

Evaluation of YOLOv4 on GTSDb Dataset

Aditya Tapshalkar
Fall 2021

1 Introduction

This paper reports my procedures and findings for this study into image processing and object detection of the German Traffic Sign Detection Benchmark (GTSDB) through the utilization of convolutional neural networks and the You Only Look Once (YOLO) framework.

My code results can be found in this GitHub repository: <https://github.com/adityataps/YOLOv4-on-GTSDB>

1.1 The German Traffic Sign Detection Benchmark

The German Traffic Sign Detection Benchmark (GTSDB) is a single-image detection problem with a dataset of pre-labeled images utilized in computer vision and machine learning projects [3]. The GTSDB consists of 900 .ppm images of streets in which a number of detectable and classifiable street signs are present.



Figure 1: One of the images included in the GTSDB dataset

The GTSDB homepage can be found here: https://benchmark.ini.rub.de/gtsdb_news.html

1.2 You Only Look Once

You Only Look Once (YOLO) is a state-of-the-art real-time object detection system utilizing Darknet, an open-source neural network framework written with C and CUDA [4]. YOLO offers real-time neural network object detection that rivals other neural network frameworks Google and Facebook in accuracy and efficiency [1]. The Darknet framework allows for researchers to utilize various YOLO versions for training on CPUs, GPUs, and on the cloud with Google Colab.

The Darknet framework can be found here: <https://github.com/AlexeyAB/darknet>

2 Training YOLO on GTSDB

2.1 Specifications

This study, consisting of GTSDB data processing and two trials of YOLOv4 neural network training, was run on my personal PC. My computer runs Windows 10 with an Intel Core i7-9700K CPU with 8 cores, 16GB RAM, and an NVIDIA GeForce RTX 2070 Super GPU.

To run my study, I utilized the Python IDE JetBrains PyCharm v2021.2 (Professional Edition), Microsoft Visual Studio Community 2019, OpenCV v4.5.3, CUDA v11.2, and CuDNN v8.2.1.

2.2 Annotating

2.2.1 Methodologies

The full GTSDB dataset was downloaded via the GTSDDB homepage. This installation included not only the entire 900 image collection but also the ground truths for each image (Figure 2).

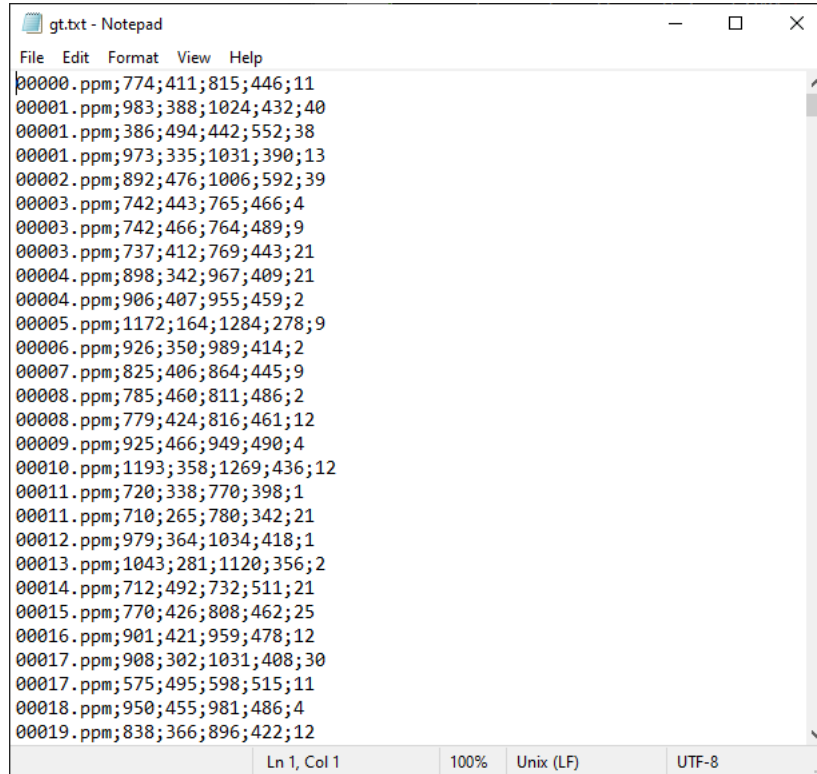


Figure 2: The provided ground truths for each image in the GTSDDB dataset

Each entry in the ground truth text file represents a detectable road sign and is formatted as such:

00000.ppm;774;411;815;446;11

With 00000.ppm representing the image, 774;411;815;446 representing the bounding box of the ground truth of the road sign in the picture (leftCol; topRow; rightCol; bottomRow), and 11 representing the class to which the road sign belongs (Figure 3). Thus, 00000.ppm;774;411;815;446;11 can be interpreted as a "priority at next intersection" sign in image 00000.ppm within the box 774;411;815;446.

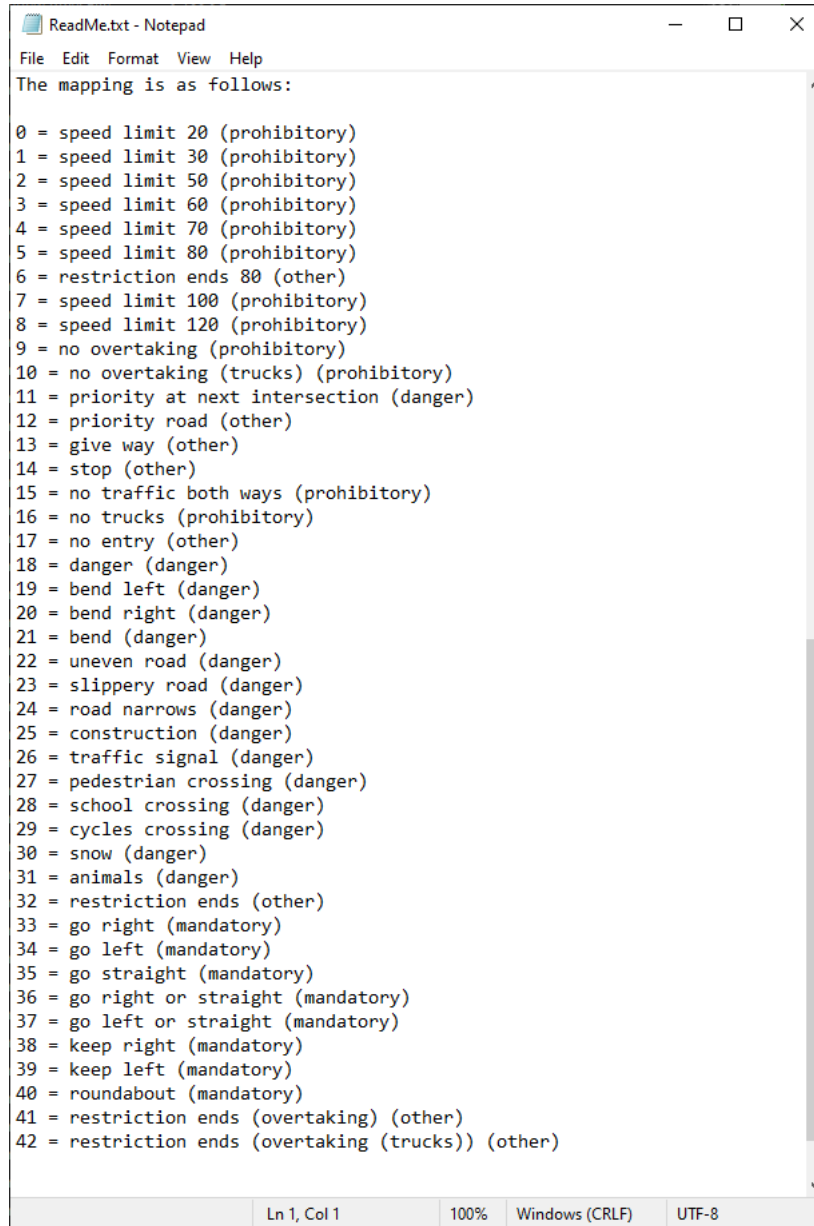


Figure 3: The provided classes of road signs found in the GTSDB dataset

The issue with this formatting of pre-annotated data was that Darknet recognizes a different syntax of image annotations as training and testing input. For example, for each image, there would exist a corresponding `.txt` file formatted as

