YOLOV5: Autonomous Vehicle Object Detection

Course: Applied Computer Vision
Author: Waleed Khan
Repo:
https://github.com/Waleedprw22/Applied_CV_Project-

Introduction:

Currently, there is a lot of work going into creating an autonomous vehicle and a critical part of it is to give the machine the ability to perceive. Many applied scientists are developing and fine tuning a variety of different algorithms, including Yolo, which will be the focus of this project. The Yolov5 algorithm, known as the you only look once object detection approach, can be utilized to detect other cars and obstacles in an autonomous vehicle. Developed by Ultralytics, the pre-trained model comes in a few flavors: nano, small, medium, large and extra large. The differences between these types come from the size of the networks. For example, for yolo_s, you will have a depth of around .33 with a width of around .5 whereas for a larger network will have greater values. In addition, the larger the network, the more accurate the results become which comes at an expense of greater computational power and time. It is important to note that this project focuses on version 5. Figure 1 is provided below to provide a comparison between the mentioned flavors.

Model	size (pixels)	mAP^{val} 50-95	mAP^{val} 50	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.0	45.7	**45**	**6.3**	**0.6**	**1.9**	**4.5**
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7

Method Overview:

Knowing that the algorithm is developed and that I can utilize Ultralytic's pretraind model to train my custom data, I first focused on data collection. The data collection phase involved several images of cars and license plates. The reasoning behind this is that the autonomous vehicle will encounter vehicles on the road that come in different sizes and will be in different orientations. What certainly stays the same across every vehicle, whether it is a truck, bus or even motorcycle, is that they will have a license plate. Feeding license plate data was therefore critical to train the model with. Additional data then included images of cars from other angles to help provide the machine with more information. After data collection and running initial runs, the objective was to experiment with the model and architecture to see if I can produce more accurate results. This includes adjusting batch and stride sizes, increasing and decreasing the depth and width of the networks, changing the optimizers, etc. The second method of experimentation will be centered around fine tuning the hyper parameters to produce more optimal results. The main metrics to focus on here is mAP metric (mean average precision), recall and precision. It is also critical to define another metric, the IoU, which is the ratio of the intersection of 'detected' and actual pixels over their union. mAP .5:.95 is the average mAP of the IoUs ranging from .5 to .95 whereas mAP .5 is the average mAP value at .5. As I experiment with the model, I will be paying very close attention to how the changes I implement will impact recall, precision and mAP values.

Experiments:

I first tried to experiment with the size of the "small" network to see how that would impact the performance. Based on documentation, we have networks that are "small", "medium", "nano", "large" and extra "large" with specific depths and widths. According to Ultralytic, the larger the network, more accurate the results are. While the results become more accurate with increasing network size, the computation time increases significantly. My first goal was to focus on the small network and adjust the width and depths such that the model would not fall between "medium" and small. I noticed that as I increased the depth or width of the network, the computational time would increase considerably with little to no change in results (see figure 1 for default results)

