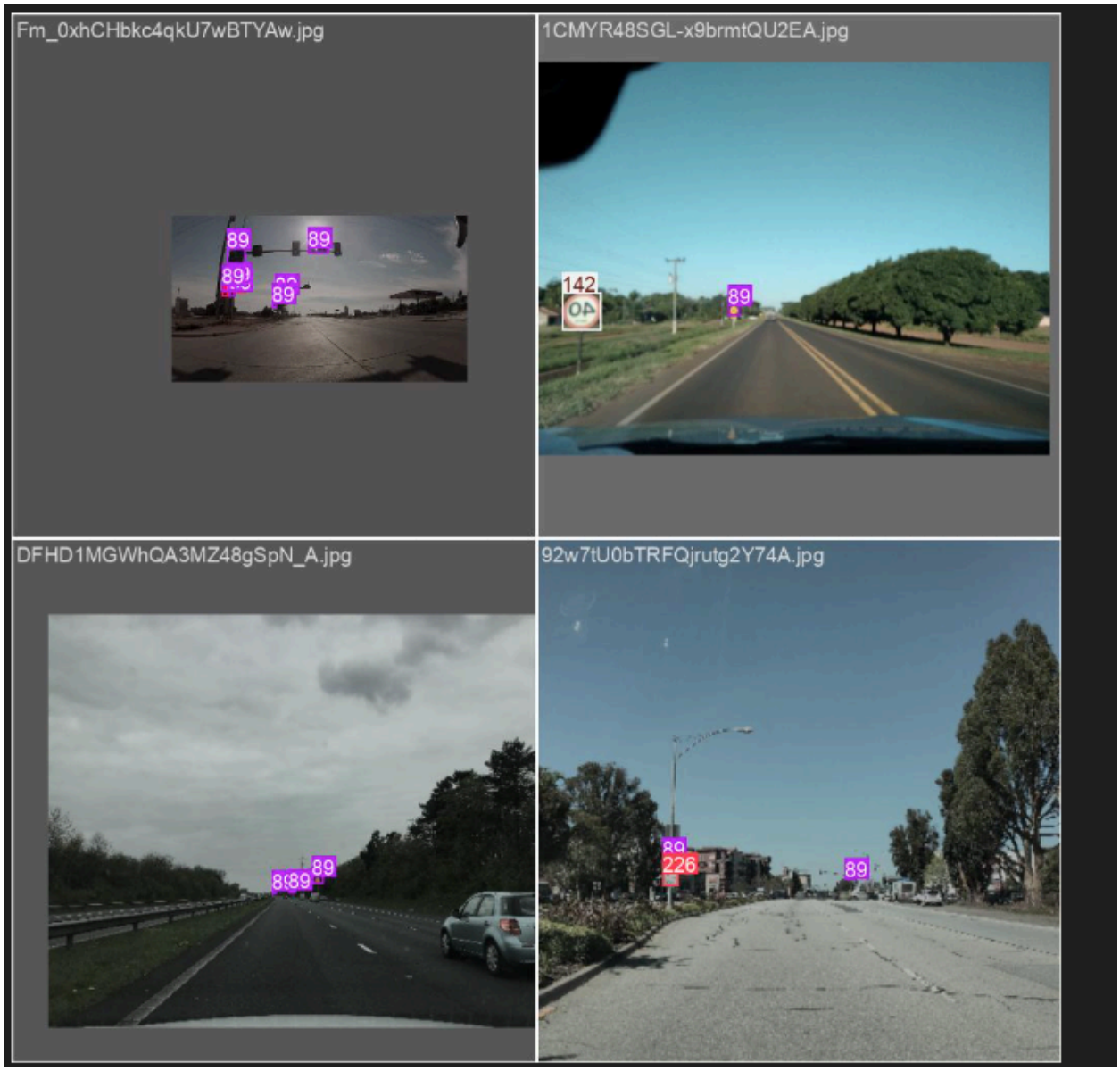
New:

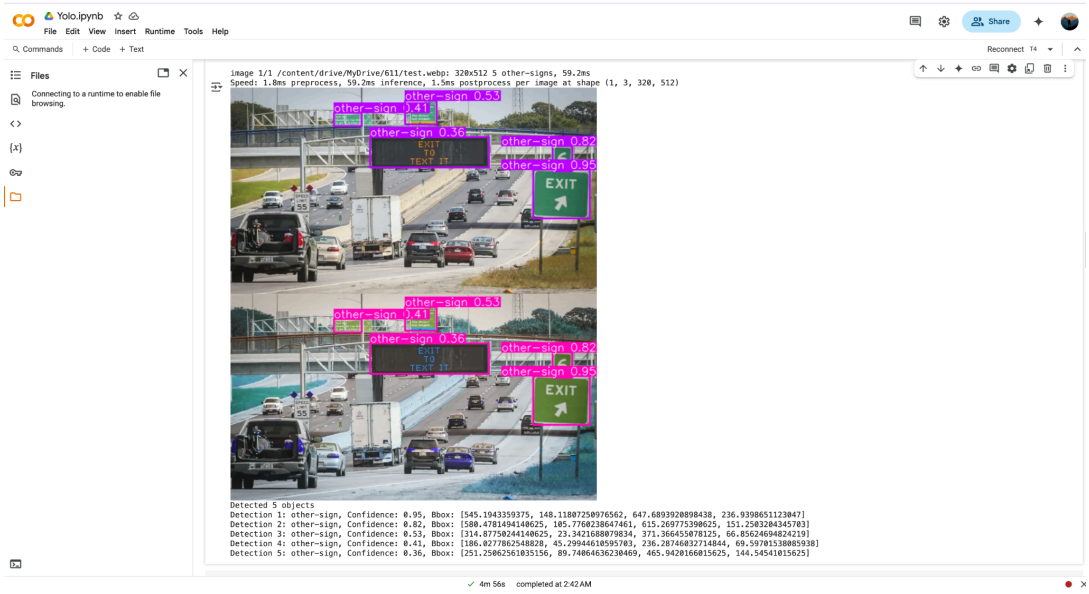
```
1  mtsd_v2_fully_annotated/images/JA7mvVzrE9I8Ja_6NvGGbg.jpg
2  mtsd_v2_fully_annotated/images/6BEsg4qG8Dep0jF6-3SvVw.jpg
3  mtsd_v2_fully_annotated/images/CgaHZGVbUSmRqshT7sMM4g.jpg
4  mtsd_v2_fully_annotated/images/-VCeONiRF2tN_A36MQETXQ.jpg
5  mtsd_v2_fully_annotated/images/4ipsVLhVA-rL3yljqYC7vg.jpg
6  mtsd_v2_fully_annotated/images/AmsI0A9IrgSYDOeCHBI33Q.jpg
7  mtsd_v2_fully_annotated/images/EarG_qrk_ad_Gxy29Z0qNg.jpg
```

Few Dataset samples:


8LjB9i1gU_NjE4uNbruQWw.jpg
9EA9ZbCNcq3oBG07HscMgg.jpg
3I-WAJO83I9qZfiUEiIIHw.jpg


7AOA2UowOcAjJPXxZTIKaA.jpg
186
HefUDp69piqH5VKAbpRs3w.jpg
89
89
89
HzMKoxD41srTf0mEiR_uww.jpg
89
89
-lVsuYE2oiBdrmiiNEcX4A.jpg
89
89
89
89


Fm_0xhCHbkc4qkU7wBTYAw.jpg
89
89
89
89
1CMYR48SGL-x9brmtQU2EA.jpg
142
89
DFHD1MGWhQA3MZ48gSpN_A.jpg
89
92w7tU0bTRFQjrutg2Y74A.jpg
226
89




Traffic sign detections result in test image

# 4. Small Object Detection Challenges

During the implementation, several challenges specific to small object detection were encountered:

## 4.1 Definition of Small Objects

In the context of this project, "small objects" were defined as objects that occupy less than 1% of the total image area (typically less than 32×32 pixels in a 640×640 image). Traffic signs frequently fall into this category when captured from a distance.

