Robotics 1 - ME 592 Final Project Report

Deploying the YOLOv5 Neural Network on a Raspberry Pi 4 Model B for Traffic Sign Recognition

Roselynn Conrady and Michael Holm

Abstract

As neural networks are developed to be better at recognizing images, it becomes essential to optimize them to be able to run on smaller hardware at faster speeds [1]. This can be difficult, because some small hardware doesn't have access to fast gpus, are any gpu at all in some cases [2]. Several solutions have been developed for this problem, including model pruning, model quantization, and network optimization [3], [4]. YOLOv5 is one of the most used neural networks for applications of image recognition, and has been somewhat optimized for both performance and speed [5].

The goal of this work was to train this YOLOv5 network to recognize common traffic signs, and be able to locate them in an image, quickly, on a single board computer. This goal was accomplished using a Kaggle dataset of various German traffic signs [6]. This model was then deployed to the Raspberry Pi 4 Model b single board computer, which had a Linux based operating system installed [7]. Using a PyTorch module designed specifically for the Raspberry Pi, this model was run on the single board computer, while collecting basic performance metrics like CPU temperature, model framerate, and model performance.

After these initial metrics were gathered, an attempt at model pruning was made, to increase the speed of the network while maintaining as much performance as possible. The YOLOv5 model was pruned using a built in function created by the YOLOv5 team, at both 30% and 50% pruning modes. This means that 30% and 50% of the model's parameters were reduced to zero, respectively. Our team found no noticeable performance improvement on the Raspberry Pi for either the 30% or the 50% pruned model. However, it was found that when running on a discrete GPU, these pruned models do perform faster in inference time. The 50% pruned model performed the fastest in terms of speed, but the worst overall. However, both prudent models came with a noticeable decrease in performance.

Finally, our team presents possibilities for future work in the field. A Raspberry Pi is not the most suitable microprocessor for neural networks, as it has no discrete GPU, and does not have great thermal management structures. There are more suitable options that could be tested. Additionally, model quantization could be used to speed up the network. This approach reduces the accuracy of model parameters from floats, to smaller sized integers. This method could be used to create noticeable improvement in the performance of a neural network on the Raspberry Pi.

Introduction

Autonomous vehicles require intelligent vision systems to detect obstacles and obey traffic laws [14]. With that said, artificial neural networks (oftentimes, shortened to “neural networks”) can be used to teach these intelligent vision systems. Neural networks are also very dense, and therefore, techniques such as model pruning are used to optimize these networks. Model pruning introduces sparsity (e.g., weights that are close to zero are set to zero) in order to improve model performance but at the cost of, for example, accuracy. We propose to deploy a machine learning model to a Raspberry Pi 4 Model B to explore model pruning techniques, and to evaluate the performance of the models at various states of pruning.

Methodology

Materials

The programming language we used was Python along with the Python modules we have used throughout the ME 592 class (e.g., PyTorch). Google Colaboratory (Google Colab) was used to gain access to their GPU to train and validate the model. The model we used was YOLOv5 (“You Only Look Once” version 5), which is a CNN-based object detection algorithm [15]. To train the model, we used the publicly available “Traffic Signs Dataset in YOLO Format” dataset from Kaggle [6]. As mentioned in the Introduction, a Raspberry Pi 4 Model B was the hardware platform we deployed the machine learning model to. Furthermore, we used the Raspberry Pi OS 64-bit and the Raspberry Pi Standard Camera (8 megapixel camera).

Procedure

In Google Colab we imported libraries and dependencies, and then we mounted our Google Drive. Next, we cloned the YOLOv5 GitHub repository and then we downloaded the “Traffic Signs Dataset in YOLO Format” dataset. Using modified code from [12], we divided the dataset’s images and their respective annotations into a training or validation folder. The result was 592 images for training, and 148 images for validation. The next step was to create a .yaml file that contains information - such as the file paths of the training and validation folders, number of classes, and class names - for the YOLOv5 algorithm to train the model on a custom dataset.

Ultimately, we decided the best traffic sign detection results were trained with the following parameters: setting the image size to 1360 x 1360 pixels, batch size of 16, training for 30 epochs, and using the default yolov5s.pt weights. Figures 1, 2, and 3 demonstrate the performance of our model on Google Colab. Overall, these results seemed to accurately detect traffic signs.