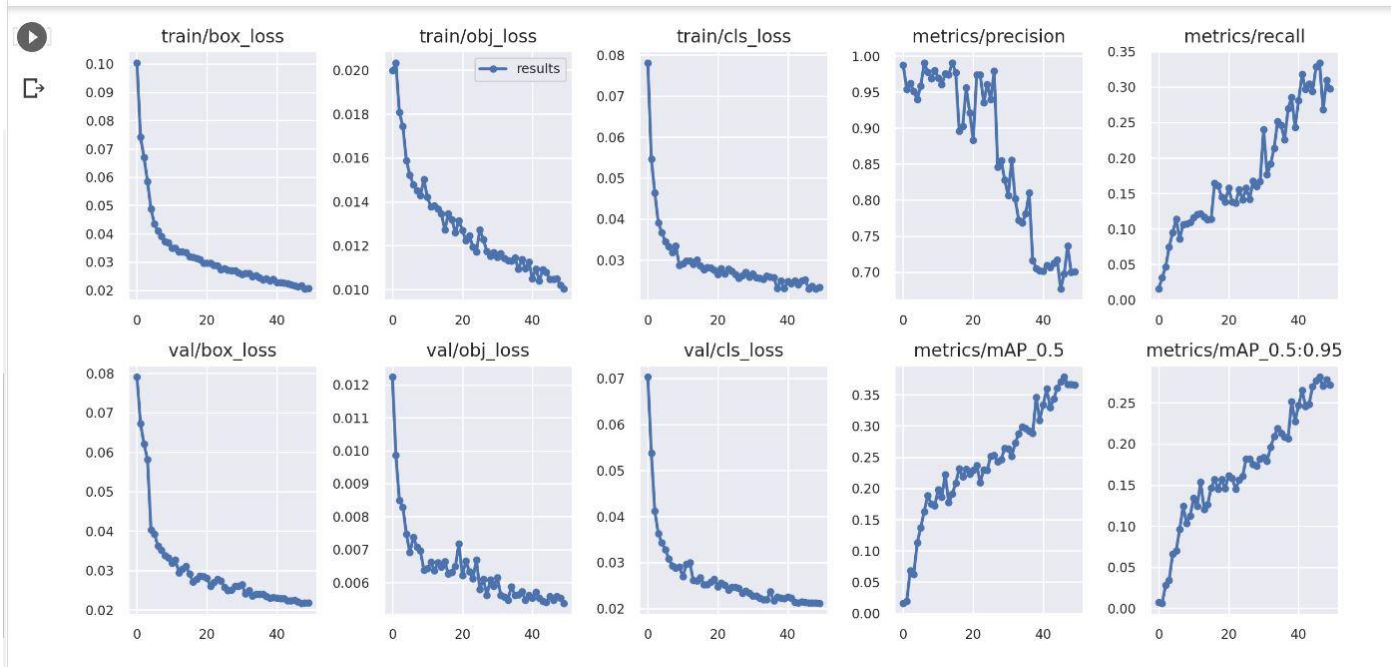

Figure 1: Default results of Yolov5s Using Custom Data Set.

Bearing in mind that the Yolo model is a pre-trained one, another set of experiments I conducted was centered around the hyper parameters. The question was, how can I better optimize the parameters for my dataset. According to Ultralytics, I can use hyper parameter evolution to optimize my parameters. The goal of this approach is to find parameters such that our fitness metric is maximized (see figure below).

```
def fitness(x):
    # Model fitness as a weighted combination of metrics
    w = [0.0, 0.0, 0.1, 0.9]  # weights for [P, R, mAP@0.5, mAP@0.5:0.95]
    return (x[:, :4] * w).sum(1)
```

Figure 2: Default Fitness model

In figure 2 above, note how majority of the weight is allocated to the mAP [.5:.95] metric. After using the default fitness function, I ran hyper parameter evolution for 5 evolutions, 10 epochs each, the model selected the parameters that gave the highest fitness score, which can be seen in the figure below. The y axis represents the fitness value whereas the horizontal axis represents the parameter that is being optimized. It is important to note that these are the optimal parameters for a fitness function that gives 90% of its weight to mAP[.5:95], and 10% to mAP[.5]. With this in mind, I expect that there may be an increase in the mAP metric but not so much with recalls or precision.