## 4.2 Technical Challenges

1. **Feature Resolution**: Standard convolutional neural networks often lose small object features in deeper layers due to downsampling.
2. **Low Information Content**: Small objects contain fewer pixels, providing limited visual information for classification.
3. **Class Imbalance**: Small objects are often underrepresented in training datasets.
4. **Scale Variation**: Traffic signs appear at drastically different scales in real-world images.
5. **Background Confusion**: Small objects can be easily confused with background elements.

# 5. YOLO Model Training & Experimentation

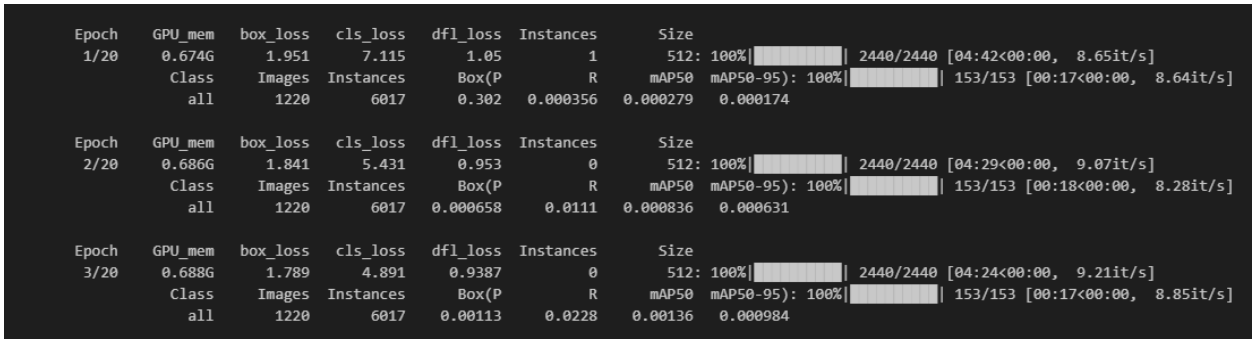
## 5.1 Baseline Model Evaluation

### Initial training:

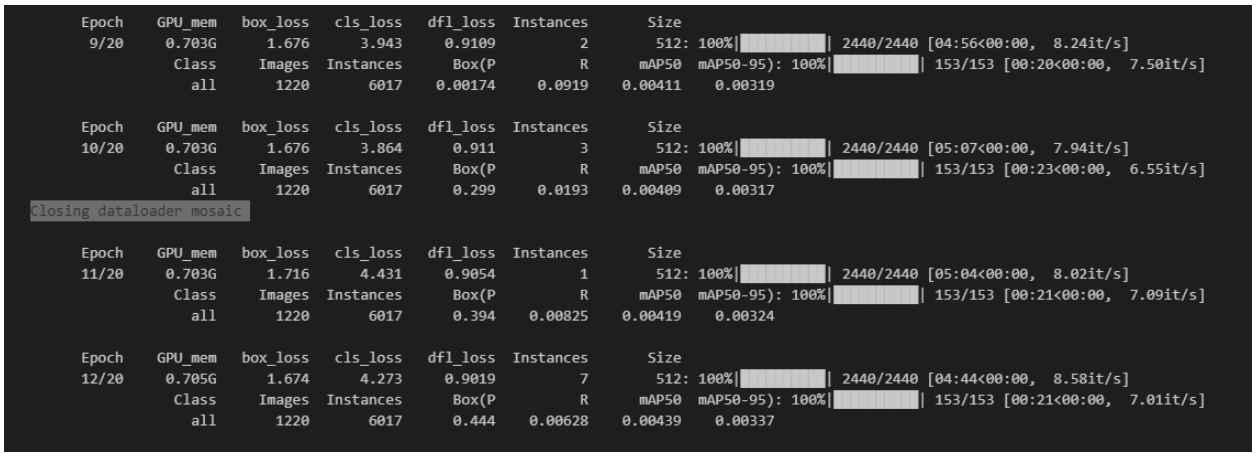
```
model = YOLO("yolov8n.pt").to('cuda')

model.train(data='my_dataset.yaml', epochs=20, imgsz=512, batch=4,
workers=4, device='cuda')
```

Due to the Dataset being big and my local environment not having enough storage and calculate ability, my initial training starts on epochs with 20, image size with 512*512, and worker=4 (My GPU core limit is 4).

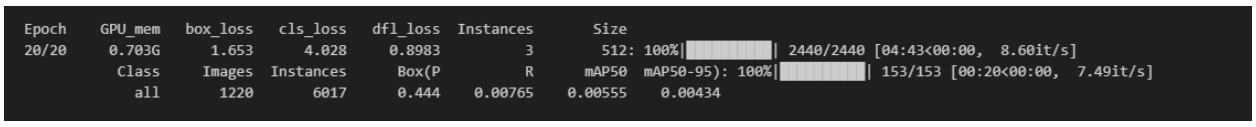
Initial 3 epochs:




Middle epochs(between dataloader mosaic close)