`<class number> <x_center> <y_center> <width> <height>.`

To format the annotated input images correctly, I utilized a tool called Datasets2Darknet, which runs a Python script to parse and convert several datasets, including GTSDB, into Darknet format labeled images. Datasets2Darknet randomly properly assigns images to training and testing sets depending on the given train-to-test ratio and reduces the original number of GTSDB classes (43) to a small number of generalized classes. A `train.txt` and `test.txt` file is created listing all of the images corresponding the training and testing sets.

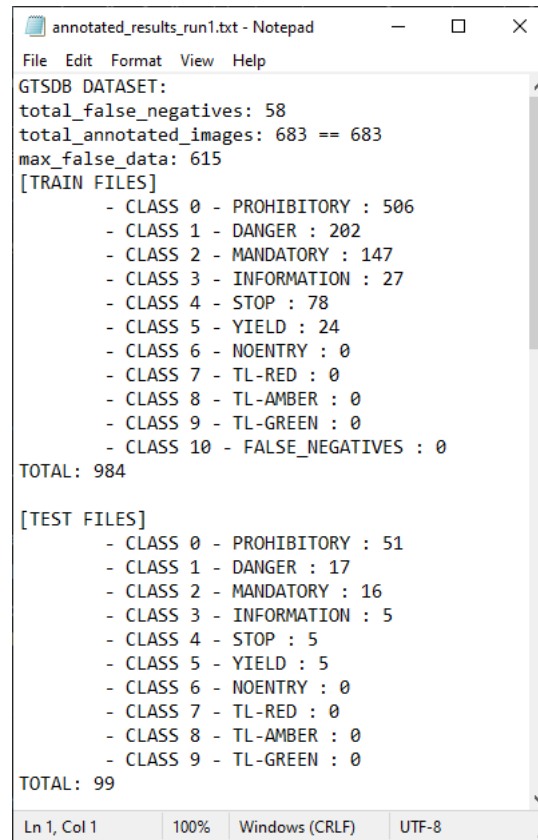
Datasets2Darknet can be found here: <https://github.com/angeligareta/Datasets2Darknet>

2.3 Neural Network Training, Trial 1

This first trial was an initial test of YOLOv4 without making any modifications. The purpose of including this run is to detail the improvements and adjustments I needed to make to obtain better results.

2.3.1 Methodologies

For my initial run of neural network training, I chose to allocate 90% of the GTSDb dataset as training data and the remaining 10% as testing. I utilized the Datasets2Darknet tool to allocate the GTSDb dataset into 10 classes, and the distribution of these images is shown in Figure 4.



```
annotated_results_run1.txt - Notepad
File Edit Format View Help
GTSDb DATASET:
total_false_negatives: 58
total_annotated_images: 683 == 683
max_false_data: 615
[TRAIN FILES]
- CLASS 0 - PROHIBITORY : 506
- CLASS 1 - DANGER : 202
- CLASS 2 - MANDATORY : 147
- CLASS 3 - INFORMATION : 27
- CLASS 4 - STOP : 78
- CLASS 5 - YIELD : 24
- CLASS 6 - NOENTRY : 0
- CLASS 7 - TL-RED : 0
- CLASS 8 - TL-AMBER : 0
- CLASS 9 - TL-GREEN : 0
- CLASS 10 - FALSE_NEGATIVES : 0
TOTAL: 984
[TEST FILES]
- CLASS 0 - PROHIBITORY : 51
- CLASS 1 - DANGER : 17
- CLASS 2 - MANDATORY : 16
- CLASS 3 - INFORMATION : 5
- CLASS 4 - STOP : 5
- CLASS 5 - YIELD : 5
- CLASS 6 - NOENTRY : 0
- CLASS 7 - TL-RED : 0
- CLASS 8 - TL-AMBER : 0
- CLASS 9 - TL-GREEN : 0
TOTAL: 99
Ln 1, Col 1 100% Windows (CRLF) UTF-8
```

