CS230

Multi-Class Traffic Sign Classification using AutoAugment and Spatial Transformer

Hongbo Miao
hongbo.miao@stanford.edu

Labib Tazwar Rahman
labib@stanford.edu

Abstract

Fast and accurate classification of traffic signs is going to play a crucial role in autonomous driving, mapping/navigation, and traffic safety. Although this topic has garnered much attention in the computer vision research community, it remains a tricky challenge owing to perception difficulties due to variations in viewpoint and lighting conditions, motion-blur, physical blocking, and low-quality image [1]. Previous work relies on manually fine-tuning multiple parameters such as brightness and contrast. In this paper, we propose a novel combination of AutoAugment and Spatial Transformer Networks that performs better across a wide range of illumination and other real-world variables. This method automates the data augmentation process and achieves 99.86% validation accuracy while using fewer parameters and faster computation.

1 Introduction

The market size for autonomous driving technology is on the rise [2]. As more and more commercial vehicles are being released with driving assistant skills [3], automating traffic sign classification may be able to prevent some accidents resulting from drivers' inability to see and follow such signs at all times. Major obstacles in detecting and recognizing traffic signs include lighting conditions due to weather and time of the day, low-quality image at high speeds, obstruction by other objects as shown in Figure 1.

In general, traffic sign recognition includes two stages: traffic sign detection and classification. This paper focuses on the traffic sign classification stage. We are proposing an approach using AutoAugment and Spatial Transformer to handle the illumination and visibility challenges for a more robust classification than previous methods.

Figure 1: Various real-life complexities in traffic sign images [1].

2 Related Work

Support vector machine (SVM) [4] and convolutional neural networks (CNNs) [5] are popular options in traffic sign classification. A combination of SVM and CNNs has also been shown to be effective

CS230: Deep Learning, Fall 2020, Stanford University, CA.

[6]. As traditional SVM-based methods require manual extraction of rich features and thus are time-consuming, there has been a shift towards CNNs [1]. These recent methods [1, 7] are generally evaluated on the German traffic signs benchmark (GTSRB) dataset [8] by modifying ResNet, using YUV color scheme.

Instead of manually fine-tuning parameters such as saturation and brightness, we use AutoAugment to automate the data augmentation stage. By using a Spatial Transformer we will equip the network to deal with more complex situations while only using 92,397 parameters, which is also a significant reduction from previous methods given our high accuracy rate.

3 Dataset

In this paper, we use the GTSRB dataset [8] which contains 39,209 traffic sign images of 43 types grouped into six categories (Figure 2). The number in each class ranges from 210 to 2,250 (Figure 3). The size of images in the dataset varies between 15×15 and 250×250 pixels.

Figure 2: Six categories of traffic signs sample images [9].

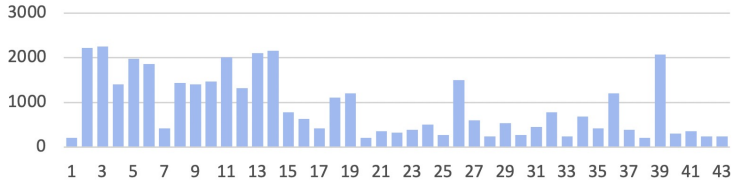

Figure 3: The number of traffic sign photos in 43 classes.

4 Methodology

4.1 Data Preparation

Because the size of images varies, we either down-sampled or up-sampled to 32×32, which is the minimum size at which most humans can identify most of the types of traffic sign photos. Additionally, as shown in Figure 3), the number of photos in each class ranges from 210 to 2,250, so some classes lack enough data. Thus, we randomly perturb images according to the following factors initially during the data preparation:

- Scale (range of cropped origin size is between 0.9 and 1.1)
- Ratio (range of the cropped origin aspect ratio cropped is between 0.75 and 1.33)
- Rotation (-15 to 15 degrees)
- Brightness (PyTorch brightness factor 0.5)

- Contrast (PyTorch contrast factor 0.5)
- Saturation (PyTorch saturation factor 0.5)
- Hue (PyTorch hue factor 0.1)

This is our initial effort to make our model more robust before our experiments.

We split the training data into our training set and validation set into 80% and 20% respectively. The training set includes 31,367 images, while the validation set has 7,842 images.

4.2 Architecture

We use the neural network in the official PyTorch tutorial [10] as the baseline network.

After experiments and iteration, our final network architecture is shown in Figure 4. The network layer type with output shape and number of parameters can be found in Appendix A.