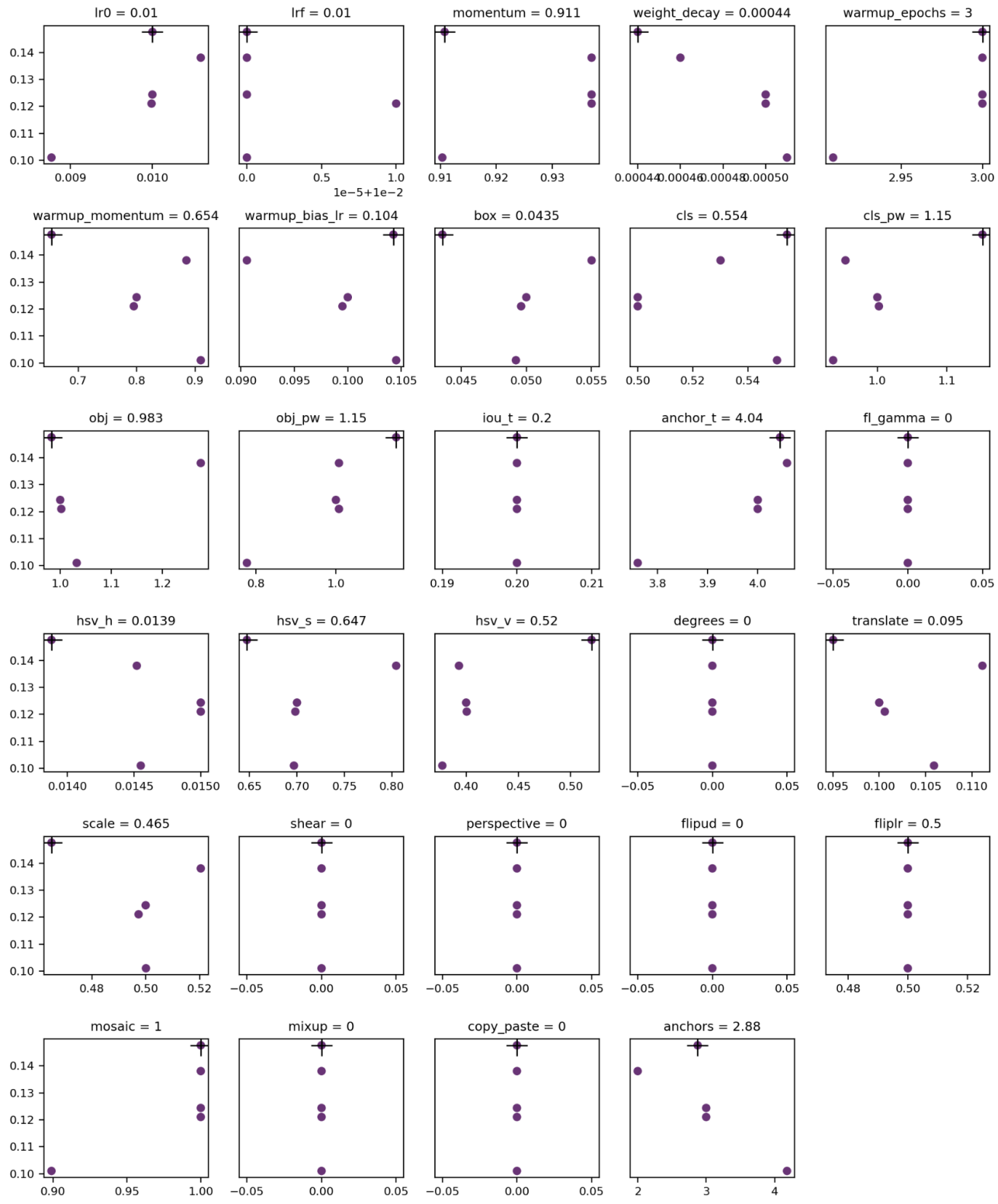

Figure 3: Results of hyper parameter evolution

After running the model with the revised parameters, we see an improvement in the results (see figure below).

