

Traffic Sign Detection Using Convolutional Neural Networks

Milán, Kriston; Döniz, Borsos

Óbudai Egyetem, Tavaszmező utca 15, 1084, Budapest, Hungary,
kmilan@stud.uni-obuda.hu, borsos.doniz@ui-obuda.hu

Abstrakt: In recent years, safety and driver assistance systems in vehicles have rapidly improved. Based on these results, my goal was to create a homemade solution for traffic sign recognition that is capable of real-time image processing and detects and classifies traffic signs in its environment, as accurately as possible. Considering my limited resources, not every single traffic sign is processed, only six of them were chosen but naturally, my solution would be scalable for every single sign. Based on my research, every single car manufacturing company uses the same Convolutional Neural Network-based approach but these solutions are not open-source. Firstly, in my presentation and study, there will be a short theoretical introduction. After this summary, I will elaborate on object detection as a wider area and the input data structure will be presented. The system's biggest flaws and the hardships caused by my hardware will be also covered in the study. In the end, the system's final efficiency will be evaluated and will be tested in a real-time environment using pre-recorded videos.

1. Introduction

The goal of this project was to create a system which is capable of detecting traffic signs in various conditions in real time. Considering the limited resources and time that I had, I decided not to involve every single sign in my solution, instead I choose randomly (but from the most common ones) six signs to detect. In the last couple of decades implementations involving artificial intelligence became widespread in numerous engineering fields. That is why after failing with "traditional" approaches I decided to create a system which is based on Deep Learning, hence Convolutional Neural Networks. The six signs which are detected are:

- No entry;
- No waiting;
- No stopping;
- Yield;
- Stop;
- Pedestrian crossing.

Naturally every other signs could be detected in the same manner as these signs that I choose. Every single data-oriented solution's biggest challenge is to create a data structure that is adequate for the task at hand. This is especially true for smaller projects without extensive manpower and financial resources. Another common hardship is the lack of proper hardware equipment, I had to overcome both issues in this project.

2. Object Detection, Environment

The problem can mostly be interpreted as an Object Detection task. During Object Detection we want to locate object in question with a Bounding Box and also classify it with the correct label and give a corresponding probability number to that class.

The solution I choose is a Transfer Learning approach, which means that I reuse a pretrained model's weights which was trained on a much larger dataset (COCO dataset) and just train the last layers with my significantly smaller own dataset. First I needed to set up the proper environment for the learning process, for that I was using the following tools:

- Tensorflow GPU 2.6.0 - Python API;
- Cuda Toolkit 11.2.2;
- CuDNN 8.1.1.33.

The hardware on which the learning process was carried out had the following most important parameters:

- GPU - NVIDIA GEFORCE GTX 960M - 2GB RAM;
- 8GB RAM;
- Intel(R) Core(TM) i5-6300HQ CPU - 2.30GHz

Tensorflow now offers various pre-trained models in Tensorflow Model-Zoo from which everybody can freely choose the most fitting for their application.

3. Data structure

The single most important aspect of every single Machine Learning project is the data that we feed the network. For my application I have collected all the input pictures with my Xiaomi Mi 10 Lite smartphone. I had to consider several aspects during data collection. Every single input picture has to have a 1:1 aspect ratio, therefore it is advisable to take pictures with this aspect ratio because even though we can resize any picture to that aspect ratio, the features of our object might be distorted in the process. This restriction is necessary because the Network expects pictures only in this exact ratio. It is also important to keep up the balance with the number of instances between the classes, otherwise a class might get under- or overrepresented. I solved this issue by keeping track of every single classes' instance number by collecting them to different folders. Multiple representation on a single picture might lead to smaller imbalances, but most pictures contained 1 or 2 objects therefore in the end the imbalance was negligible. While I was collecting these pictures I also tried to make these pictures from various angles, distances, lightning conditions and so on, to make my dataset as diverse as possible even before augmenting it. The final structure of my dataset looked like this before augmentation:

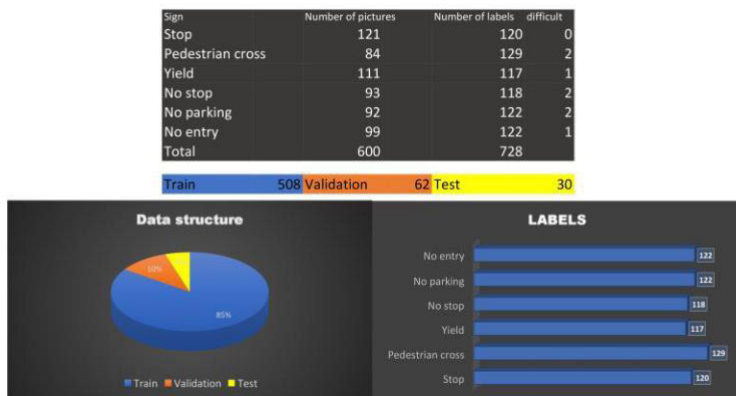


Figure 1: Data Structure

This instance number unfortunately is way too few to actually start the learning process therefore I experimented with various data augmentation techniques. These are also provided by the Tensorflow API, we don't have to manually apply every single technique therefore we can try them with different setups. The final techniques that I choose were the following:

1. Random_adjust_saturation
2. Random_adjust_contrast
3. Random_add_Gaussian_patch
4. Random_padding
5. Random_horizontal_flip

With these procedures I could enlarge my dataset by the factor of 5, 5.5.

For labeling my images I was using the free-source LabelImg program which provides different labeling formats, and is also capable of adding difficult and truncated attribute to the label. For my application I had to use the Pascal VOC format which stores every important detail about the label in an XML format. Unfortunately this application is not capable of handling EXIF files, therefore I had to rotate all of my input pictures because the orientation data could not be loaded. During the process of labeling we have to make sure we do not add labeling noise to the system which might occur when we create a bounding box with too much background or if we do not include some part of the object in question. The Pascal VOC specification [3] advises that in case of "too small" object it is better not the label it as true positive even though we might introduce false negatives with this approach. This is decided intuitively by the person labeling and might bias the final performance therefore later it would be advisable to evaluate the effect of this phenomenon with various labeling thresholds.

4. Training

For the training process I was using the Object Detection API's scripts with with few alterations. The training script in it's original form is not capable of monitoring the validation loss nor the Precision and Recall curves. That is why I decided to alter the script so that I would capable of monitoring all these parameters. In the beginning I did not change any hyperparameters for the model, started with the custom values. Since with these values I could not achieve adequate Precision and Recall scores I started tuning these parameters, most importantly the learning rate, but I was also experimenting with different optimizing algorithms both adaptive and time-based solutions. For training-time plots and evaluation I was using Tensorboard. Equipped with these tools I was able to notice possible indications for overfitting, inadequate learning rate and so on. Every time I noted such and indicator, I stopped the training and re-started the process with updates hyperparameters. It would possible to speed up this process with multiple GPUs so that I can train the model with various parameters and only kill of those that are certain not to converge into the right direction anymore. The model I was using in almost every single instance was the SSD Mobilenet V2, to be exact the one that was trained on 640x640 pictures. This model offers a fair balance between speed and power, tensorflow states that it is capable of running inference in 39 ms (that's naturally hardware dependent) and achieved 28.2 mAP

on the COCO dataset respectively. I was experimenting with different models from the Model Zoo but none of them achieved better performance in precision and the boost in speed was barely noticeable. This might have happened because I had not been tuning these models as accurately as the SSD-Mobilenet one. Therefore it might be reasonable to revisit these with spending more time on fine-tuning. The batch size I was using during training was only 4, because above the size 5, I was not able to start the learning process because I ran out of memory since my GPU only has 2GB. I would be able to increase this batch size if I resized the images before feeding into the system. Unfortunately I labeled all the images in their original, 3000x3000 size, so the model has to load in these high resolution pictures into the memory before resizing them, so another possible improvement would be re-labeling every picture in 640x640 resolution and using bigger batch size.

The evaluation metrics that I followed were the COCO metrics since the pre-trained model was also evaluated with these parameters.

5. Testing

The evaluation procedures are already aiming to monitor our model's performance, but I have been testing it in various conditions. I was running inference on single pictures, to visually confirm the results. Moreover I was calculating precision and recall values for the test set and because the model is expected to run in real time, I was using short self-recorded and resized videos in various lighting and weather conditions. My best trained solution was able to achieve 0.81 mAP and 0.85 mAR, considering how small the data structure and how weak my hardware environment is, this performance is acceptable. The highest FPS I was able to run the model with, was 12.1 and the average frame rate was between 9 and 10. It is important to mention that this solution struggles with smaller objects as every one-step detector. As for the large and medium objects, the performance was around 85-90 % or even higher, but as for smaller objects, it dropped to around 60% resulting in an overall performance around 80%. For this reason I would like to experiment in the future with YOLO-based architecture since some papers [13] indicate that these models perform better on small objects. During this project objects smaller than 32x32 pixel were treated as small objects, but that is also a threshold value which could be chosen differently.