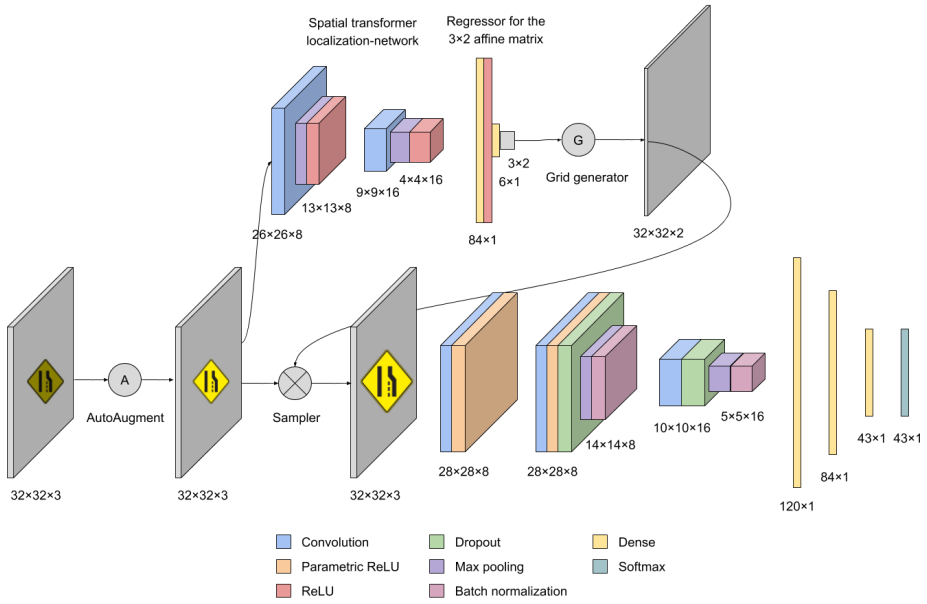

Figure 4: The architecture of our network.

As our loss function, we use categorical cross-entropy in the following form:

$$\text{Loss} = -\sum_{i=1}^{\text{N}} y_i \cdot \log \hat{y}_i$$

where $\hat{y}_i$ is the i-th scalar value in the model output, y_i is the corresponding target value, and N is the number of scalar values in the model output.

5 Experiments and Results

5.1 Hyperparameters

Learning rate: To speed up our training, the first experiment we did was finding the better learning rate. When training a model, allowing the learning rate to vary helped reduce the overall training time and improve the numerical optimal solution. We used Adaptive Moment Estimation (Adam) as our optimizer. For learning rate, we tested 0.1, 0.01, 0.001, and 0.0001. When the learning rate is 0.1 or 0.01, our cost function does not converge to an optimal solution or diverge. 0.0001 cause slows down the training too much, which makes us unable to test other factors efficiently. 0.001 is the best option we found.

Batch size: As the batch size is preferably greater than the number of image classes, which is 43 in our case, to ensure maximum hardware usage and thus a lower training time, we initially set the batch-size 512, which is the highest possible. However, in our ResNet experiments, this batch size caused out-of-memory issues. Thus, to be consistent, we use 128 as our batch size for all experiments.

Number of epochs: For our baseline model, we started with epoch 50 as nearly none of our metrics improved after 50 epochs. However, as our model became more complex, it was rare to observe overfitting, so we increased it to 200 for all final experiments and benchmarks.

5.2 Experiments

5.2.1 Wide Range of Illumination and Imbalance of the Images

During our data preparation and early experiments, around 30% of images were too dark for identification, so we figured that using human-level Bayes error rate will not work. So we tried ResNet-50 which yielded an 99.78% accuracy. Our baseline network reached 98.53%, indicating room for bias reduction.

We identified the top five classes causing the high error rate. We have two situations here: 1) "20 Speed", "Attention Bikes", "Attention Bottleneck", "Attention Traffic Light", consistently showed 4% lower accuracy than others, which are around 95% and 2) the accuracy of the class "Attention Curvy" fluctuates between 95% and 100%. After checking those images, we spotted two possible reasons for this: 1) Most of these five classes have a wide variance in illumination and 2) "20 Speed", "Attention Bikes", "Attention Bottleneck", and "Attention Curvy" lack enough data.

By changing PyTorch brightness, contrast, saturation to 0.5, and hue to 0.1, the accuracy of these classes bumped from around 95.0% to around 98.50%, while not downgrading others.