Figure 1: Batch of validation images that our original model predicted

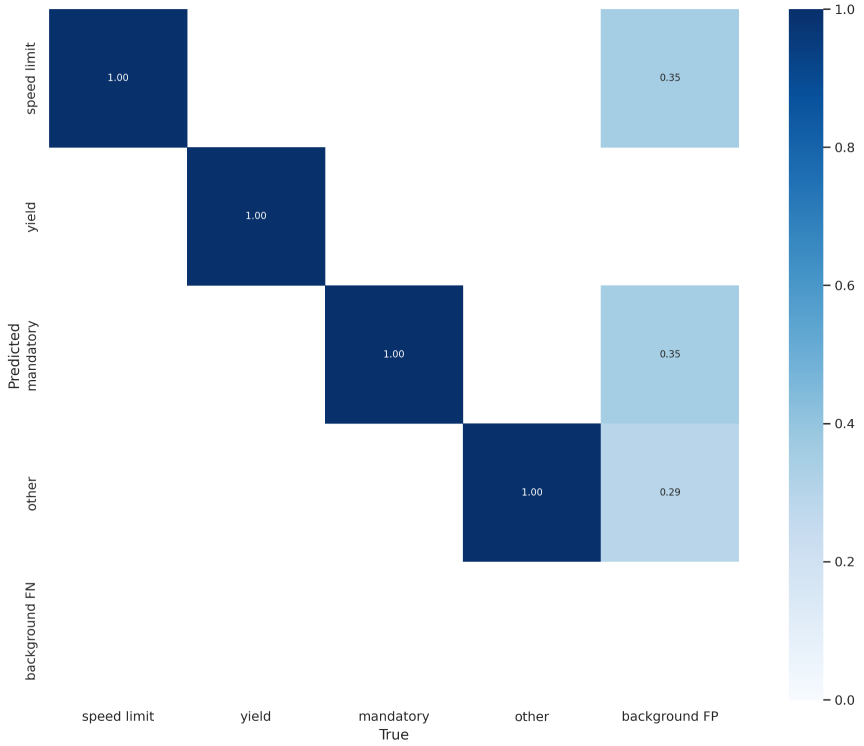

Figure 2: Confusion matrix for our original model

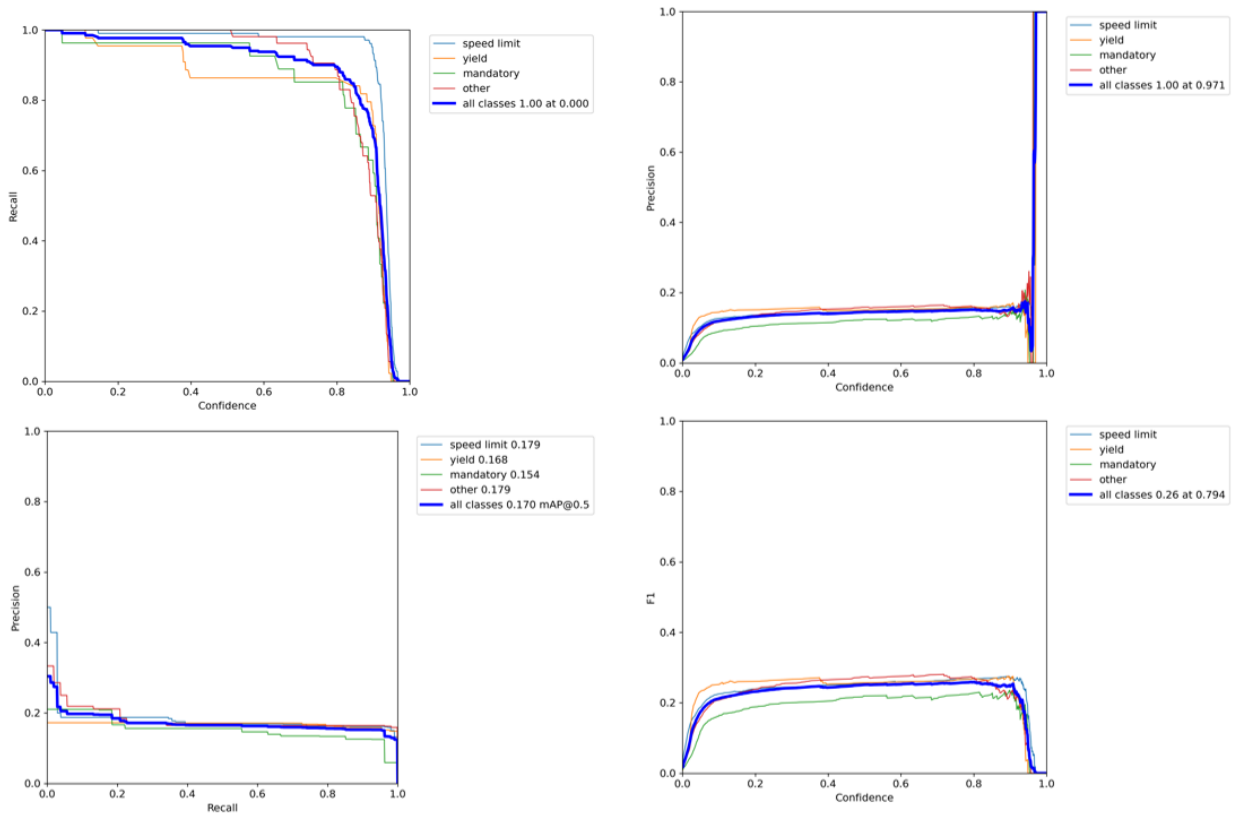

Figure 3: From top left going clockwise: Recall, Precision, Precision-Recall, and F1 curves of our original model

Next, we pruned the model in Google Colab with 30% and 50% global sparsity using PyTorch's "prune" function. We ran validation tests to compare the performances between the original model, the pruned 30% global sparsity model, and the pruned 50% global sparsity model.