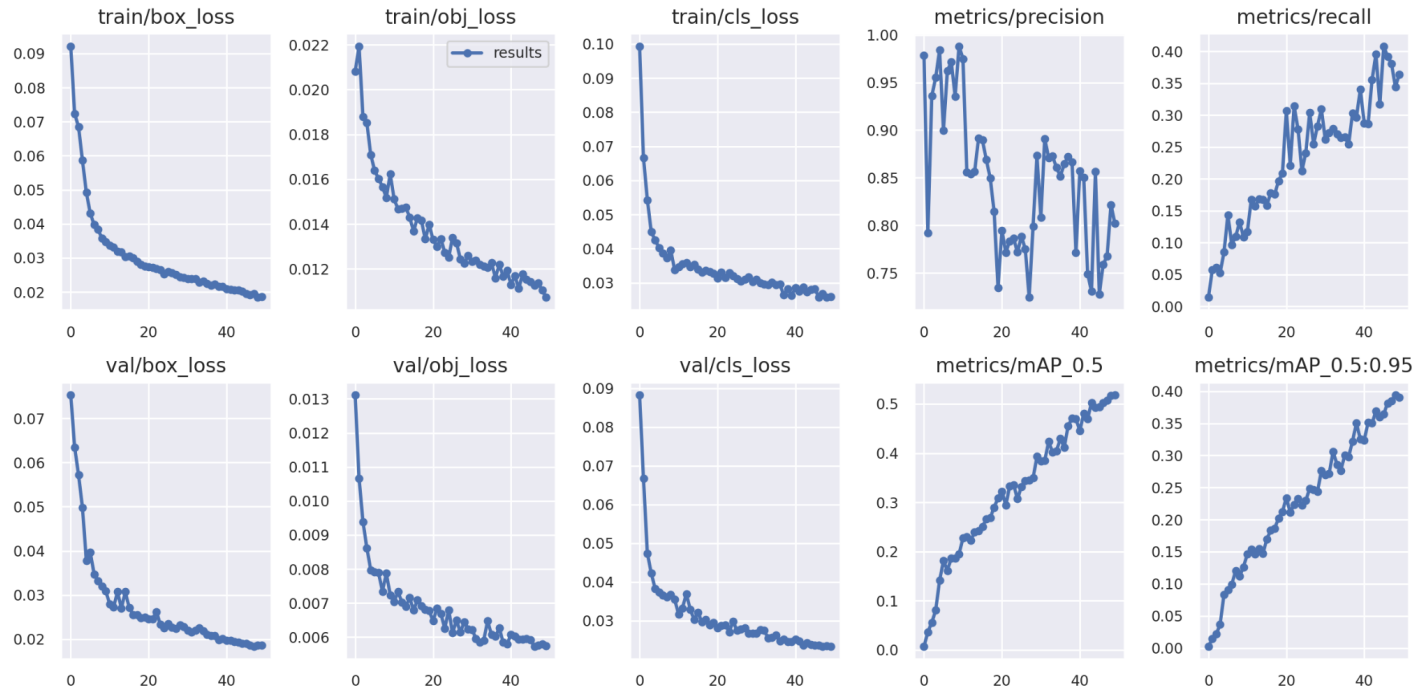

Figure 4: Results After Running Modified Evolutionary Algorithm V1

There is an increase in the precision from .725 to .8, an increase in recall from .30 to .37, an increase in mAP 0.5 from .35 to .53, and an increase in mAP_.5:.95 from .27 to .38. For the standard fitness definition and for this specific data set, the hyper parameters generated are more optimal than what they were set to initially. The next step I took was I reversed the weights for mAP .5 and mAP .5:.95 in the fitness function. When re-running the hyper parameter evolution algorithm and re-running the yolo program after redefining “fitness”, the model performed slightly worse than the default case. See figure below:

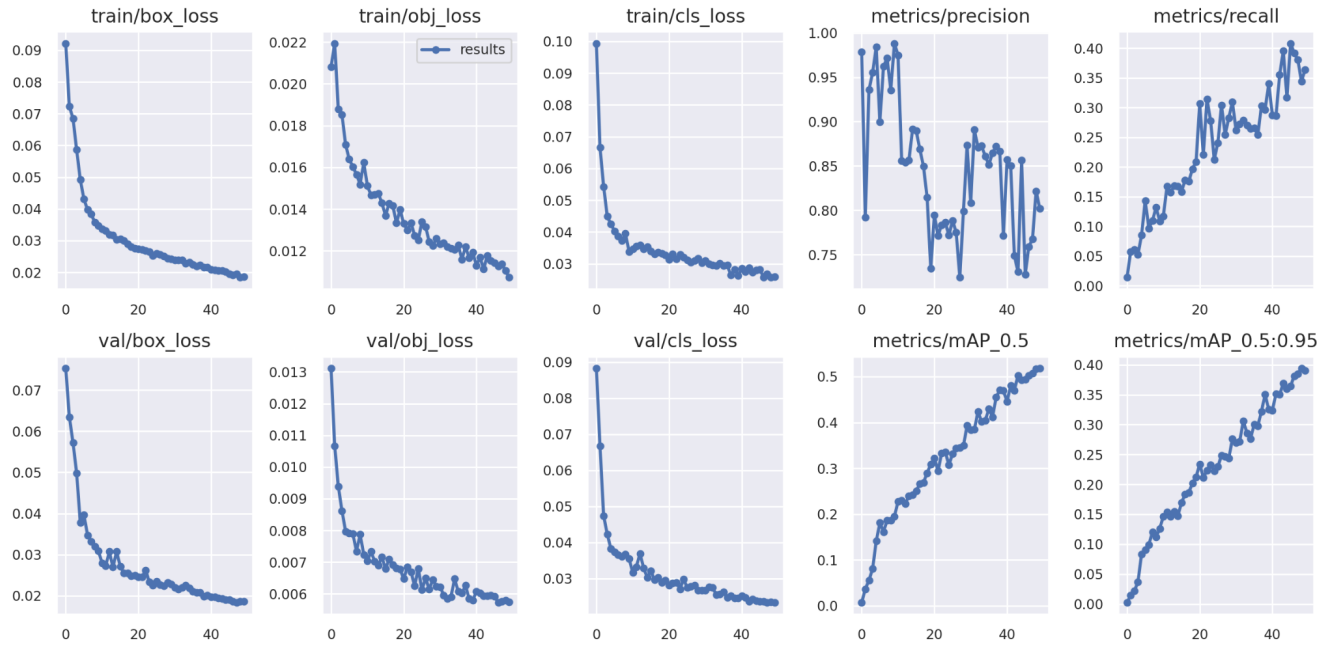

Figure 5: Results After Running Modified Evolutionary Algorithm 2

I reverted my parameters to what I had before and decided to experiment with other parts of the model. In the train file, I adjusted the IoU threshold from .65 and .6 to 0.5. I did not want to decrease the threshold any further to avoid false detections but at the same time, I do want to capture pixels that would otherwise slip. In addition, I also doubled the number of strides in the Yolo file and attempted to increase and decrease the batch size in train.py to see if that would impact the results in any way. After randomly experimenting with different parts of the python files of Yolo, I came to the conclusion that the most impactful section is evolution for loop in the train.py file. I therefore went back into the evolution section in the train.py file and changed the code by changing parent from ‘single’ to ‘weighted’. I also made the hyper parameter evolution algorithm consider the previous five results rather than just the previous two. When re-running

the model with the updated algorithm, the results improved a lot more (see figure below).