Figure 4: The results of dataset allocation for trial 1

After configuring and building Darknet with OpenCV, CUDA, and CuDNN with Visual Studio, the newly formatted images and corresponding text files were then fed into the YOLOv4 algorithm for training on my GPU.

2.3.2 Results

Training the GTSDb data for 20000 iterations (number of classes * 2000) took roughly 12 hours, allowing me to understand how YOLOv4 measured model reliability.

- Intersection Over Union (IoU): the ratio of the area of the intersection of the ground truth annotated label box and the detected box to the union of the area between the two boxes; the closer to 1 the better, signifying accurate and precise detection of target objects. A threshold for IoU can be configured to determine whether a detection is labeled a true or false positive.

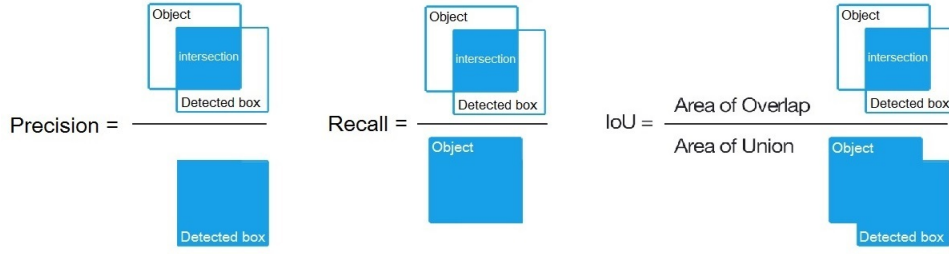


Figure 5: Demonstration of the IoU metric

- Mean Average Precision (mAP): the average of the area under the precision-recall curve for each class [2, 5]. Here, precision is defined as how accurate the model can correctly predict (percentage of predictions that are correct) and is calculated by

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

whereas recall measures how well the model can find all the positives, calculated as

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}.$$

Average precision is described as the area under the precision-recall curve for each class, making the mean average precision to be the the averages of average precisions [5]. Generally, the higher the mAP value the better the model detects the target objects.

- F1 Score: a popular metric in determining model accuracy utilizing precision and recall. F1 score can be calculated as

$$F1 \text{ Score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}.$$

Upon evaluating the trained model on the test dataset, for a detection threshold of 0.25 and IoU threshold of 50%,

- Precision = 0.98
- Recall = 0.86
- F1 Score = 0.92
- Average IoU = 90.21%
- mAP@0.50 = 54.05%

after 20000 iterations.

Although the predictions my model makes is mostly correct, my model seems to fail at detecting objects, as observed in my high number of false negatives.