Lastly, we deployed the models onto the Raspberry Pi and evaluated the performance of the models. This was done by transferring the trained model to the Raspberry Pi using YOLOv5's built in model transfer functions, whose code can be found in our Git repository. Various scripts were created to gather metrics while the model was running. These metrics included time per frame analyzed, and CPU temperature over time.

Results and Discussion

From the validation tests from within Google Colab, we found that there was minimal change between the original model and the pruned 30% global sparsity model. There was a slight decrease in inference time with the pruned 30% global sparsity mode, but overall, it performed worse than the original model. The pruned 50% global sparsity model performed significantly worse than the other models, but it performed the fastest in terms of speed. Figure 4 highlights the number of detected labels precision (P), recall (R), mean average precision (mAP) at an IoU

threshold of 0.5, and mAP at an IoU threshold of 0.95. Additional model performance comparison images can be found in the appendix.

```
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 28/28 [01:16<00:00,  2.74s/it]
                 all        889        228      0.151        0.9      0.171      0.138
         speed limit        889        104      0.155      0.981      0.181      0.149
               yield        889         44      0.158      0.864       0.17      0.138
           mandatory        889         27      0.131      0.852      0.154      0.123
               other        889         53      0.162      0.906      0.178      0.144
Speed: 0.6ms pre-process, 17.2ms inference, 1.8ms NMS per image at shape (32, 3, 1376, 1376)

               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 28/28 [00:30<00:00,  1.10s/it]
                 all        889        228      0.153      0.683      0.165      0.113
         speed limit        889        104      0.158      0.971      0.199      0.132
               yield        889         44       0.16      0.545      0.164      0.112
           mandatory        889         27      0.148       0.63      0.143     0.0967
               other        889         53      0.146      0.585      0.156      0.112
Speed: 0.6ms pre-process, 15.4ms inference, 1.9ms NMS per image at shape (32, 3, 1376, 1376)
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 28/28 [00:29<00:00,  1.07s/it]
                 all        889        213     0.0585      0.322     0.0421     0.0123
         speed limit        889        104     0.0516      0.683     0.0642     0.0201
               yield        889         37     0.0606      0.135      0.032    0.00747
           mandatory        889         25     0.0554       0.28     0.0333     0.0117
               other        889         47     0.0664      0.191     0.0389    0.00988
Speed: 1.2ms pre-process, 16.3ms inference, 1.8ms NMS per image at shape (32, 3, 1376, 1376)
```

Figure 4: Top to bottom: original model, 30% pruned global sparsity model, and 50% pruned global sparsity model

Upon deployment to and evaluation of the Raspberry Pi with the original model, it was found that performance is not satisfactory for any real time sign recognition applications, however the network is suitable for basic sign recognition applications, as long as there is no need for real time recognition. These findings are supported by the figures shown. In Figure 5, a graph is shown of time per frame analyzed by the Raspberry Pi, over the course of 19 frames. As can be seen in Figure 5, inference times are clearly not usable for real time sign recognition. An attempt was made to improve the calculation time per frame for this network by pruning the model used using a built in YOLOv5 function, but these pruned models did not create any noticeable increase in performance on the Raspberry Pi. However, when running the model on a Google Colab discrete GPU, there was a decrease in inference time speed. Because of this, our team believes that if we used a single board computer with a discrete GPU, we would be able to achieve increased performance by pruning the YOLOv5 model. Unfortunately, the Raspberry Pi does not have a discrete GPU to test these pruned models well.

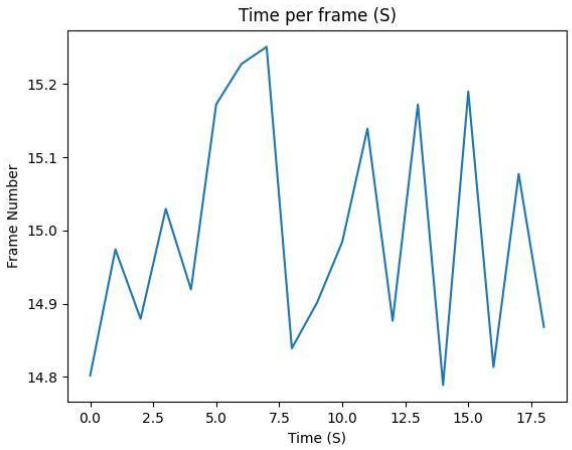

Figure 5: Calculation time per frame

Additionally, Figure 2 below shows a graph of CPU temperatures over the course of the 19 frames analyzed. The Raspberry Pi starts to throttle itself at 60 degree Celsius, and it can be noticed that CPU temperatures just barely reach this point. This means that the Raspberry Pi may be throttling itself. This could explain the oscillating nature of the analysis time per frame chart in Figure 5. This also means that using a more effective cooling technique could lead to increased performance. (Note: The Raspberry Pi used did have small heat sinks and a fan).