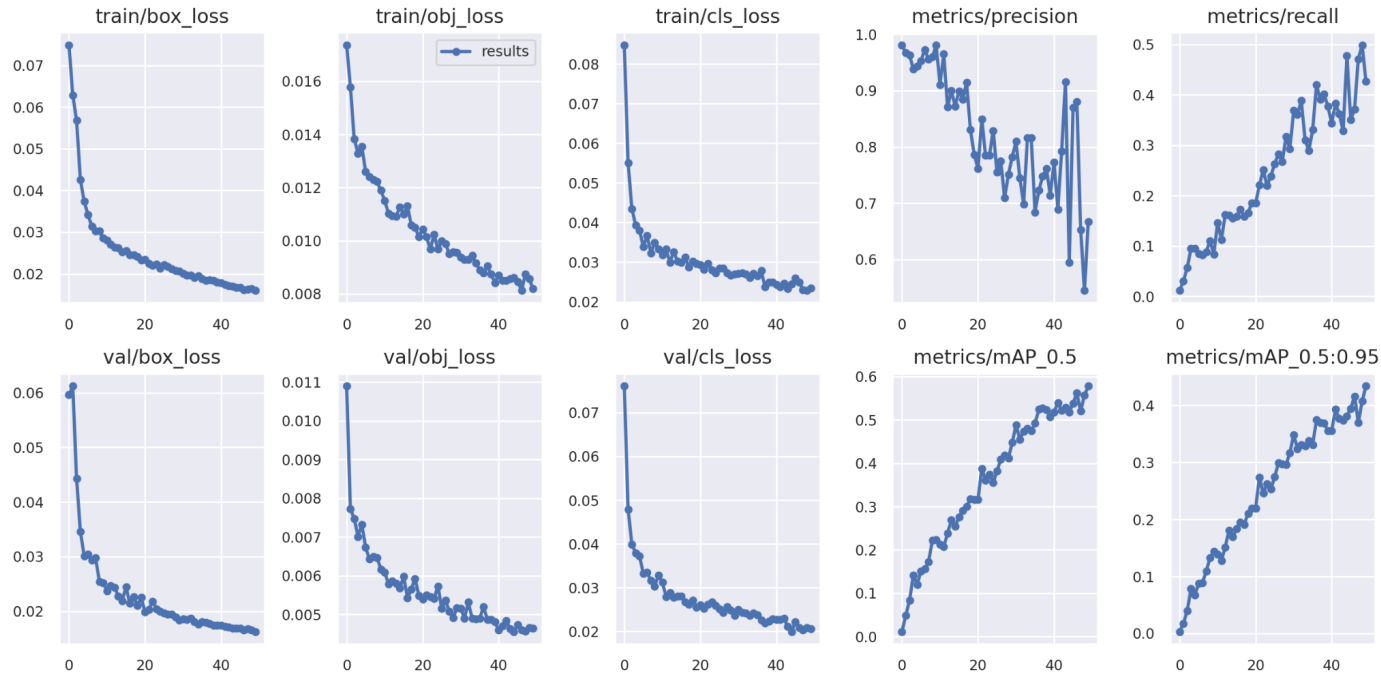

Figure 6: Results of modified evolutionary algorithm V3

The recall graph has increased to around .24, the precision has decreased to around .68, the mAP (.5) has increased to nearly .6, and the mAP(.5:.95) has increased to around .43. It is interesting to see that while the mAP and recall metrics have went up, precision has decreased. Note that precision is defined as true positives divided by the sum of true and false positives. Recall is defined as the ratio of true positives to the sum of true positives and false negatives. Perhaps the reason may be due to how the fitness function was defined for these executions. The fitness function gave a weight of .9 to mAP[.5:.95] and a weight of .1 to mAP[.5]. In other words, the fitness function gave zero weights to precision and recall, which has led to the results that we see here.

I then began to experiment with the limits of the parameters. For example, when I changed the image rotation's upper limit from 45 to 90 degrees, the model performed worse than expected (see figure below).

