Northeastern University

Computer Vision Methods on Traffic Sign Recognition

Authors

Allen Zhang

Muzhi Wu

Abstract:

Computer vision is an emergent technology and the heart of autonomous driving, furthermore, having highly accurate traffic sign recognition in self-driving cars is a prerequisite for practical deployment.There are many tasks that play into this problem, such as object detection, instance segmentation and most importantly image classification. Our final project tackles German traffic sign recognition with limited computing time and power, in the hope to figure out methods using accessible resources like google colab and dataset from kaggle to approach industry level of recognition accuracy.

Introduction:

1. The problem

According to a 2015 survey from National Highway & Traffic Administration, human error accounts for 94% of all car accidents, and a shocking 1.3 million related deaths around the world per year on average. This human error has been curbing down by the introduction of computer vision AI in autonomous driving. In order to advance technology and diminish deaths even further, we must figure out a fast, easily implementable neural network for modern vehicles.

2. Our Approach

The focus

Generally in computer vision tasks, accuracy and run time are the rule of thumb for the best model selection; however, since we want to create a simple set up that is easily accessible, we will also put an emphasis on implementation difficulty, then use all of those metrics to determine the best image classification setup.

Batch Size & Layers

Starting by surveying different machine learning models and techniques, we decided to stick with the convention of using CNN (convolutional neural network) as the foundation of our model, as there are no other suitable methods (e.g. ANN, RNN) with comparable performance at image classification. By feeding small batches of data into CNN, the convolutional layer will filter and leave the most important pixels, then Maxpool reduce the dimensionality even more, as a result, CNN has exponentially less connections compared to the traditional fully connected neural network. While CNN is a complicated model compared to the others, but the most influential parts are only the batch size and number of feature extracting layers (i.e. Convolution, Maxpool). Thus we will loop through different numbers to investigate the optimum set up for these two parameters.

Transfer learning

After allocating the best combination batch size and number of Layers, we will utilize transfer learning to improve our time performance even more. The essence is to borrow a large trained neural network to classify a dataset too small or save training time on classifying a large but related data set. In theory, the increase of training efficiency will be significant; however, this may come at a cost of accuracy, as this is a bi product of not training/tuning the feature extraction layer in the imported trained model.

Limitations:

Although we have a plethora of tools and techniques, there are a few hard barriers that limit our final model performance. First the data distribution is heavily skewed towards the circular & speed sign images, which in theory will impact the test accuracy with a lower than 100% asymptote. Second, due to limited resources, we could only use limited GPU and had no TPU boost, thus the time can be drastically increase with the right equipments. Lastly, the time limitation has bogged down the project as a whole, with more than an hour of training with 30 epoch on google colab, we did not progress as much as scheduled in the early stages.

Preliminaries:

1. Dataset

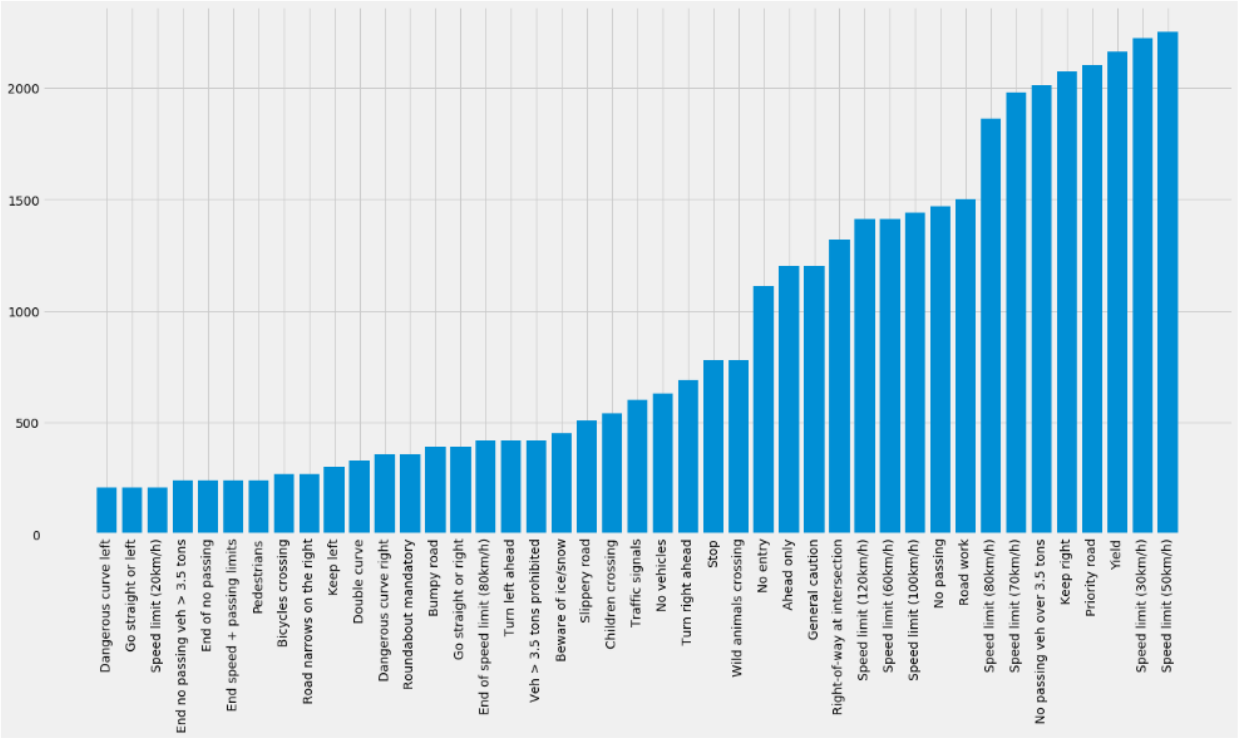
The German Traffic Sign Recognition Benchmark (GTSRB) contains 43 classes of traffic signs, split into 39,209 training images and 12,630 test images. The images have varying light conditions and rich backgrounds. GTSRB is a dataset with a direct real-world scenario application, i.e., classifying traffic signs. Images in this dataset are instances of physical traffic signs, with each real-world traffic sign occurring only once.