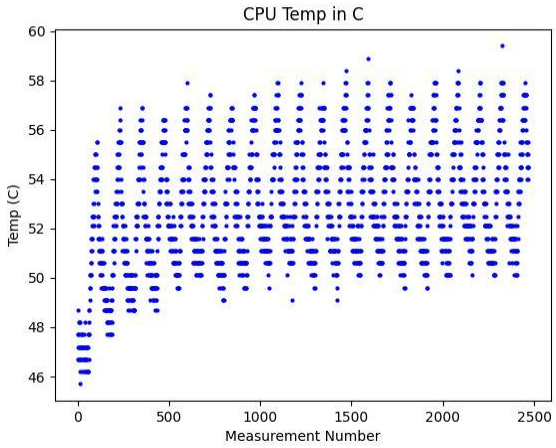

Figure 6: CPU Temp overtime

Aside from the slow performance of the network, results were satisfactory. The network does a good job of recognizing road signs shown to it in various environments, and is able to recognize several signs at once, while having no false positives. Shown in Figure 3 below is an image of a speed limit sign being recognized by the network. This image was taken by an 8 megapixel camera designed specifically for the Raspberry Pi [8]. Additional images of signs being recognized can be found in the appendix. We did notice that the model occasionally had trouble recognizing mandatory signs from the phone, but shown by the test data, the model performs well when proper mandatory sign pictures are used.

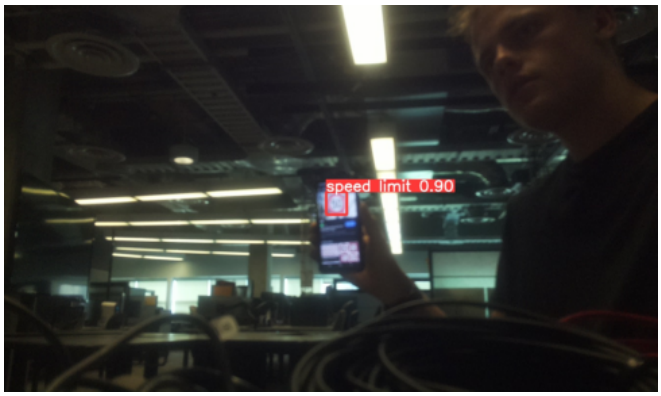

Figure 7: Recognized speed limit sign

This project has several areas of possible future work. Due to the short time span available for this project, we were unable to look into these additional areas. One of these areas is known as model quantization. This practice reduces the memory size of the parameters used in the model. This is done by reducing the float parameters to integers, or some other type of smaller memory number memory type. According to this article [9], it is possible to quantize the YOLOv5 model, and it does result in a faster model, however it takes quite a bit of work to quantize this model, and our team doesn't have sufficient time to be able to do this my the date this paper is due.

Another possible area of future work is using an improved cooling method for the Raspberry Pi's CPU. As shown in Figure 2, the CPU's temperature frequently gets very close to, or just above 60 degree Celsius. This graph was created in a fairly cool air conditioned room, so it's possible

that in a hot room, the Raspberry Pi would begin to throttle itself. By using an improved cooling method, possibly a basic water pump to cool the CPU, results could be analyzed to see what the Raspberry Pi can achieve when CPU temperature is not a factor.

Conclusion

Ultimately, our team found that the Raspberry Pi, in the cooling configuration that we used it in, is not suitable for real time YOLOv5 traffic sign recognition. However, it is able to achieve good accuracy in its ability to recognize images of traffic signs, and is able to recognize these signs in various environments. We also found that pruning the YOLOv5 model does not provide increased performance speed on the Raspberry Pi, however it does provide better inference time when using a discrete GPU. The team found that the Raspberry Pi is approaching a 60 degree Celsius throttle point when running this model, and theorizes that, if the model were to be run in a slightly warmer environment, the CPU would begin to throttle itself to maintain a suitable temperature. The team believes that, with sufficient time, this model could be quantized to be able to run much faster, and a more suitable single board computer could be used, in place of the Raspberry Pi, ideally one with discrete GPU. Finally, the team believes that a better cooling system for the CPU will result in faster inference speeds for the model when deployed on the Raspberry Pi.