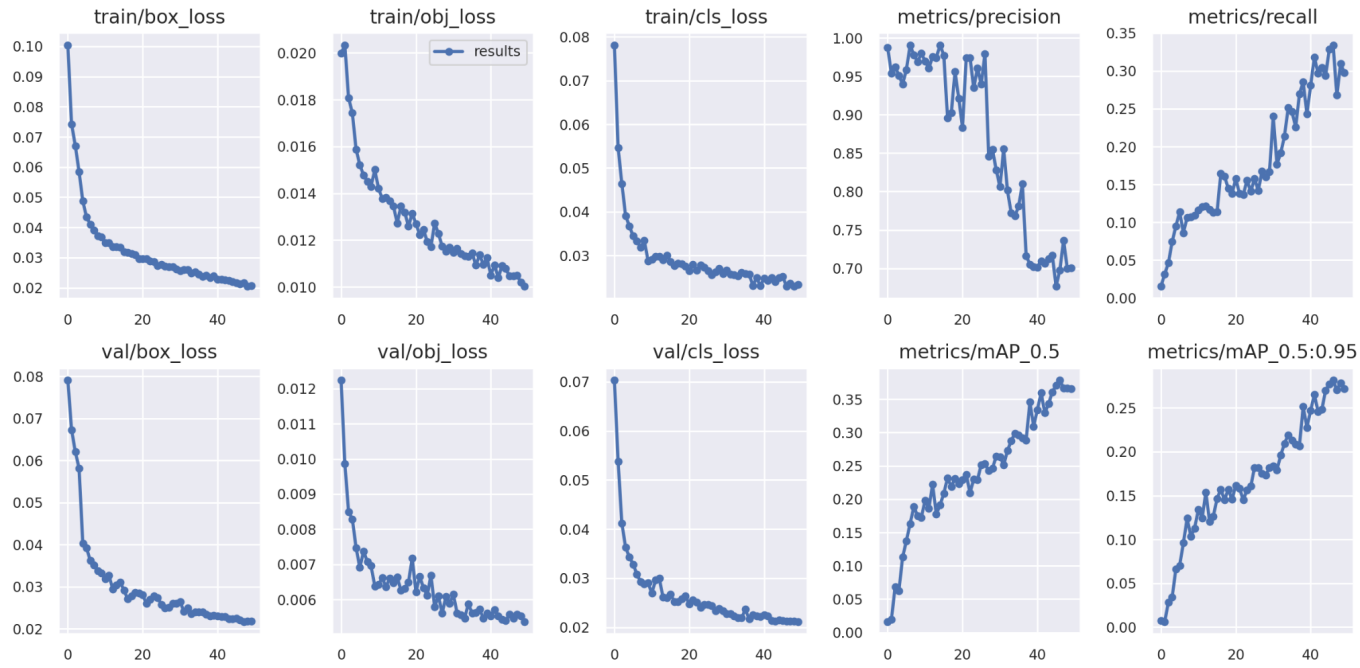

Figure 7: Results of modified evolutionary algorithm V4

For this particular parameter, by changing the upper limit, the results are very similar to the very initial default model (reference figure 1). What this indicates is that the model takes the upper limit over the value produced for image rotation. Furthermore, it also seems that by rotating the images by 90 degrees, it creates confusion with the model. Thinking about it from a human perspective, it isn't realistic to see a license plate at 90 degrees but I thought that it may help with the training. In addition, by having the plate at 90 degrees, it will be a lot harder for the machine to detect and compare with ground truth with better precision and accuracy. While I conducted this test, I conclude it was an illogical experiment that came with its own lessons.

Conclusions and Further Work

Starting with method one, I immediately realized that I am not getting anywhere by messing with the architecture. Ultralytics came up with the right depth and widths such that the performance would increase from smaller to greater networks. In addition, the optimizers, strides and batch sizes also fall under the same category in that they seem to be optimal. In other words, the architectures seem to be optimized enough. Method two however revealed that there is still plenty of room to optimize yolov5s through the hyper parameters. In fact, if you look at the mAP[.5] in figure 1 and figure 6, you will see that the metric surpasses documented mAP. In this dataset, there were multiple classes based off of the license plate state which complicates the model significantly. There are 30 hyper parameters and each parameter has an upper and lower bound. One can carefully manipulate the limits to find a more optimal range.

In addition, it would be better to use binary classification to identify whether or not there is a vehicle nearby. Furthermore, training the data with a lot more images will also be necessary to help the machine detect vehicles better. In regards to the hyper paramaters, they can further be optimized by running more epochs. Therefore I conclude that combining these improvements with the architectural changes from yolov5 to yolov8 will then produce results that are a lot more optimal than the outputs here or from running yolov8 when pre-trained on the cocoa dataset. In this project, I have shown the significance of running the model on custom data rather than mere pre-trained data as it will certainly have an impact on the results.