There are several reasons for choosing this dataset over others, including the fact that it is highly accepted and used to compare traffic sign recognition methods in the literature. In addition, its authors and the organizations behind them held an open competition where scientists from different fields contributed their results and tested the GSTRB dataset, their graphs and test results are informative and can be a reference to our project.

The integrity of a model depends on the data quality. Despite the great advantages, there are still problems that need to be addressed. Starting by counting the sample size for each class, we noticed a skewness to the speed limit and circular signs, which could be influential to the classification due to the weight and probability adjustment during the gradient descent phase in backpropagation.

so we used a random for loop method to quickly survey batches of different classes of images. Inferring the preliminary result, the images are nicely taken with decent resolution and well centered. However, it is often found that the sign is almost indistinguishable from the background. Furthermore, the images are all in different sizes, which means we need to

standardize the data set by setting a reasonable dimension to be fed into the convolution neural network later on.



(Figure 1. Exploratory data analysis on the dataset)

2. Setup

Our experimental equipment uses Windows 10 as the platform, Intel Core i7 CPU, and NVIDIA GeForce GTX 1070 Ti GPU. We initially used Google Colab as our IDE; however, due to the drawback of time performance, we had to switch to jupyter notebook midway as a faster alternative for running keras/tensorflow. We implemented the renown ResNet50 & VGG16 as our transfer learning trained CNN models. Lastly, the time performance is recorded in the jupyter notebook as a reference.

Results:

Optimizer	Batch_size	Run_time(s)	Accuracy(%)
GradientDescent	16	142.88	43.13
GradientDescent	32	128.54	28.19
GradientDescent	64	122.40	18.23
Adagrad	16	147.65	27.33
Adagrad	32	130.75	30.46

Adagrad	64	123.33	26.63
Adam	16	150.78	81.13
Adam	32	131.40	80.30
Adam	64	123.03	79.70

(Table 1. Adagrad = Adaptive gradient descent, Adam = Adagrad + RMSProp)

When we use GradientDescent as the optimizer, we test the model performance when using different batch sizes. We tested batch sizes of 16, 32 and 64. As the batch size increases, the run_time keeps decreasing. The accuracy varies a lot. When the batch size is 16, the accuracy is 43.13%. When the batch size is 32, the accuracy is 28.19%. When the batch size is 64, the accuracy is 18.23%. Obviously, as the batch size increases, the accuracy keeps decreasing.

We also tested different optimizers for model training under the same batch size, which is 32. When using GradientDescent as the optimizer, total run_time is 128.54 seconds, and the accuracy rate is 28.19%. When using Adagrad as the optimizer, the training speed is not significantly different from the previous one. The total run_time is 130.75 seconds, and the accuracy rate does not change significantly, with an accuracy rate of 30.46%. When Adam is used as the optimizer, there is no significant difference in the training speed. The total run_time is 131.40 seconds, and the accuracy rate is significantly improved, with an accuracy rate of 80.30%.

Similarly, we also found that compared to using Adagrad and Adam as optimizers, different batch sizes have the same impact on training time and accuracy. The smaller the batch size, the higher the accuracy of the model and the longer the training time. After we modified our model, we adopted Adam as the optimizer and set the batch size to 32. The training time is 267.95 seconds and the accuracy is 98.40%.

Optimizer	**Batch_size**	**Run_time(s)**	**Accuracy(%)**
Adam	16	150.78	81.13
Adam	32	131.40	80.30
Adam	64	123.03	79.70
Adam*	32	267.95	98.40
Adam(VGG16)	32	116.16	44.22

(Table 2. Adam*= 4 Convolutional layer + 2 Maxpool & Batch Normalized CNN)

Discussion:

Although the choice of batch size can obviously affect the performance of the model, as the batch size decreases, the accuracy will increase, but it is more important to choose the correct optimizer. In comparison, Adam performed much better than GradientDescent and Adagrad. Adam is more suitable for computer vision classification. Also, we implemented transfer

learning to see if it will outperform our best model. However, due to a frozen feature extraction layer. The increase of time performance is not enough to compensate for the significant loss of accuracy, which was out of expectation.

Conclusion:

By testing and fine tuning different CNN parameters and trying different combinations of training techniques, our best model produced a test accuracy of 98%, while having a short training time of 168 seconds, surpassing most submissions on the German Traffic Sign dataset. Our result validated the importance of batch size, normalization and convolutional layers in CNN. Furthermore, we have shown that transfer learning is not applicable in all scenarios, especially when dealing with a rural & specific type of image classification problem. Last, we have demonstrated that highly accurate CNN can be produced and improved upon using traditional methods of looping and combination. In the future we hope to use more advanced techniques and procedures to further push CNN's limits on Image Recognition and Autonomous driving.

References

1. https://pavisj.medium.com/convolutions-and-backpropagations-46026a8f5d2c
2. https://towardsdatascience.com/gradient-descent-with-momentum-59420f626c8f
3. https://www.v7labs.com/blog/transfer-learning-guide

Code Reference:

1. https://www.kaggle.com/code/lalithmovva/99-accuracy-on-german-traffic-sign-recognition
2. https://github.com/Arko98/Gradient-Descent-Algorithms/blob/master/README.md

Our Code:

https://github.com/AllenForReal/Traffic-Image-Processing-Computer-Vision-