References

1) Situnayake, Daniel. “Make Deep Learning Models Run Fast on Embedded Hardware.” *Edge Impulse*, 29 Apr. 2020, https://www.edgeimpulse.com/blog/make-deep-learning-models-run-fast-on-embedded-hardware.
2) Community. “Best Single Board Computers for AI and Deep Learning Projects.” *It's FOSS*, 10 Jan. 2022, https://itsfoss.com/best-sbc-for-ai/.
3) Bandaru, Rohit. “Pruning Neural Networks.” *Medium*, Towards Data Science, 1 Sept. 2020, https://towardsdatascience.com/pruning-neural-networks-1bb3ab5791f9.
4) Danka, Tivadar. “How to Accelerate and Compress Neural Networks with Quantization.” *Medium*, Towards Data Science, 17 Aug. 2021, https://towardsdatascience.com/how-to-accelerate-and-compress-neural-networks-with-quantization-edfbbabb6af7.
5) “Pytorch.” *PyTorch*, https://pytorch.org/hub/ultralytics_yolov5/.
6) Sichkar, Valentyn. “Traffic Signs Dataset in Yolo Format.” *Kaggle*, 3 Apr. 2020, https://www.kaggle.com/valentynsichkar/traffic-signs-dataset-in-yolo-format.
7) Raspberry Pi. “Raspberry Pi Os.” *Raspberry Pi*, https://www.raspberrypi.com/software/.
8) Industries, Adafruit. “Raspberry Pi Camera Board V2 - 8 Megapixels.” *Adafruit Industries Blog RSS*, https://www.adafruit.com/product/3099?gclid=Cj0KCQjwmuiTBhDoARIsAPiv6L_4ORQla_XspOejvUBTung5WkjuIPgTJ79DnjcEGYejdN6v22_xlXoaAkl1EALw_wcB.
9) Neuralmagic. “Sparse Yolov5: 10x Faster and 12x Smaller - Neural Magic.” *Neural Magic - Software-Delivered AI*, 11 Jan. 2022, https://neuralmagic.com/blog/benchmark-yolov5-on-cpus-with-deepsparse/.
10) Zhu, Michael, and Suyog Gupta. "To prune, or not to prune: exploring the efficacy of pruning for model compression." *arXiv preprint arXiv:1710.01878* (2017).
11) Sichkar, Valentyn. “Traffic Signs Dataset in Yolo Format.” *Kaggle*, 3 Apr. 2020, https://www.kaggle.com/valentynsichkar/traffic-signs-dataset-in-yolo-format.
12) EhsanR47. “EHSANR47/Traffic-Signs-Dataset-in-Yolo-Format: Traffic Signs Dataset in Yolo Format.” *GitHub*, https://github.com/EhsanR47/Traffic-Signs-Dataset-in-YOLO-format.
13) Balakishan77. “Balakishan77/yolov5_custom_trained_traffic_sign_detector: Yolov5 Object Detection on Traffic Signs Dataset with Custom Training.” *GitHub*, https://github.com/Balakishan77/yolov5_custom_trained_traffic_sign_detector.
14) Bechtel, Michael G., et al. "Deeppicar: A low-cost deep neural network-based autonomous car." 2018 IEEE 24th international conference on embedded and real-time computing systems and applications (RTCSA). IEEE, 2018.
15) Ultralytics. “Ultralytics/yolov5: Yolov5 in PyTorch > ONNX > CoreML > TFLite.” *GitHub*, https://github.com/ultralytics/yolov5.

Appendix

GitHub repository link (use the “main” branch):
https://github.com/rho-selynn/592-final-project-YOLO