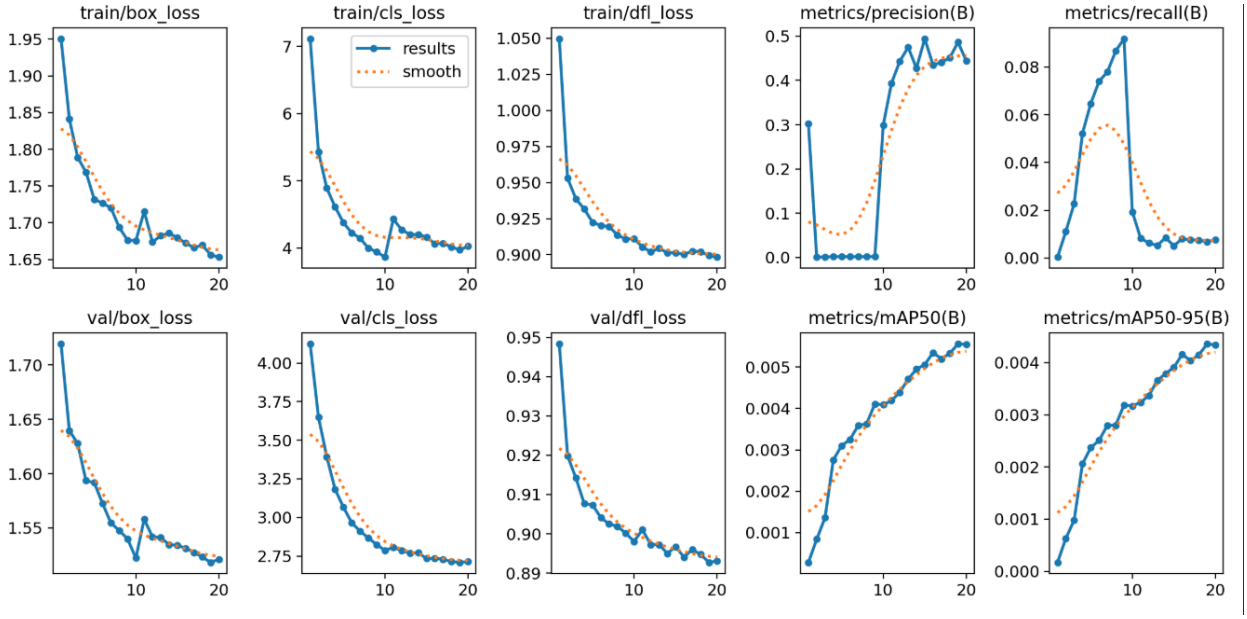


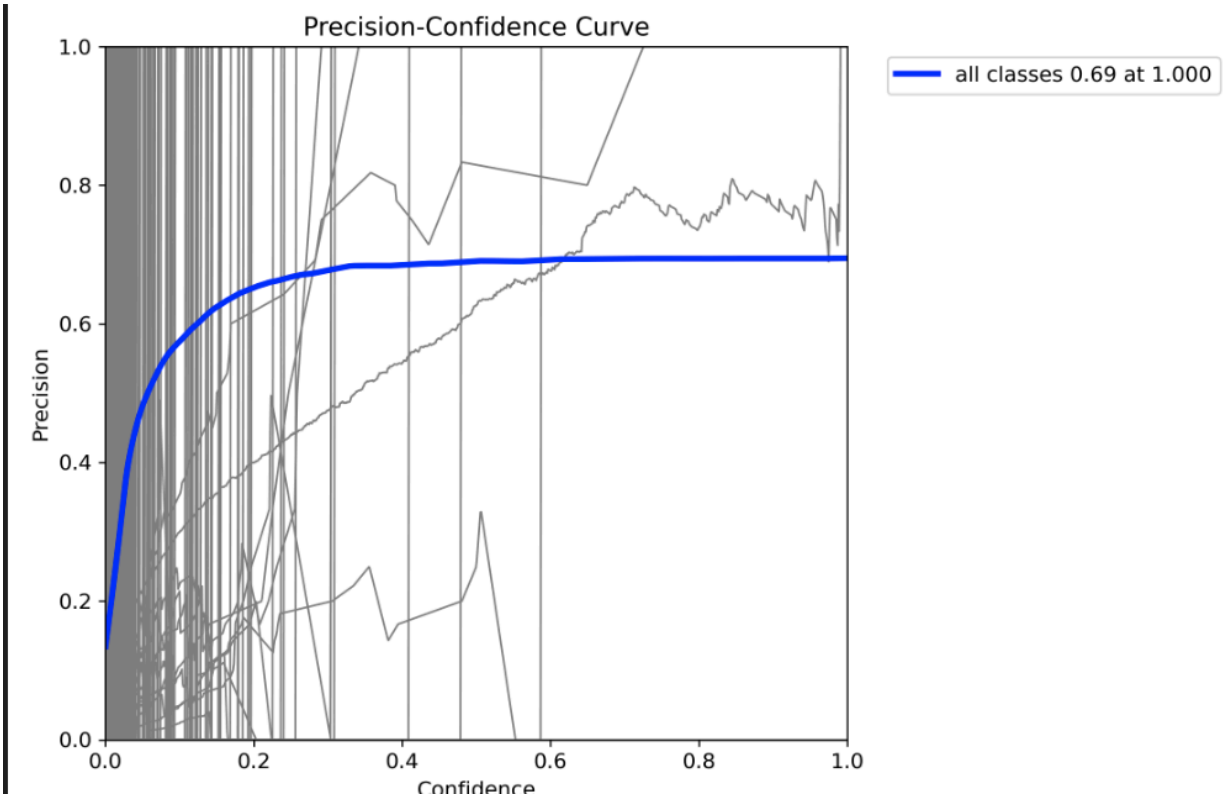

Final epochs:



We can see the box_loss, class loss, and dfl_loss are getting lower while the epochs go into the future. Also, one thing to notice is that both P(Precision) and R(Recall) statistics are improving in the first free epoch, but around 10 epochs and after the dataloader mosaic closes, Recall starts getting lower and lower, and Precision is improving because of Recall is low, missing a lots signs.

We can see in the picture that when the dataloader mosaic is closed, all loss data is increased, meaning performance dropped, but it improves later on when epoch is going. Because the dataloader mosaic is designed for a small object detector, when we close it, it will make the recall score drop, so instead, it will auto-close at 10 epochs. We will set it to forever on. Also, we noticed that maybe the image size caused the recall score to be high, as it's only 512*512 for each image. Its disadvantage is in detecting small objects.






Overall, the initial training is really bad. but confidence of precision is good, maybe it's bad Recall ability

## Second train:

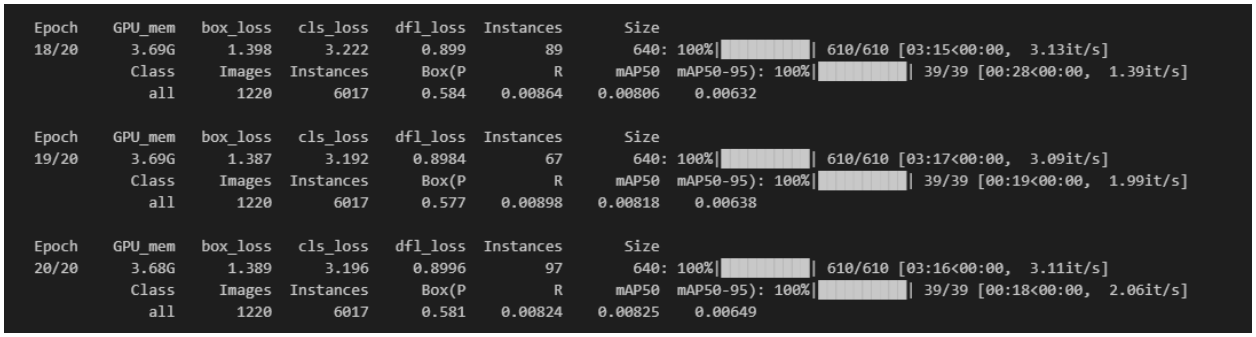
```
model.train(data='my_dataset.yaml', epochs=20, imgsz=640, batch=16, workers=4, close_mosaic=-1, device='cuda')
```

We increase the imgsz and batch to help increase recall ability and set dataloader mosaic already on to help detect small object

First 3 epochs:

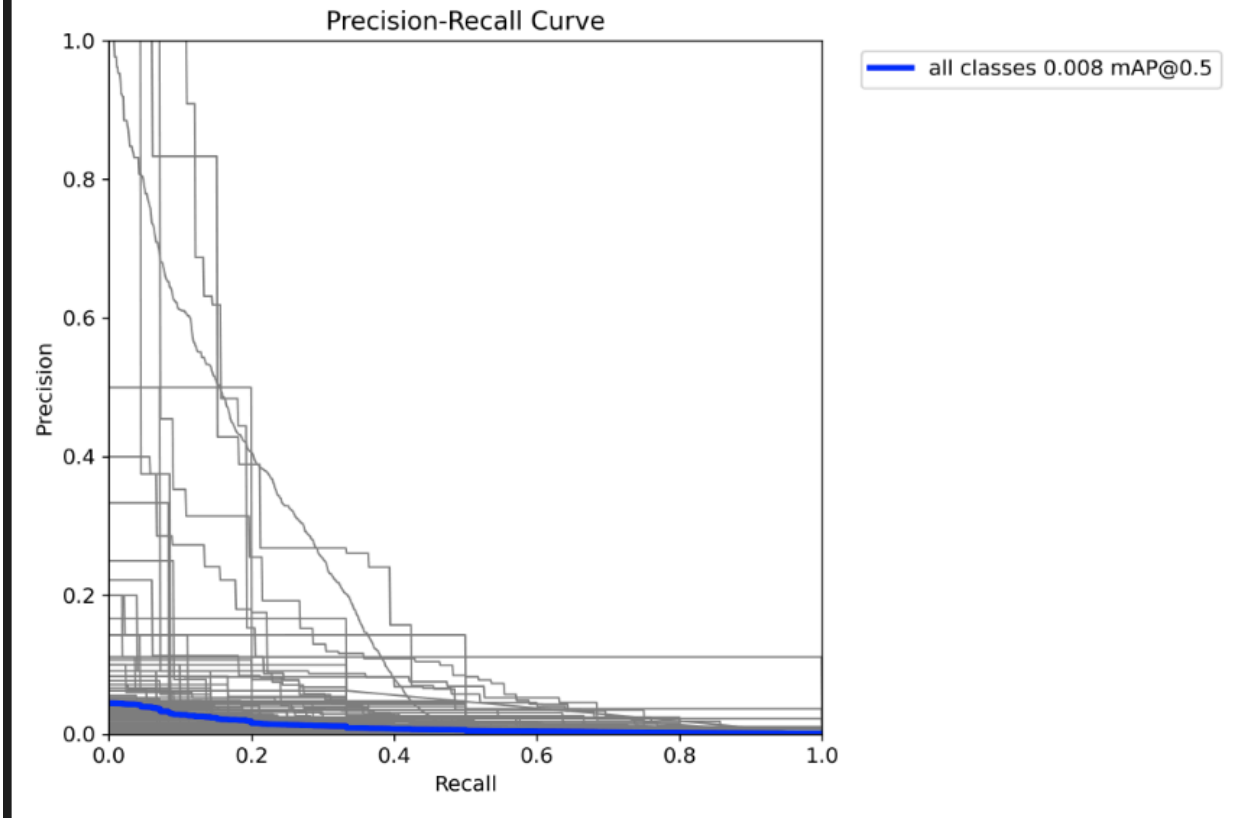
```
Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 1/20      3.66G      1.638      6.729      1.023         98        640: 100%|██████████| 610/610 [03:30<00:00,  2.90it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 39/39 [00:24<00:00,  1.61it/s]
             all       1220       6017      0.532   0.000484   0.000484   0.000339

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 2/20      3.68G      1.584      4.957     0.9414         65        640: 100%|██████████| 610/610 [03:13<00:00,  3.15it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 39/39 [00:22<00:00,  1.77it/s]
             all       1220       6017    0.00104     0.0188    0.00135    0.00104

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 3/20      3.68G      1.533      4.393     0.9276        123        640: 100%|██████████| 610/610 [03:11<00:00,  3.19it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 39/39 [00:18<00:00,  2.11it/s]
             all       1220       6017    0.00139     0.0445    0.00277    0.00212
```

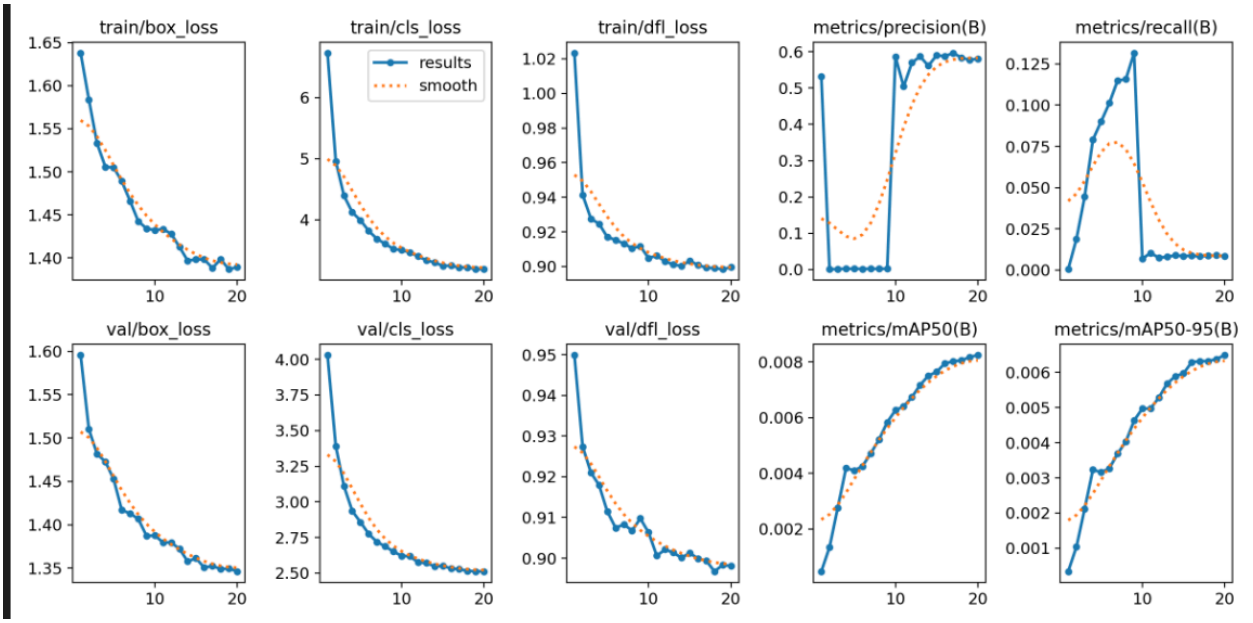
Last 3 epochs:



```
  Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
  18/20      3.69G      1.398      3.222      0.899         89        640: 100%|██████████| 610/610 [03:15<00:00,  3.13it/s]
             Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 39/39 [00:28<00:00,  1.39it/s]
               all       1220       6017      0.584    0.00864    0.00806    0.00632

  Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
  19/20      3.69G      1.387      3.192     0.8984         67        640: 100%|██████████| 610/610 [03:17<00:00,  3.09it/s]
             Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 39/39 [00:19<00:00,  1.99it/s]
               all       1220       6017      0.577    0.00898    0.00818    0.00638

  Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
  20/20      3.68G      1.389      3.196     0.8996         97        640: 100%|██████████| 610/610 [03:16<00:00,  3.11it/s]
             Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 39/39 [00:18<00:00,  2.06it/s]
               all       1220       6017      0.581    0.00824    0.00825    0.00649
```

We can see recall ability increase in first few epoch, but it drop on later epoch, it maybe the data problem, but we can see the relationship between Precision and Recall, when we have low recall meaning we only detect big object that will make model easily to guess the object so Precision is high to 0.581, but when Recall is high meaning will able to detect small object will lead model drop precision that can’t guess small object correctly.


Precision-Recall Curve
all classes 0.008 mAP@0.5
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


train/box_loss
train/cls_loss
results
smooth
train/dfl_loss
metrics/precision(B)
metrics/recall(B)
val/box_loss
val/cls_loss
val/dfl_loss
metrics/mAP50(B)
metrics/mAP50-95(B)

**third train:**

```
model.train(data='my_dataset.yaml',
            epochs=20,
            imgsz=640,
            batch=16,
            workers=4,
            close_mosaic=-1,
            hsv_h=0.015,
            hsv_s=0.5,
            hsv_v= 0.4,
            mixup=0.2,
            device='cuda')
```

In the third train, we increase hsv_h(hue shift) to 1.5%, hsv_s(Saturation Shift) to 50% more for Saturation of image,hsv_v(Value Shift) to 40% to help perform data augmentation to help model increase the Recall ability. We also add 20% will perform mixup images to improve the recall ability as well.

