Road Sign Detection using Deep Learning: YOLOR Part 2

Younes EL BOUZEKRAOUI
Georgia Institute of Technology
ybouzekraoui3@gatech.edu

***Abstract*—As technology has advanced quickly, cars have taken on increasing importance in our daily lives. Traffic Signs Recognition (TSR) systems are one of the most significant research areas. This project is about Traffic sign detection using YOLOR, the purpose of this second part is to train and test YOLOR and find the optimal parameters.**

I. Introduction

All YOLOR models, with their different parameters, were trained, validated, and tested on the GTSDB Dataset in Google Collaboratory. using a GPU Nvidia Tesla T4 with a memory of 16 GB

II. Images Augmentation

Images augmentation is a technique widely used in Object Detection in order to create artificially more data to avoid over-fitting and deal with the unbalanced classes issue. Therefore, in order to deal with those problems that we encountered in Part1 we performed only pixel level augmentation (Brightness, Contrast, Hue, Saturation, Gray level, Motion Blur) to avoid creating artificial images that are not likely to be found in real-world data (Such as a rotated road sign).

The figures below show the class distribution before (Left) and after (Right) the augmentation.

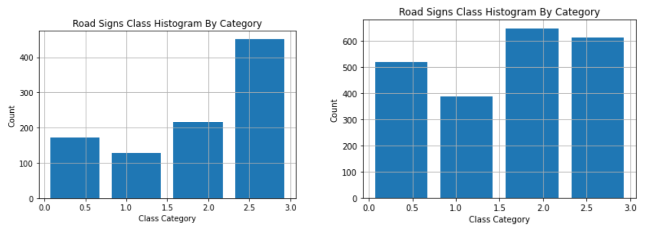


Fig. 1. Classes Distribution

As we can see we were able to have more well-balanced classes and more data, and we expect that this will help the model avoid overfitting.

III. Training the model

A. *Training strategy*

Starting with pre-trained weights, the strategy for training the model involves training it over 300 epochs using the largest batch size that the hardware can support, in our case a batch size of 6. and using various metrics to track its evolution over time and define stopping criteria

The metrics of each run can be tracked, saved, and visualized with the help of WANDB, The majority of this report's charts were produced by WANDB.

The stopping criteria are determined by the following metrics:
The Objectness loss as it measures how well the model identifies the locations and classes of objects.
The mAP 50:95 as it defines the mean average precision mAP over different IoU thresholds, from 0.5 to 0.95, step 0.05 (0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95).

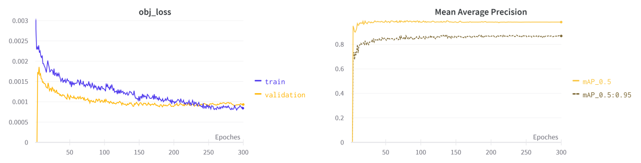


Fig. 2. Objectness loss and mAP

As we can see from the figures above, there is No observable evolution in terms of validation objectness loss after around 100 epochs, therefore we decided to stop the training at 100 epochs to economize resources and more importantly avoid over-fitting

In term of mAP 50:95 , also no visible evolution after around 100 epochs or even less.

B. *Parameter Tuning*

The learning rate and momentum are the hyperparameters chosen for tuning since both parameters are significant for model performance and processing speed, as they determine how the algorithm is conducting out its search for the optimal weights.

*1) **Learning rate:*** Multiple values of the learning rate are tested, I searched across various orders of 10 to find an optimal range of learning rates. Then, I search in smaller increments

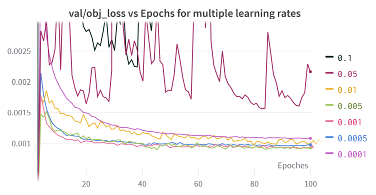


Fig. 3. Validation objectness loss - Learning rate tuning

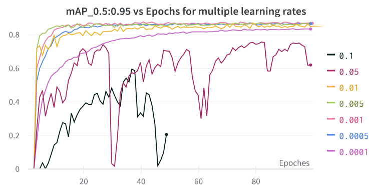


Fig. 4. mAP - Learning rate tuning

As we can see from the figure above, high learning rate results in a model that can't learn because it is taking big steps and thus missing the optimal value and resulting in a model that might not be able to predict anything accurately. On the other hand, very small learning rates, results in slow learning, this means more training time and more resources. A desirable learning rate is one that's low enough so that the network converges to something useful but high enough so that it can be trained in a reasonable amount of time. Learning rates [0.0005,0.001,0.005] are a good fit, and we got a minimal loss and maximum mAP for a learning rate of 0.005

*2) **Momentum**:* The idea of momentum, as applied to YOLOR and other Neural Networks, states that earlier modifications to the weights should have an impact on the present direction of movement in the weight space.