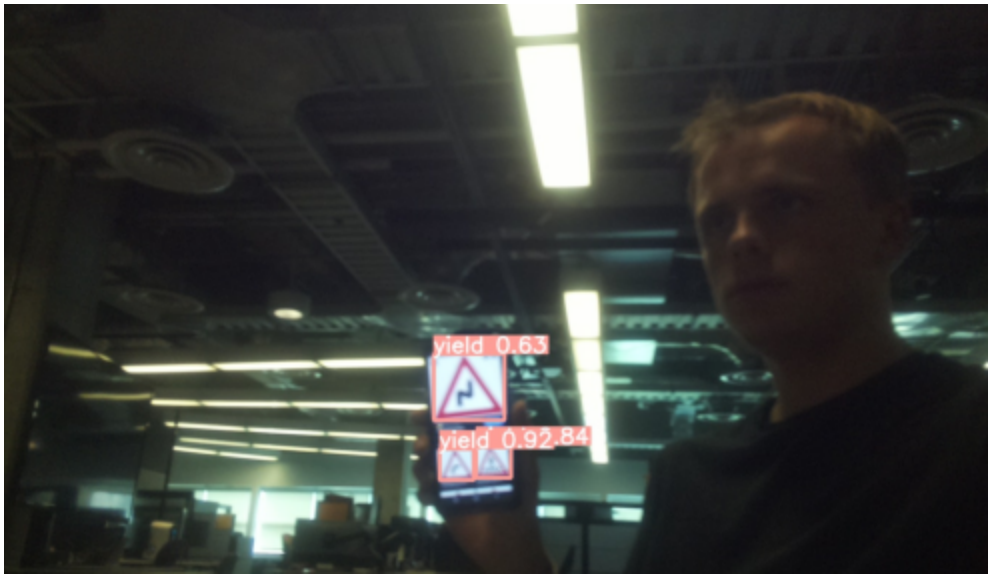

Figure 8: Yield sign detection from Raspberry Pi camera

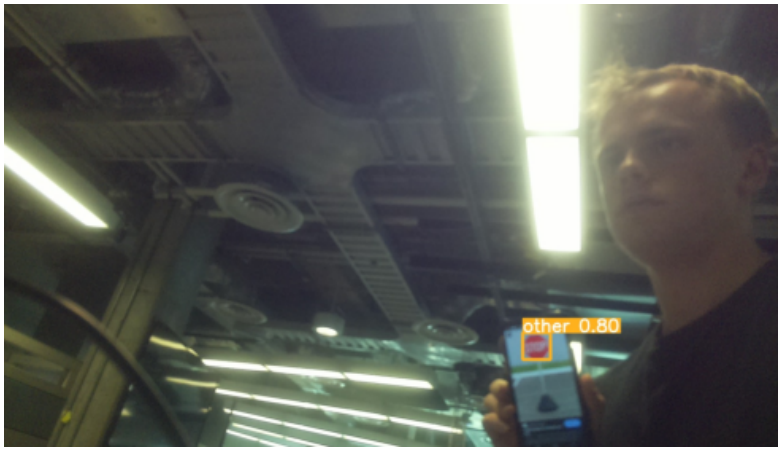

Figure 9: Other sign detection from Raspberry Pi camera

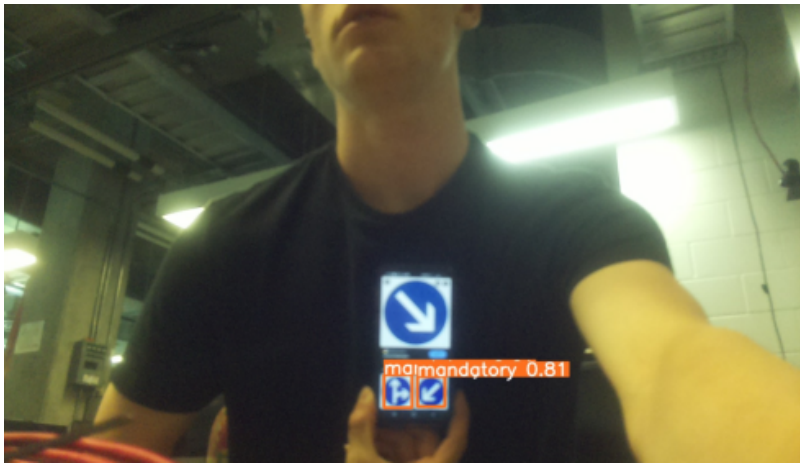

Figure 10: Mandatory sign detection from Raspberry Pi camera

Figure 11: Batch of validation images from original model