AutoAugment: Instead of continuing manual-tuning of transform factors, we introduced AutoAugment. The idea is that given a model and a target dataset, AutoAugment uses reinforcement learning to automate the search for optimal image transformation policies, e.g., horizontal/vertical flipping, rotation, changing color, brightness, contrast of an image, etc. Using a controller RNN, AutoAugment samples an augmentation policy at a time. Then it trains to convergence a child network with a fixed architecture. This gives the validation accuracy as a reward to update the controller so that it can improve its policy generation over time. Instead of spending a large amount of time to train on our dataset to find the best policy, we use the policy found on ImageNet as ours.

Figure 5: Image transformation results. From left to right are original images, our fine-tuned transformation results, AutoAugment results.

As Figure 5 shows, AutoAugment results are much better for low-light images as it increases the underexposed parts of the images while decreasing the brightness of the overexposed parts. This further helps us increases the accuracy of the aforementioned classes from around 0.4% to around 99.0% while not downgrading others.

5.2.2 Bias Reducing

To address our issue of high bias, we added dropout layers in our neural networks to prevent overfitting the training data by dropping out neurons. This forces our model to avoid relying too much on a particular set of features. As a recommendation, if n is the number of hidden units in any layer and p is the probability of retaining a unit, a good dropout net should have at least n/p units ([11]). In our model, we have three convolutional layers followed by three fully-connected layers. We added the dropout layer after the second and third convolutional layers, and another one after the first fully-connected layer. We did not apply it to other layers because we think the information in the beginning and the end of the network is more important for the neural network. After experimentation, we found 0.5 to be a decent dropout rate.

We added Parametric Rectified Linear Units (PReLUs) for the first two layers. This allows the neural network to find the predetermined slope by itself. It achieves higher accuracy by improving the network fitting with nearly zero extra computational cost and little overfitting risk.

Moreover, we added a 1×1 convolutional layer after the 3×3 convolutional layer to have non-linearity, which allows the network to learn the more complex functions [12].

These methods helped us increase the accuracy to around 99.30%.

Spatial Transformer: As AutoAugment made inputs more complex, we added Spatial Transformer instead of using the traffic sign coordinates in the image provided by the dataset. This was done to enable spatial manipulation of our data within the network and to help the network learn invariance to translation, scale, rotation, and more generic warping. As Figure 6 shows, the spatial transformer learns to focus on the traffic sign during the training.

Figure 6: Left side are original images.
Right side are the output of the spatial transformer.

After all these experiments, our best result so far was 99.32% of the training set accuracy and 99.86% of the validation set accuracy.

5.2.3 Comparison with ResNet

We tested ResNet-34, ResNet-50, ResNet-101, ResNet-152 with our fine-tuned transforms and AutoAugment learned on ImageNet (Table 1) along with the baseline network and our final network. After 200 epochs, ResNet-101 had the best validation accuracy, even higher than ResNet-152 which is a more complex network. We observed that the training and validation accuracy of ResNet-152 was still increasing slowly at the end. So we believe that ResNet-152 needs to train longer to learn all its parameters.

Moreover, contrary to what we expected, ResNet-34, ResNet-50, ResNet-101, and ResNet-152 have better results with our fine-tuned transforms than with AutoAugment learned on ImageNet. First, we rule out the reason that AutoAugment causes training and validation data mismatch because AutoAugment provide a better result on our network and help us reach higher accuracy. Second, when we added AutoAugment to our network, it took more epochs to find a better result. Third, after checking accuracy plots as well as the train and validation loss, the network is not overfitting. Based on these, we think if we train longer, it could potentially have similar or better results as the ResNet with fine-tuned transforms.

[1]CPU: 2.6GHz 6-core 9th-generation Intel Core i7 processor
[2]GPU: NVIDIA Tesla T4

Table 1: Network comparison.

		Train	Val	Time (CPU[1])	Time (GPU[2])
ResNet w/ AutoAugment	**ResNet-34**	0.9890	0.9964	-	-
	ResNet-50	0.9855	0.9960	-	-
	ResNet-101	0.9900	0.9966	-	-
	ResNet-152	0.9887	0.9958	-	-
ResNet	**ResNet-34**	0.9965	0.9974	(too slow)	(lost)
	ResNet-50	0.9946	0.9978	(too slow)	196m
	ResNet-101	0.9946	0.9980	(too slow)	224m
	ResNet-152	0.9905	0.9967	(too slow)	240m
Baseline network		0.994	0.9853	55m	61m
Final network		0.993	0.9986	166m	162m

It is worth mentioning that our model only has 92,397 parameters which consumes less computing resource compared to ResNet with more than tens of millions of parameters.