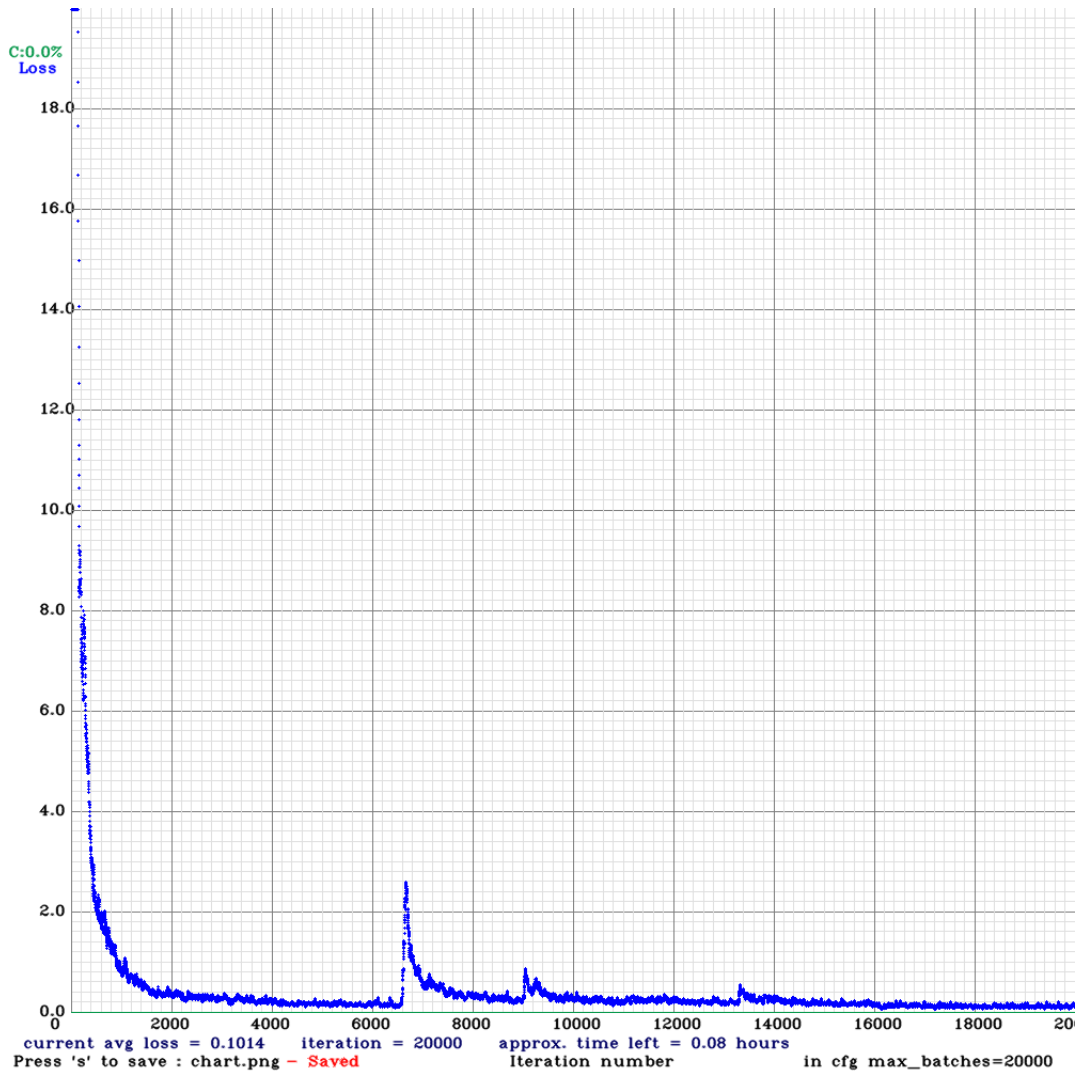


Figure 6: Average loss during training YOLOv4 for trial 1

2.3.3 Improvements to make

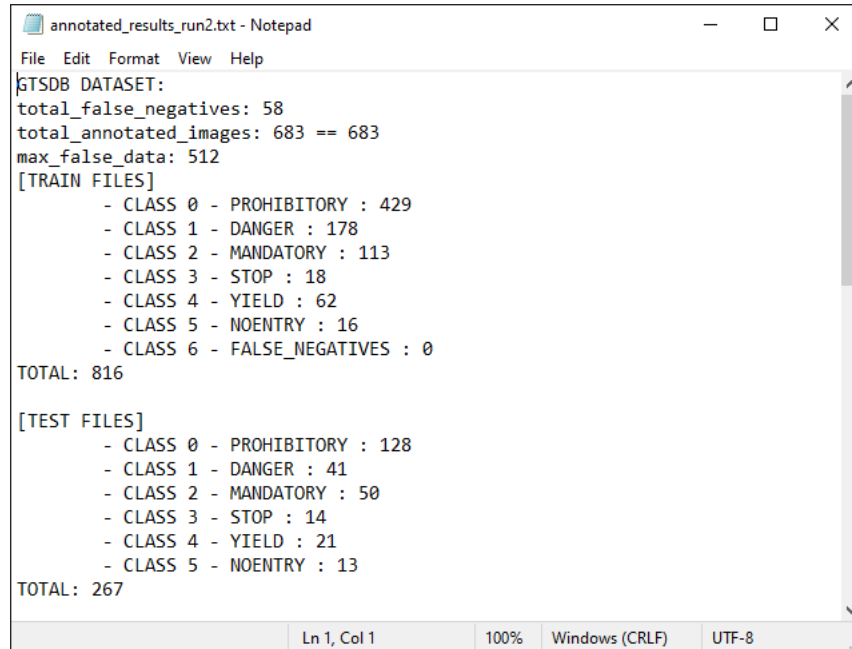
Upon review, I realized that the model had actually started overfitting the training data, hence the large number of false negatives and thus low recall and low mAP. Going through my methodologies, I found the culprit to be my configuration of the Datasets2Darknet tool. I was utilizing classes that were not found in the GTSDb dataset but rather in the other similar datasets that Datasets2Darknet supports. In actuality, the `gtsdb_parser.py` file in Datasets2Darknet listed only 6 generalized classes, omitting the "INFORMATION" and "TL-" categories found in my original method of GTSDb dataset allocation. Seeing that the number of batches (and thus number of iterations) was to be decided by multiplying the number of classes with 2000, the original configuration resulted in 20000 iterations when the correct number of batches should have been $6 * 2000 = 12000$. These improvements would soon be made in the subsequent training of the neural network.

2.4 Neural Network Training, Trial 2

Taking my results from trial 1 into consideration, I changed my configuration of annotated data and kept my YOLO configuration the same, as I was confident that my issue with detection in trial 1 was overfitting due to accounting for unnecessary classes.