Model	Learning Rate	Time-based change	Optimization	Step	mAP(0.5:0.95)	mAR(0.5:0.95)	F1 score
my_ssd_2	0.07999999821186066	Manual-step	Momentum	7000	0.717	0.78	0.75
my_ssd_3	0.05			10000	0.726	0.79	0.76
my_ssd_5	0.01	cosinus csökkenés + warmup	Adam	20000	0.723	0.77	0.78
my_ssd_8	0.08			15000	0.81	0.846	0.83
my_ssd_7	0.07999999821186066			20000	0.75	0.79	0.77

Figure 2: Evaluated models

On figure 2 the list of the most successful training results can be seen. As I mentioned before, I was testing a lot of different combinations of hyperparameters, I am not showing every result from every training process. One possible solution for the smaller objects' detection is that we limit the distance from which our system is surely capable of detecting. This exact distance is yet to be determined, as it requires further testing.



Figure 3: Testing on single image

Figure 3 shows an instance where the bigger sign in the foreground is easily detected, but the ones in the background with small size (and partially truncated), are not. Both during testing on single pictures and on videos the initial result indicated by the evaluation numbers were confirmed, the model named `my\_ssd\_8` performed the best in these tests. It was also noticeable (as expected) that models with higher mAP were less likely to detect false positives, and the ones with higher mAR more easily (from further, sooner) detected true positives.

6. Conclusion, further improvements

Considering the limited data and computing capacity I had I conclude the system to be adequate. Naturally there is still plenty of room to improve just considering the precision and recall values. This could be done by enlarging our dataset, more fine-tuning and possibly introduce distributed learning (learning on multiple GPUs) to be able to run training with more diverse parameters. I would not only like to improve my precision and recall values, but also expand my testing environment. I want to create an environment which is more similar to the real surroundings, for example using a dashboard camera instead of smartphone recorded videos. As I mentioned before another possible solution for improvement is switching model architecture to YOLO which might perform better on smaller objects. The easiest but most expensive way for better performance would be switching to better hardware because it would enable me to hasten the learning process. It would be also possible to use the TensorRT software development kit, which is capable of cutting the inference time in half with different optimizations resulting in possible double frame rate.

References

- [1] Aurélien Géron: Hands-on Machine Learning with Sickit-Learn, Keras & Tensorflow, Sebastopol, 2019
- [2] Introduction to Artificial Intelligence: Prof. Bojana Dalbelo Basic
- [3] Mark Everingham, Luc Van Gool, Christopher K.I Williams John Winn, Andrew Zissermann: The Pascal Visual Object Classas (VOC) Challenge
- [4] Johnathan Hui: mAP (mean Average Precision) for Object Detection, 2018.03.07, <https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173>
- [5] Mohamed Elgendy: Deep Learning for Vision Systems, New York, 2020
- [6] Vincent Dumoulin and Francesco Visin: A guide to convolution arithmetic for deep learning, 2018 January 12
- [7] Andrew G., Howard Menglong, Zhu Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam: MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, 2017 April 17., <https://arxiv.org/pdf/1704.04861.pdf>
- [8] Matthijs Hollemans: MobileNet version 2, 2018 April 22, <https://machinethink.net/blog/mobilenet-v2>
- [9] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg: SSD: Single Shot MultiBox Detector, 2016
- [10] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen: MobileNetV2: Inverted Residuals and Linear Bottlenecks, 2019 March 21, <https://arxiv.org/pdf/1801.04381.pdf>
- [11] Sebastian Ruder: Transfer Learning - Machine Learning's Next Frontier, 2017. March 27, <https://ruder.io/transfer-learning/index.html#fnref6:1>
- [12] Memory usage and computational considerations:
<http://imatge-upc.github.io/telecombcn-2016-dlcv/slides/D2L1-memory.pdf>
- [13] Nhat-Duy Nguyen, Tien Do , Thanh Duc Ngo, Duy-Dinh LeAn: Evaluation of Deep Learning Methods for Small Object Detection, 2020. April 27., Journal of Electrical and Computer Engineering
- [14] Ángel Morera, Ángel Sánchez, A. Belén Moreno, Ángel D. Sappa, José F. Vélez: SSD vs. YOLO for Detection of Outdoor Urban Advertising Panels under Multiple Variabilities, 2020. August 15.
- [15] Jonathan Dekhtiar, Bixia Zheng , Shashank Verma , Chetan Tekur: Leveraging TensorFlow-TensorRT integration for Low latency Inference, 2021. January 28.