For instance, a high momentum value (such as 0.99) indicates that the previous update has had a significant impact, whereas a low momentum value (such as 0.1) indicates that the previous update has had very little impact.

It is intended to speed up the optimization process, for example, by reducing the number of function evaluations necessary to achieve the optima, or to enhance the efficiency of the optimization algorithm, for instance, by resulting in better results.

To identify an ideal range of momentum, several values across a range of orders of 0.1 are evaluated, then a more refined search across a range of orders of 0.01 is performed to find the optimal momentum.

A fixed value of the learning rate was used 0.005, as it gave us the optimal learning rate in the previous section.

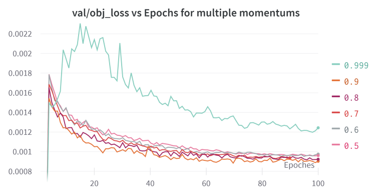


Fig. 5. Validation objectness loss - Momentum tuning

As we can see from the figure above, the momentum 0.9 gave the lowest validation objectness loss, therefore, a more precise search is conducted around this value

After testing multiple values, the optimal momentum found was 0.98.

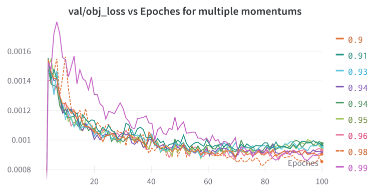


Fig. 6. Validation objectness loss - Momentum tuning

*3) **Optimizer**:* Testing both Adam and SGD Optimizers

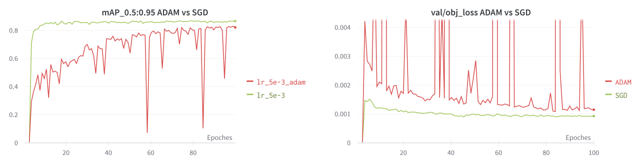


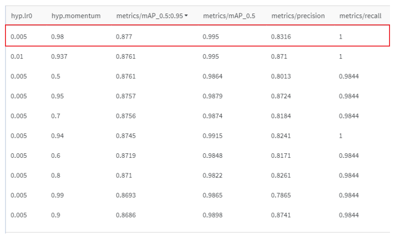
Fig. 7. ADAM vs SGD

The above figures compare the validation objectness loss and the mAP 0.5:0.95 using the same learning rate: 0.005 and momentum 0.937 for the two optimizers ADAM and SGD. With a faster convergence, a higher mAP, and a less fluctuation curve, the SGD-based model outperformed the ADAM-based Model.

When comparing two models using different optimizers, one model's learning rate and momentum values did not match up well with the other. This behavior showed that, although being configured to the same learning rate and momentum value, the two SGD and ADAM optimizers behaved differently. This is due to the approach that each model uses to solve the optimization problem.

C. *Selecting the best model*

Based on the results obtained from the parameter tuning, we compared the different models based on the following metrics mAP50 mAP50:95 Precision, and Recall to help choose the optimal model, the table below shows the results:



hyp.lr0	hyp.momentum	metrics/mAP_0.5:0.95	metrics/mAP_0.5	metrics/precision	metrics/recall
0.005	0.98	0.877	0.995	0.8316	1
0.01	0.937	0.8761	0.995	0.871	1
0.005	0.5	0.8761	0.9864	0.8013	0.9844
0.005	0.95	0.8757	0.9879	0.8724	0.9844
0.005	0.7	0.8756	0.9874	0.8184	0.9844
0.005	0.94	0.8745	0.9915	0.8241	1
0.005	0.6	0.8719	0.9848	0.8171	0.9844
0.005	0.8	0.871	0.9822	0.8261	0.9844
0.005	0.99	0.8693	0.9865	0.7865	0.9844
0.005	0.9	0.8686	0.9898	0.8741	0.9844

Fig. 8. Comparison Table

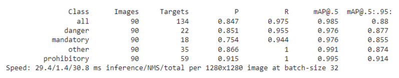
The optimal model we retained is the SGD-based model with a learning rate of 0.005 and a momentum of 0.98

IV. Testing And Evaluating

In this Section, the optimal model using the best weights was tested on the test images, which were not seen by the network before, with the purpose of evaluating the performance of the model on unseen data.

The test set contains 90 images, 12 of them are considered background images (without traffic signs).