2.4.1 Methodologies

I ran the GTSDb dataset through Datasets2Darknet once again, however this time cutting down the number of classes to the 6 necessary classes organized as per the `gtsdb_parser.py` file. The number of batches then was cut to 12000, in hopes of preventing overfitting. Additionally, to further prevent overfitting, the train-to-test ratio was changed from 0.9 to 0.75, allowing for variation upon object detection.



```
annotated_results_run2.txt - Notepad
File Edit Format View Help
GTSDb DATASET:
total_false_negatives: 58
total_annotated_images: 683 == 683
max_false_data: 512
[TRAIN FILES]
- CLASS 0 - PROHIBITORY : 429
- CLASS 1 - DANGER : 178
- CLASS 2 - MANDATORY : 113
- CLASS 3 - STOP : 18
- CLASS 4 - YIELD : 62
- CLASS 5 - NOENTRY : 16
- CLASS 6 - FALSE_NEGATIVES : 0
TOTAL: 816
[TEST FILES]
- CLASS 0 - PROHIBITORY : 128
- CLASS 1 - DANGER : 41
- CLASS 2 - MANDATORY : 50
- CLASS 3 - STOP : 14
- CLASS 4 - YIELD : 21
- CLASS 5 - NOENTRY : 13
TOTAL: 267
Ln 1, Col 1 100% Windows (CRLF) UTF-8
```

Figure 7: The results of dataset allocation for trial 2

The newly allocated Darknet formatted dataset was then fed to YOLOv4 for training.

2.4.2 Results

Upon testing the trained YOLO model on my test dataset under the same detection and IoU thresholds,

- Precision = 0.98
- Recall = 0.97
- F1 Score = 0.97
- Average IoU = 89.66%
- mAP@0.50 = 95.10%

after 12000 iterations.