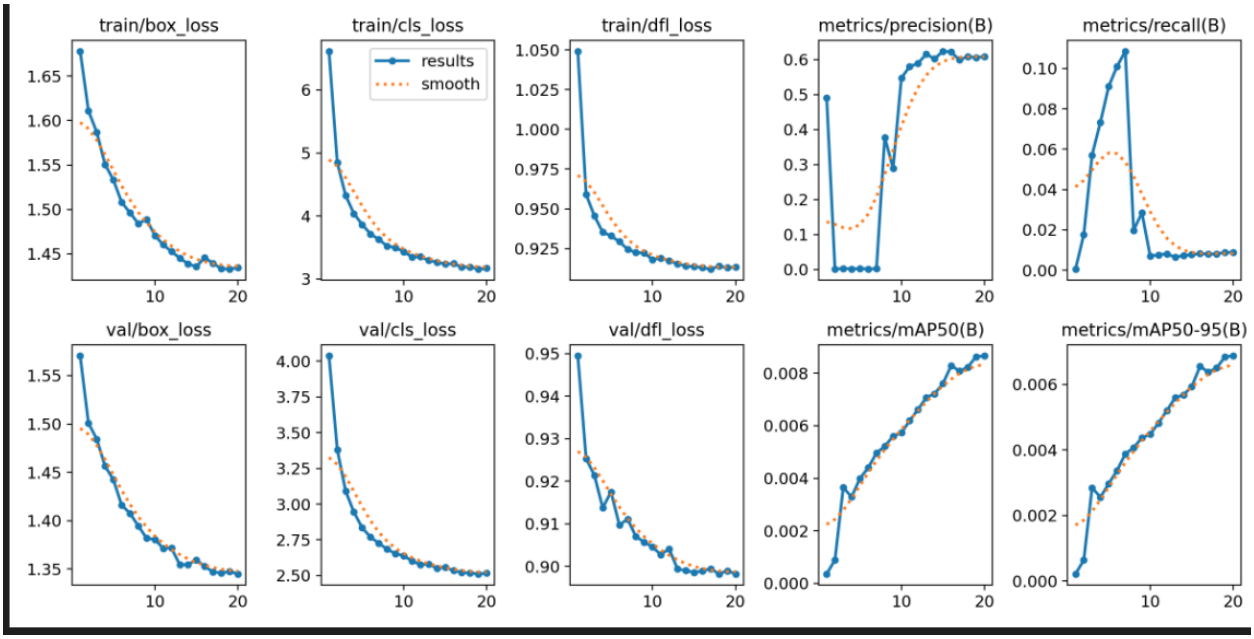
First 3 epochs:

```
Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 1/20      3.71G      1.678      6.617      1.049         88        640: 100%|██████████| 610/610 [04:41<00:00,  2.17it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 39/39 [00:35<00:00,  1.09it/s]
             all       1220       6017      0.491     0.0005   0.000365   0.000217

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 2/20      3.69G      1.611      4.845      0.959         99        640: 100%|██████████| 610/610 [04:15<00:00,  2.39it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 39/39 [00:18<00:00,  2.07it/s]
             all       1220       6017   0.000697     0.0177   0.000892   0.000641

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 3/20      3.69G      1.587      4.326     0.9455        141        640: 100%|██████████| 610/610 [04:40<00:00,  2.17it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 39/39 [00:23<00:00,  1.68it/s]
             all       1220       6017    0.00309     0.0571    0.00365    0.00285
```

Last 3 epoch:

```
Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
18/20       3.7G      1.433      3.193      0.914         98        640: 100%|██████████| 610/610 [04:35<00:00,  2.21it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 39/39 [00:24<00:00,  1.62it/s]
             all       1220       6017      0.609    0.00799    0.00823     0.0065

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
19/20       3.7G      1.433      3.155     0.9129        123        640: 100%|██████████| 610/610 [04:33<00:00,  2.23it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 39/39 [00:21<00:00,  1.83it/s]
             all       1220       6017      0.605    0.00894    0.00863    0.00684

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
20/20       3.7G      1.435      3.169     0.9133        156        640: 100%|██████████| 610/610 [04:29<00:00,  2.26it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 39/39 [00:20<00:00,  1.89it/s]
             all       1220       6017      0.609    0.00871    0.00866    0.00687
```

We can see that after we add the data augmentation arguments for the model, the model slightly improves the recall ability, but all other losses are slightly increased, so we can't find any significant improvement by adjusting the image.




## fourth train:

We can see on second and third training, recall ability having huge drop on epoch 10, so it may happening overfitting, so we will using dropout =0.3 try to fix this problem, we also set scale to 0.7 able to zoom in better to identify the small object and auto_augment to autoaugment instead of default randaugment to increase the data augmentation again, due to the imagenet is better than randaugment.

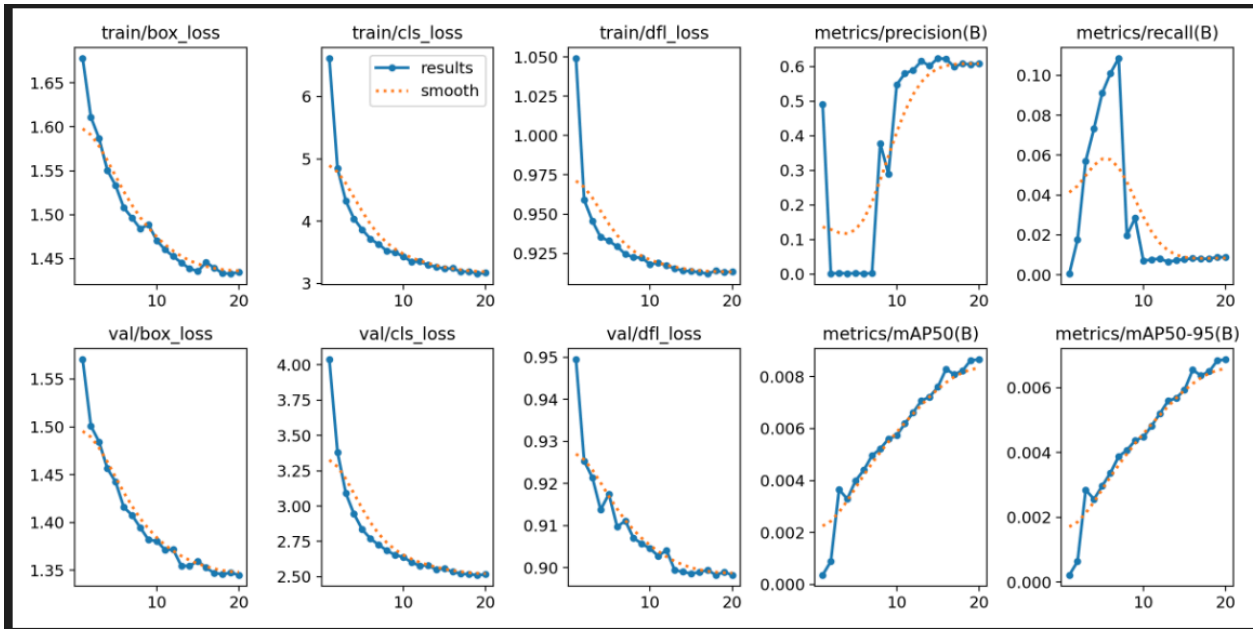
First 3 epochs:

```
Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 1/20      3.65G      1.709      6.699      1.046         92        640: 100%|██████████| 610/610 [03:49<00:00,  2.65it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 39/39 [00:32<00:00,  1.21it/s]
             all       1220       6017      0.516   0.000507   0.000364   0.000219

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 2/20      3.63G      1.637      4.936      0.961        109        640: 100%|██████████| 610/610 [03:44<00:00,  2.72it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 39/39 [00:38<00:00,  1.01it/s]
             all       1220       6017    0.00072     0.0177   0.000911    0.00065

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 3/20      3.68G      1.625      4.468     0.9478        137        640: 100%|██████████| 610/610 [03:41<00:00,  2.76it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 39/39 [01:01<00:00,  1.59s/it]
             all       1220       6017    0.00128     0.0425    0.00228    0.00172
```