Figure 12: Batch of validation images from pruned 30% global sparsity model

Figure 13: Batch of validation images from pruned 50% global sparsity model

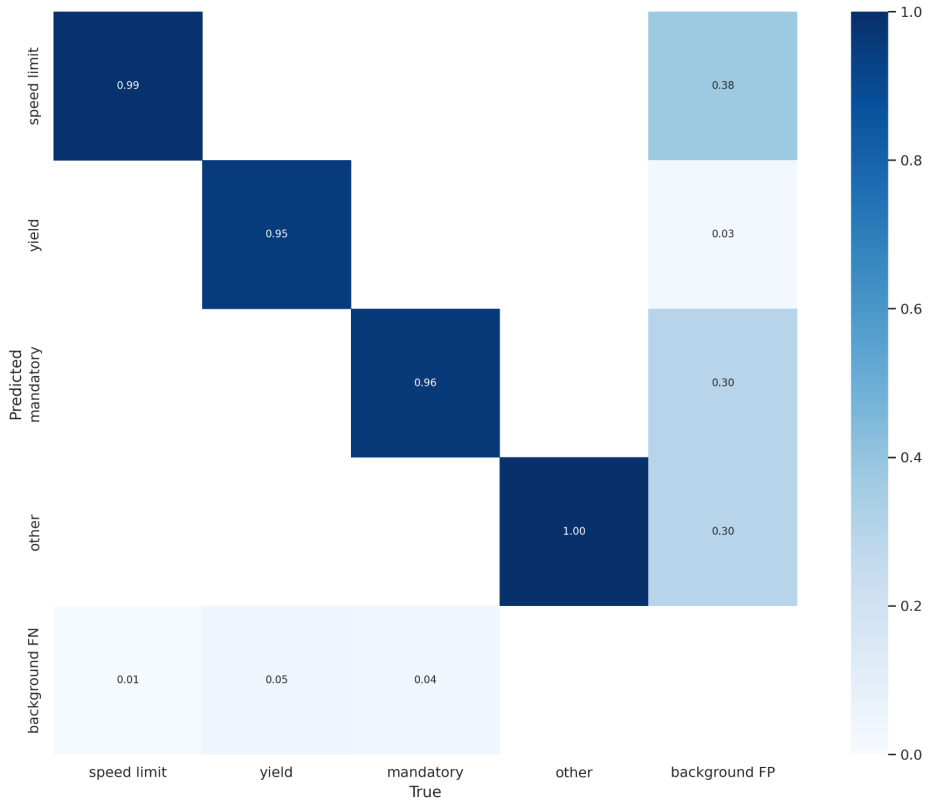

Figure 14: Confusion matrix for the pruned 30% global sparsity model

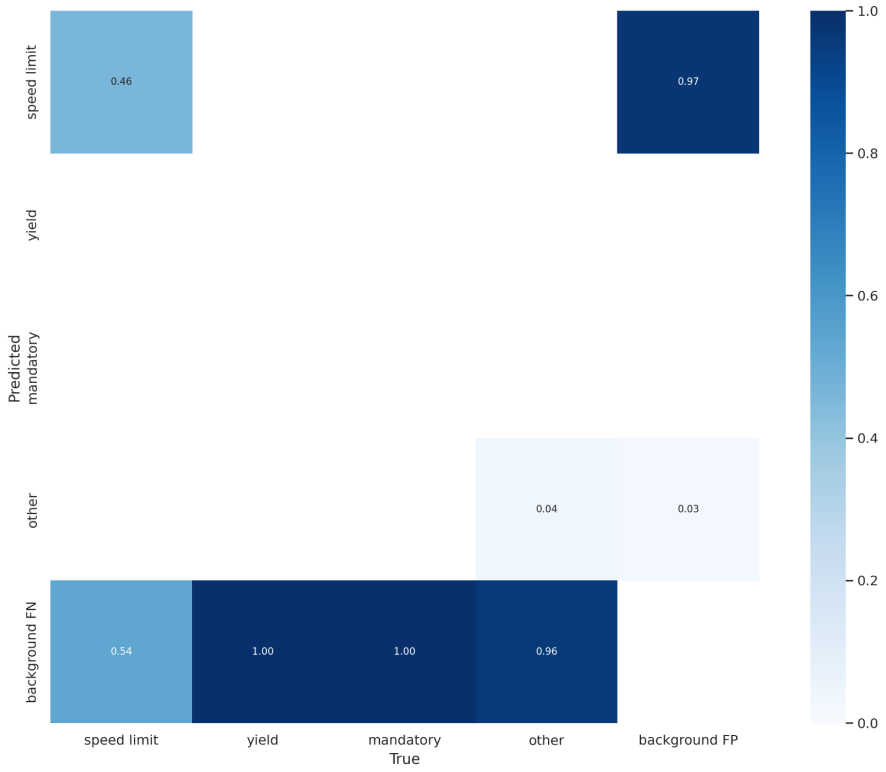

Figure 15: Confusion matrix for the pruned 50% global sparsity model