I was impressed to see a significant increase in recall from the first trial. These results seem to confirm that my model from trial 1 was likely overfitted to the training data. My F1 score metric was greater this trial than it was in trial 1, meaning that this model was better at avoiding false positives while honoring true negatives.

2.4.3 Further Study

I would like to try training new models with altered hyperparameters and thresholds and observe the variations in model reliability. Modifying the number of iterations (as expected) made the model more reliable; given unlimited resources and unlimited time I am highly interested in determining the right parameter values to create an unrivaled state-of-the-art neural network.

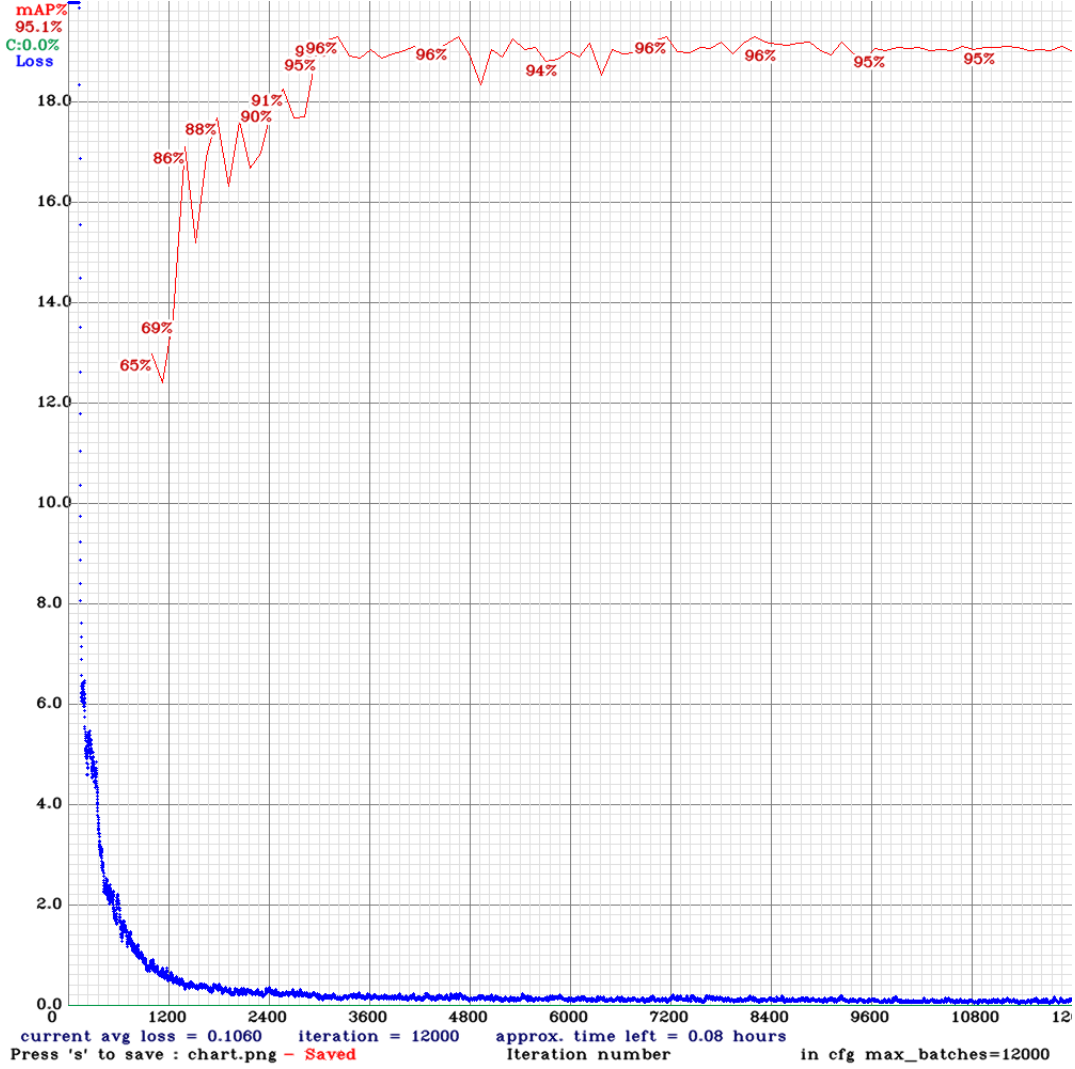


Figure 8: Average loss during training YOLOv4 for trial 1, with mAP displayed in red