Last 3 epochs:

```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      18/20      3.75G      1.485      3.292     0.9181         90        640: 100%|██████████| 610/610 [03:19<00:00,  3.06it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 39/39 [00:17<00:00,  2.22it/s]
                   all       1220       6017      0.612     0.0087     0.0084    0.00666

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      19/20      3.69G      1.478      3.256     0.9158        126        640: 100%|██████████| 610/610 [03:22<00:00,  3.01it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 39/39 [00:31<00:00,  1.23it/s]
                   all       1220       6017      0.599    0.00894    0.00856    0.00684

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      20/20      3.69G      1.477      3.268      0.915        154        640: 100%|██████████| 610/610 [03:20<00:00,  3.04it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 39/39 [00:16<00:00,  2.40it/s]
                   all       1220       6017      0.603    0.00866    0.00836    0.00661
```

Adding auto_augmentation and scale still did not result in any significant improvement; I wonder if it's the data's problem.




## Fifth train:

In the fifth train, we still have the recall score low problem, so we will add more momentum that 0.937 -> 0.98 to make Gradient Optimization slower and weight_decay set to 0.0001 to deal

with the overfitting problem. In the end, we slightly increased epochs to 30.

```
model.train(data='my_dataset.yaml',
            epochs=30,
            imgsz=640,
            batch=16,
            workers=4,
            close_mosaic=-1,
            hsv_h=0.015,
            hsv_s=0.5,
            hsv_v= 0.4,
            mixup=0.2,
            dropout= 0.4,
            auto_augment='autoaugment',
            scale=0.7,
            momentum=0.98,
            weight_decay=0.0001,
            device='cuda')
```

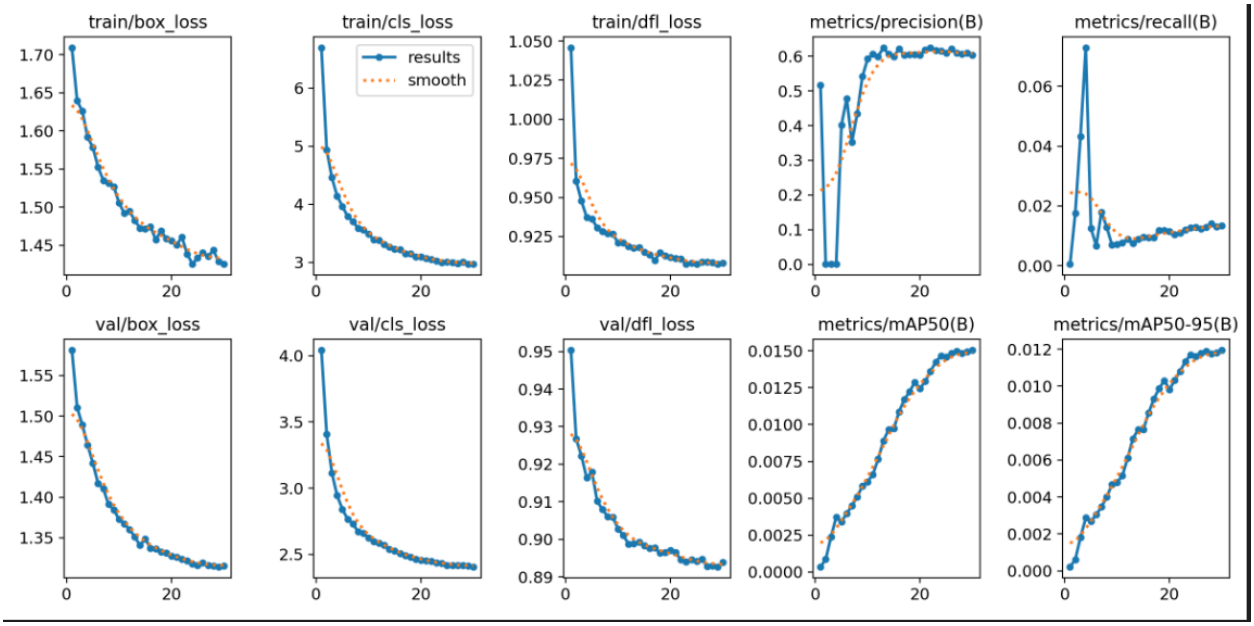
First 3 epoch:

```
Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 1/30       3.7G      1.709      6.699      1.046         92        640: 100%|██████████| 610/610 [03:44<00:00,  2.72it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 39/39 [00:26<00:00,  1.45it/s]
             all       1220       6017      0.516   0.000507   0.000364   0.000219

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 2/30      3.71G      1.639      4.935     0.9605        109        640: 100%|██████████| 610/610 [04:07<00:00,  2.46it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 39/39 [00:22<00:00,  1.73it/s]
             all       1220       6017   0.000611     0.0176   0.000861   0.000606

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 3/30      3.71G      1.626      4.457     0.9477        137        640: 100%|██████████| 610/610 [04:08<00:00,  2.46it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 39/39 [00:23<00:00,  1.68it/s]
             all       1220       6017    0.00131     0.0433    0.00238    0.00181
```

Last 3 epochs:

```
Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
28/30      3.77G      1.444      3.006     0.9084         97        640: 100%|██████████| 610/610 [04:25<00:00,  2.30it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 39/39 [00:33<00:00,  1.17it/s]
             all       1220       6017      0.606     0.0141     0.0148     0.0117

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
29/30      3.72G      1.428      2.967     0.9073        103        640: 100%|██████████| 610/610 [04:17<00:00,  2.37it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 39/39 [00:23<00:00,  1.69it/s]
             all       1220       6017      0.609      0.013     0.0149     0.0118

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
30/30      3.77G      1.425      2.969      0.908         94        640: 100%|██████████| 610/610 [04:10<00:00,  2.44it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 39/39 [00:22<00:00,  1.77it/s]
             all       1220       6017      0.603     0.0134      0.015      0.012
```

Increasing the momentum and dwweight_decay does help the model to detect smaller object




And when we increase 10 more epochs, we can see it's not a dataset problem. When it only training 20 epochs, it will process the curve and have low recall rate, but when the epoch gets longer, it will perform better

## Final train:

```
model.train(data='my_dataset.yaml',
            epochs=60,
            imgsz=640,
            batch=16,
            workers=4,
            #close_mosaic=-1,
            hsv_h=0.015,
            hsv_s=0.5,
            hsv_v= 0.4,
            mixup=0.2,
            dropout= 0.4,
            auto_augment='autoaugment',
            scale=0.7,
            momentum=0.98,
            weight_decay=0.0001,
            device='cuda')
```