The summary of the different metrics is shown in the figure below.



```
               Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95:
                 all          90         134       0.847       0.975       0.985        0.88
              danger          90          22       0.851       0.955       0.976       0.877
           mandatory          90          18       0.754       0.944       0.976       0.855
               other          90          35       0.866           1       0.991       0.874
         prohibitory          90          59       0.915           1       0.995       0.914
Speed: 29.4/1.4/30.8 ms inference/NMS/total per 1280x1280 image at batch-size 32
```

Fig. 9. Summary test results

The Yolor code was modified in order to plot the Confusion Matrix, which we can see in the figure below.

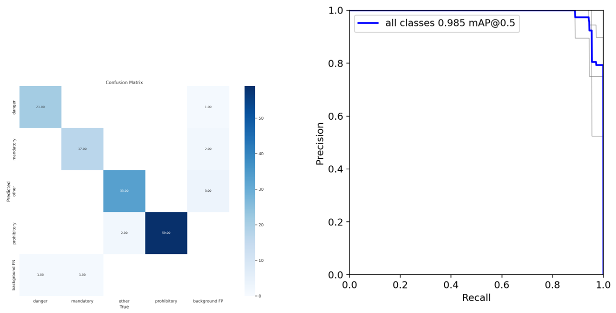


Fig. 10. Confusion Matrix and Precision-Recall Curve

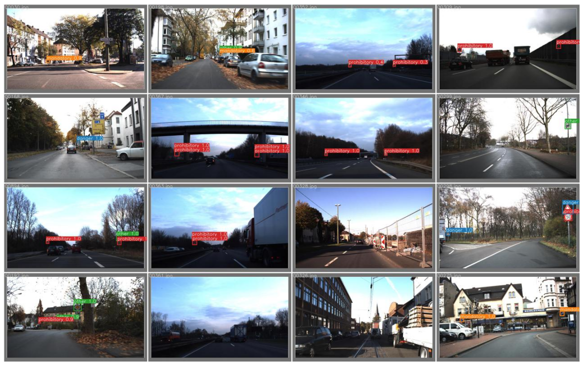


Fig. 11. Example of Predicted Labels

As we can see from the results, our model performed well on the test images. We got an overall mean average Precision (mAP50) of 98.5% The mAP50 of each class is higher than 97.6% with a maximum mAP50 for the prohibitory class, this can be explained by the high number of labels of this class on the training data.
Concerning the Inference time, we were able to get 46ms as inference time (Inference Speed: 1/0.046 = 21fps) for batch-size-1.
In terms of False positives (FP: a sign is detected but no sign is present in the image) and False Negative (FN: a sign is not detected but a sign is present in the image) we got 6 FP and 2 FN .
False positives and False Negatives is a common issue encountered in object detection, as we discussed in part-1 before the training, we added the background images to the training data so that the network gets used to images without labels, and thanks to that and to the data augmentation that we did, we were able to get this low number of FP.
There are other ways to even reduce more this number such as making the confidence threshold higher or adding more background images to the training set , or performing more data augmentation
Compared to the state of the art in sign or object detection, this model was able to achieve very good results in terms of mean Average Precision of more than 98% for mAP50 however in terms of speed our model is not very fast, we got 21 fps as inference speed, the other state of the art in object detection, however, have an inference speed of more than 50 fps for the same image size (1280px*1280px).
In our approach, we focused on maximizing the precision, and we did not give importance to the network size, and even we did not resize the images to make them smaller (640px) but we used the maximum possible size available (1280px), that resulted in a very high precision but in the other hand the model is somewhat heavy.
We can try to train the model on 640px images or even reduce the model's size however this will reduce the accuracy, thus we need to find a trade-off between these two.

V. Challenges

- Training on the Free Version of google Collab is not the best solution.
- Cannot use the Hyperparameter Evolution feature as it requires a very long time
- Debugging YOLOR code.
- The amount of data available is not enough for training a powerful model.

VI. Possible improvements

- Using multi GPU to accelerate the training.
- More Road sign images will improve the performance of the model (other traffic sign datasets)
- Using additional information like the distance of the object could improve the performance of the road sign detection
- Evaluating and optimizing the model in terms of resources and power usage.

References

[1] The German Traffic Sign Detection Benchmark
`https://benchmark.ini.rub.de/gtsdb_dataset.html`
[2] You Only Learn One Representation: Unified Network for Multiple Tasks
`https://arxiv.org/abs/2105.04206`
[3] Traffic Signs Detection and Recognition System using Deep Learning
`https://arxiv.org/ftp/arxiv/papers/2003/2003.03256.pdf`
[4] Traffic Sign Detection
`https://github.com/aarcosg/traffic-sign-detection`