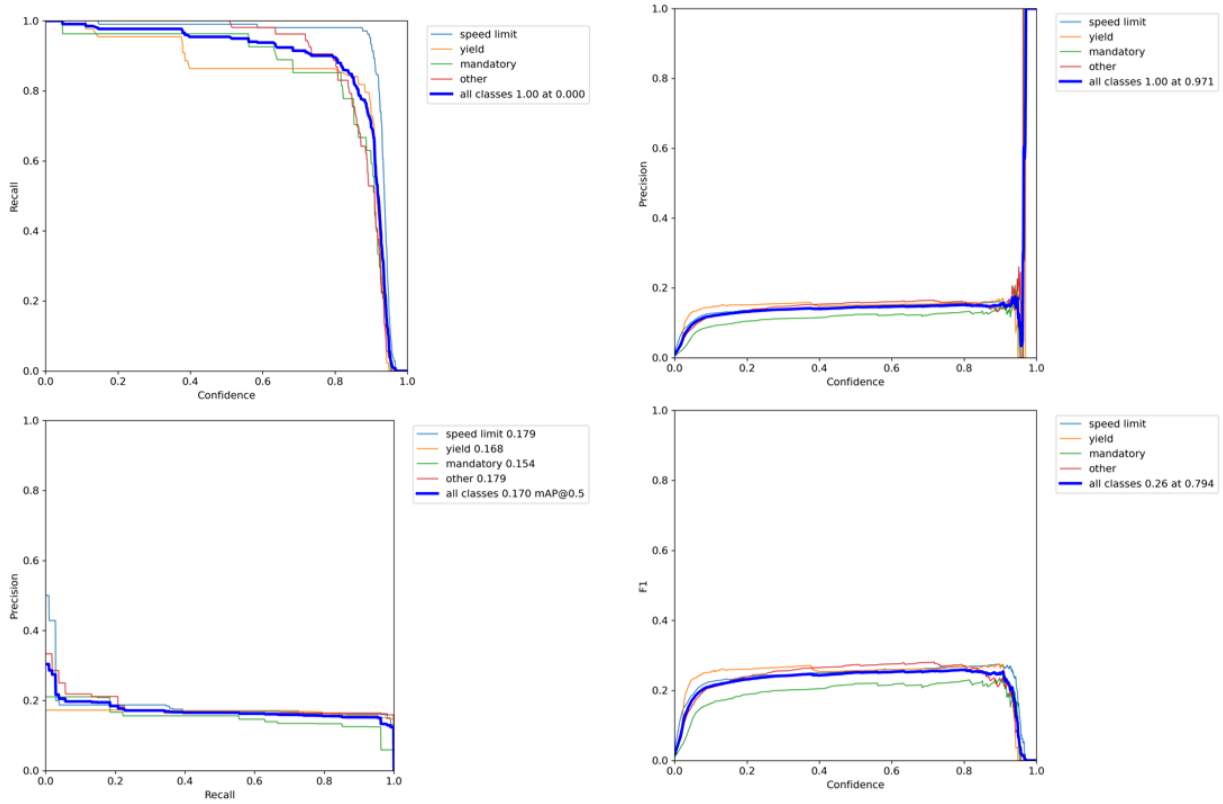

Figure 16: From top left going clockwise: Recall, Precision, Precision-Recall, and F1 curves of our pruned 30% global sparsity model

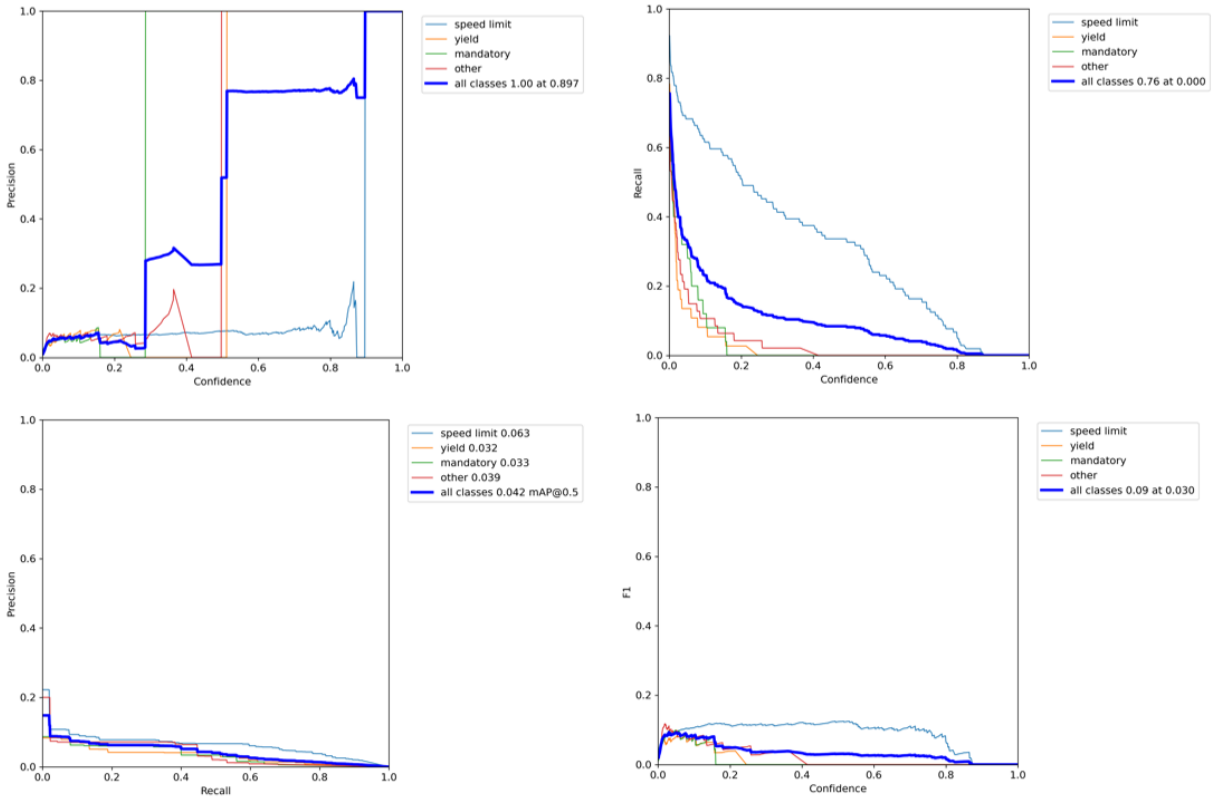

Figure 17: From top left going clockwise: Recall, Precision, Precision-Recall, and F1 curves of our pruned 50% global sparsity model