3 Reflection and Regards

1. YOLOv4 Hyperparameters

YOLOv4 is versatile and offers configurable hyperparameters (learning rate, epochs, optimization method, etc.) to fit most object detection and neural network needs. However, I decided to keep my hyperparameters consistent to the default configuration during both trials (outside of number of batches to fix overfitting and batch size with recommendation by users in the Darknet Discord channel). In the future I would like to explore the strength of certain hyperparameters and determine which ones I could alter to get my F1 score as possibly close to 1 as I can.

2. Training Strategy

My training strategy was to let YOLOv4 run its iterations with help from the Datasets2Darknet repository. For training, following the guide on training custom datasets from the Darknet GitHub page was instrumental in learning how to configure YOLO to train my GTSDB dataset.

3. Detection Performance and Evaluation

I looked at model performance metrics such as the mean average precision value and F1 score to

determine reliability of the trained models. The goal of neural network training is to maximize true positive and true negative counts while minimizing false positive and false negative counts.

4. False Positive / False Negative Detection

My first trial exhibited a large number of false positives, meaning that my detector was sometimes not detecting road signs that should have been detected. I believed that this was a result of overfitting due to too many iterations. I trained YOLO again but this time reducing the number of classes and iterations by 40%. Doing so yielded a higher reliability sporting a higher precision and recall.

5. Return to Data Annotating

After running trial 1, I did return to data annotating to fix my data allocation.

6. Comparisons with State of the Art Object Detection

This study detailed and analyzed the training process and results for the GTSDb dataset, but only after processing all 43 original classes and conglomerating similar classes into 6 generalized Darknet classes. Although this method of classification exhibits great performance, I expect the reliability of this model to decrease significantly if all 43 original classes were trained, as the average number of images per class would then decrease significantly. Some modifications seen in similar implementations of YOLO on GTSDb were varying network resolution in the `.cfg` file, adding additional convolutional layers, and modifying the thresholds for IoU and detection.

7. Main Challenges

My initial challenges was setting up the environment to run Darknet and YOLO. I spent a few days installing and building Darknet and its dependencies, but after running tests on pre-trained COCO models, I knew I had it up and running.

Aside from environment setup, I did not have many other challenges training the network for the GTSDb dataset. Training this model was a learning experience every step of the way, and I am excited to pursue similar work in my career and research.

Lastly, I give my regards to Professor James Tsai and his graduate colleagues for assigning this interview activity. I am grateful for the chance to go beyond my comfort zone to show how quickly I can learn the fundamentals and utilize an unfamiliar framework to create a strongly reliable model.

3.1 Postscript

Upon testing my second YOLO model on the best weights, my F1-score increased to 96.48%. Wow!

References

- [1] BOCHKOVSKIY, A. Yolov4 — the most accurate real-time neural network on MS COCO dataset. <https://alexeyab84.medium.com/yolov4-the-most-accurate-real-time-neural-network-on-ms-coco-dataset-73adfd3602fe>, Jul 2020.
- [2] BOCHKOVSKIY, A., WANG, C.-Y., AND LIAO, H.-Y. M. Yolov4: Optimal speed and accuracy of object detection, 2020.
- [3] HOUBEN, S., STALLKAMP, J., SALMEN, J., SCHLIPSING, M., AND IGEL, C. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In *International Joint Conference on Neural Networks* (2013), no. 1288.
- [4] REDMON, J. Darknet: Open source neural networks in C. <http://pjreddie.com/darknet/>, 2013–2016.
- [5] YOHANANDAN, S. Map (mean average precision) might confuse you!, Jun 2020.