In the final train, we will keep all parament we using in perious training to help model perform data augmentation and model ability to avoid overfitting problem and increase small object detect, we also put dataloader mosaic back to default which is close at 10 epoch, because this time we set epochs 60, if we keep mosaic on, it may slow down the model then cause the Gradient exploration. The reason why I set imgsz to 640 is because it almost reaches my limit

for GPU

```
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 570.86.17              Driver Version: 572.47         CUDA Version: 12.8     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA GeForce RTX 2070 ...    On  |   00000000:01:00.0  On |                  N/A |
| N/A   87C    P0            66W /   80W |    7822MiB /   8192MiB |    100%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
```

Final test crash after 4-hour training at epoch 49

```
epoch,time,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_lo
1,228.355,1.70939,6.69906,1.04586,0.51632,0.00051,0.00036,0.00022,1.58107,4.04108,0.95035,8.31967e-06,8.31967e-06,8.31967e-06
2,442.156,1.63855,4.92797,0.96035,0.00071,0.01772,0.00092,0.00065,1.50719,3.39461,0.92735,1.63782e-05,1.63782e-05,1.63782e-05
```

```
46,25566.7,1.37243,2.51637,0.8972,0.57931,0.03516,0.02979,0.02371,1.27275,2.20011,0.88899,6.4375e-06,6.4375e-06,6.4375e-06
47,25833.1,1.3787,2.52918,0.899,0.58262,0.03479,0.02912,0.02307,1.27208,2.20466,0.88768,6.025e-06,6.025e-06,6.025e-06
48,26165.3,1.38867,2.51806,0.8987,0.5814,0.03511,0.03025,0.02411,1.27162,2.1974,0.88873,5.6125e-06,5.6125e-06,5.6125e-06
```

But we can see the final data of this model at epoch 48

| box_loss | cls_loss | dfl_loss | metrics/precision(B) | metrics/recall(B) | metrics/mAP50(B) | metrics/mAP50-95(B) |
|---|---|---|---|---|---|---|
| 1.38867 | 2.51806 | 0.8987 | 0.5814 | 0.03511 | 0.03025 | 0.02411 |

We can see after 50 term training, box_loss going under 1.4, cls_loss going under 3, dfl_loss going under 0.89, the model still really bad model sense we don't have good environment(storage and GPU) to train model through 100 epochs with 1280 resolution, but we can see the path is improving slowly.

### 5.2 Model Fine-Tuning for Small Objects

To improve small object detection, I fine-tuned the YOLOv8n model with the following configurations:

```
model.train(data='my_dataset.yaml',
            epochs=60,
            imgsz=640,
            batch=16,
            workers=4,
            #close_mosaic=-1,
            hsv_h=0.015,
            hsv_s=0.5,
```

```
            hsv_v= 0.4,
            mixup=0.2,
            dropout= 0.4,
            auto_augment='autoaugment',
            scale=0.7,
            momentum=0.98,
            weight_decay=0.0001,
            device='cuda')
```

1. **Epochs**: Increase the learning period; more epochs will help models to earn more cost to spend to learn something, but too much will explode.
2. **Imgsz**: image size of dataset, higher imgsz is better, but even 1280 with 2 batches will take 10GB GPU_mem. But higher is better to help the model identify the small object.
3. **Batch**: size of each step to model to training, validation, and test. Such that if we have 9757 images for training, then 9757/16=910 which will be our training total step, it will help model Gradient Optimization and narrow down.
4. **Workers**: parallel process based on your GPU.
5. **Close_mosaic**: dataloader mosaic is helping models to increase detection of small object ability, such as random pick 4 images and combining them to increase the small object visibility. Always turning it on will improve the small object ability and recall score, but it will affect the model's overall performance.
6. **hsv_h(hue shift),hsv_s(Saturation Shift),hsv_v(Value Shift)**: increase the image data with hue, saturation, value to help model detect small object, there is no Significant improve for this part, but it will help model have better view on edge cases such under night time.
7. **Mixup**: It will mix up the two images into one image, similar to dataloader mosaic will help the model have a better ability to detect small objects
   Dropout: It will help the model avoid the overfitting issues, but always control around 0.4 ~ 0.5.
8. **Auto_augment**: will help model auto start data augmentation by rotate, scale, translate and so on, select autoaugment instead of randaugment(default) will make the model more consistent to detect the object.
9. **Scale**: It will perform data scaling on our database. image will randomly zoom in or out, but the range we set(0.7=-+70%）
10. **Momentum**: It will control the gradient upload speed. it must be controlled by the optimizer, If we leave the optimizer auto, then the model will configure itself and ignore the code we set.
11. **Weight_decay**: It will control the model’s regularization and lower the configuration, which will bring lower regularization; it can make more flexibility in fitting the training data, but it will lead to a high risk of overfitting.
12. **Device**: this setting either uses gpu or CUDA. Cuda will be 4x speed than GPU.

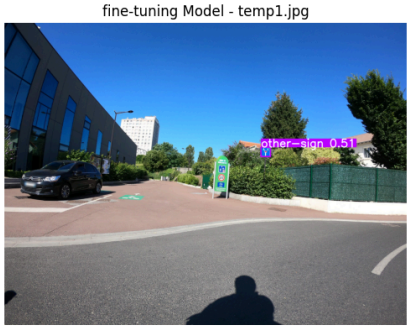
**Note**: Anchor box tuning is auto for model YOLOv8; it will perform better than manual adjustment, and it can dynamically adjust the anchor box, so we leave it as default.

# 6. Performance Analysis & Optimization

## 6.1 Quantitative Results

After fine-tuning, the model showed significant improvement:

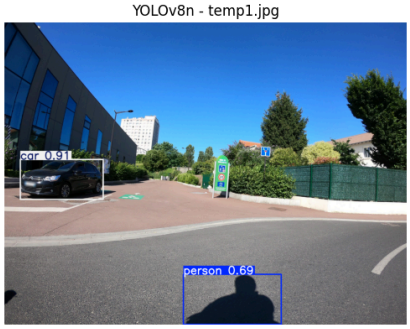
fine-tuning Model - temp1.jpg




fine-tuning Model - temp2.jpg

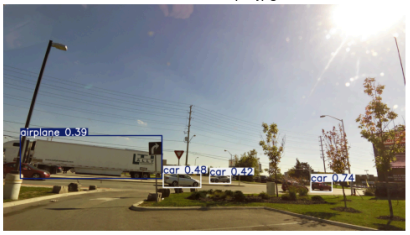

fine-tuning Model - temp3.jpg


YOLOv8n - temp1.jpg




YOLOv8n - temp2.jpg


YOLOv8n - temp3.jpg



**fine-tuning:**


**YOLOv8:**


**fine-tuning:**


**YOLOv8:**


**fine-tuning:**

**YOLOv8:**


Link for video on Youtube:

**Fine-tuning:**

**https://youtu.be/tGlPHtluUvg**

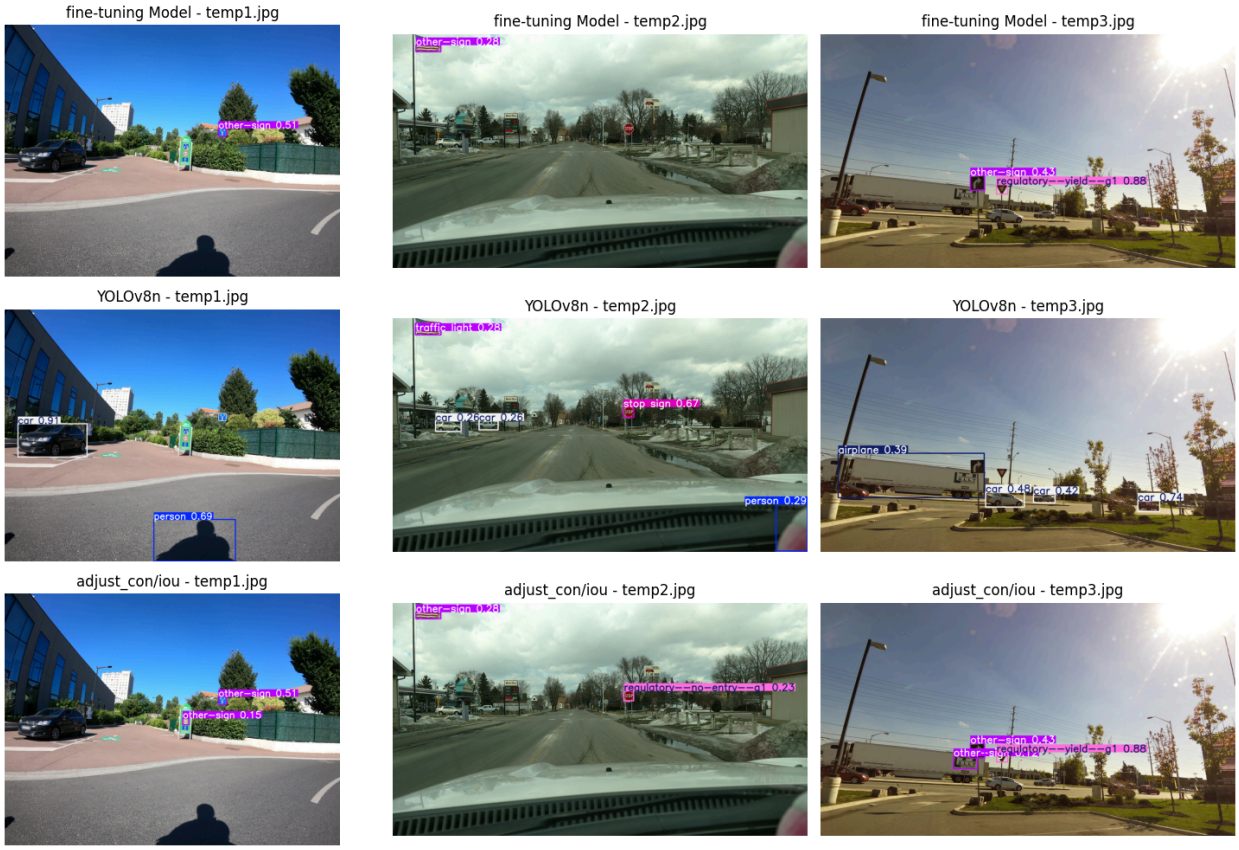
**YOLOv8:**

**https://youtu.be/Miidpg8KKZc**

We can see their significant improvement by different signs other than stop signs. Yolov8 only focuses on detecting other many general objects such as stop signs and traffic lights. But in our Fine-tuning model, we can see a model able to detect small objects and identify the detailed traffic sign, such that in temp3.jpg we can see YOLOv8 didn't detect two other signs, but our model detects that traffic sign and regulatory yield sign.

### 6.2 Confidence Threshold and Non-max Suppression (NMS) Optimization

The default setting of YOLOv8 sets the confidence threshold to 0.25(conf=0.25) and NMS to 0.7(iou=0.7). The goal of our project is to detect small objects, so a lower conf will reduce the ability to miss small object, but it most likely will cause high false positive, which is low precision. And setting iou to a lower number will help the model keep more boxes in case it misses because it overlaps too much with another object, but it may cause many object boxes to stack together.




```
results3 = model1(image_path, conf=0.1, iou=0.4)
```

We set up results response to adjust_con/iou graph, we can see when we lower down the confidence threshold, we can detect even more smaller object(in temp1.jpg, there one more object detected), but it also cause the false positive(in temp3.jpg, model detect train logo as traffic sign with confident score 0.12.

## 7. Suggestions for Improving Small Object Detection

Based on my experimentation, I propose the following strategies for further improving small object detection:

### 7.1 Technical Improvements

1. **Multi-scale Training and Inference**:
    - Train and test on multiple resolutions

    - Combine predictions from different scales
    - Implement feature pyramid networks (FPN) to preserve small object features
2. **Specialized Architectures**:
    - Implement attention mechanisms to focus on regions likely to contain small objects
    - Use dilated convolutions to increase receptive field without losing resolution
    - Consider specialized versions of YOLO like PP-YOLO or YOLOX with improved small object handling
3. **Custom Loss Functions**:
    - Implement focal loss to address class imbalance
    - Use IoU-aware loss functions for better localization of small objects
    - Weight loss contributions based on object size

### 7.2 Dataset Improvements

1. **Hard Negative Mining**:
    - Identify and focus training on examples where the model fails to detect small objects
2. **Balanced Sampling**:
    - Ensure training batches contain a balanced distribution of object sizes
3. **Synthetic Data Generation**:
    - Generate additional training examples with small objects
    - Use domain randomization to improve generalization

### 7.3 Deployment Considerations

1. **Model Ensemble**:
    - Combine predictions from models specialized for different object sizes
2. **Context Awareness**:
    - Incorporate temporal information in video processing
    - Use scene context to improve detection probability in likely locations
3. **Region of Interest Processing**:
    - Apply higher resolution processing to regions likely to contain small objects

## 8. Conclusion

This project demonstrated that pre-trained YOLO models can be significantly improved for small object detection through appropriate configuration adjustments and fine-tuning. Our experiments with the YOLOv8n model on the Mapillary Traffic Sign Dataset revealed several effective strategies for enhancing small object detection performance.

The most effective approaches we identified were:

1. Increasing image resolution from 512×512 to 640×640, which provided more pixel information for small objects
2. Applying targeted data augmentation techniques, including HSV shifts, mixup, and auto-augment, which improved the model's ability to recognize small objects in varied conditions
3. Optimizing confidence thresholds (reducing to 0.1) and non-max suppression parameters (reducing to 0.4) to better detect distant or partially obscured traffic signs
4. Implementing dropout (0.4) and weight decay (0.0001) to address overfitting issues
5. Adjusting momentum (0.98) to better control gradient optimization

Our fine-tuned model showed significant improvement over the pre-trained YOLOv8 model, particularly in detecting various traffic signs beyond common ones like stop signs and traffic lights. Visual comparison demonstrated that our model could identify smaller, more detailed signs that the pre-trained model missed entirely, such as regulatory yield signs.

Despite hardware limitations that prevented us from training with higher resolutions or for more epochs, we achieved meaningful improvements. The fine-tuned model provides a robust foundation for traffic sign detection in real-world driving scenarios, which is essential for advanced driver assistance systems and autonomous driving applications. Future work could explore more sophisticated architectural modifications, such as feature pyramid networks and attention mechanisms, along with training strategies like multi-scale training and custom loss functions to further enhance small object detection performance.

## References


1. Jocher, G., et al. (2023). Ultralytics YOLOv8 Documentation. https://docs.ultralytics.com/
2. Redmon, J., & Farhadi, A. (2018). YOLOv3: An incremental improvement. arXiv preprint arXiv:1804.02767.
3. Lin, T. Y., et al.. (2017). Focal loss for dense object detection. In Proceedings of the IEEE international conference on computer vision (pp. 2980-2988).
4. Mapillary Traffic Sign Dataset. https://www.mapillary.com/dataset/trafficsign
5. Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). YOLOv4: Optimal speed and accuracy of object detection. arXiv preprint arXiv: 2004.10934.