5.2.4 Error Analysis

We highlighted the worst five classes in precision, recall, and F1 score for both baseline and our final network. Both networks perform worst on "Attention Bikes". In fact, there is only one wrong label; however, due to the fact that "Attention Bikes" has only 54 in the validation set, the error rate was relatively high.

In the final prediction result of our final network, there were 11 classes with only has one labeled wrong. The rest of 32 classes had zero labeled wrong. Table 2 shows the classification comparing results between the baseline and our final network. The detailed comparison results for all classes can be found at Appendix B.

Table 2: Classification results.

	Baseline Network			**Final Network**		
	Precision	**Recall**	**F1 Score**	**Precision**	**Recall**	**F1 Score**
Macro Average	0.9822	0.9833	0.9827	0.9985	0.9980	0.9982
Weighted Average	0.9855	0.9853	0.9854	0.9986	0.9986	0.9986
Accuracy			0.9853			0.9986

Figure 7 compares the accuracy and loss between the baseline network and our final network.

The confusion matrix (Figure 8) shows our final network performed well for all 43 traffic signs.

6 Conclusion and Future Work

In this paper, we proposed a fast and high accuracy model for traffic sign classification. The proposed model first uses AutoAugment learned on ImageNet as our data augmentation method. Then it uses spatial transformers to help the model learn invariance to translation, scale, rotation and generic warping. Finally, the algorithm we proposed reached 99.86% validation accuracy. Compared to the ResNet, our network consumes less computing resource and save more training time.

Given our network performs well on the GTSRB dataset, as a next step, we want to try a bigger size traffic sign dataset. Once we switch to a bigger size dataset, these are some areas that we want to continue exploring:

Fast AutoAugment: One limitation of AutoAugment is that it is computationally expensive. To speed up the search time for effective augmentation policies, in the future we will try Fast AutoAugment which shows comparable performance on ImageNet [13].

Residual blocks: We want to borrow the idea of residual blocks from ResNet which might help solve the vanishing and exploding gradient problems and allow us to train deeper neural networks without a huge loss in performance.

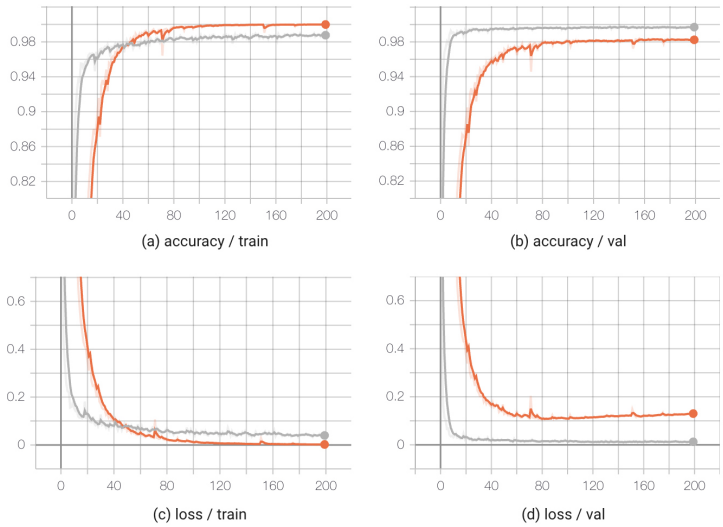

Figure 7: Accuracy and loss plots. Red lines are for the baseline network, while gray lines are for the final network.

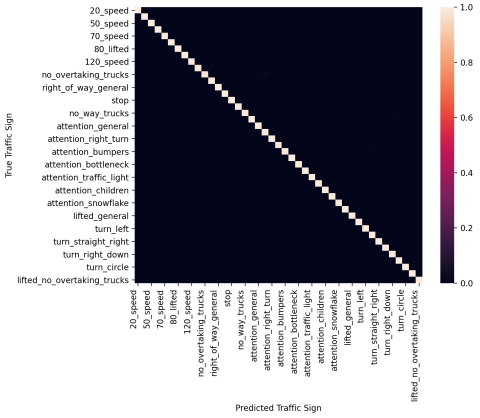

Figure 8: Confusion matrix of the validating result
(limited labels on axis are presented due to space limit).

7 Contributions

Hongbo Miao: Research, project (network prototyping, experiment, error analysis, visualization), paper writing (dataset, methodology, experiments and results, conclusion, appendix), slides, video.

Labib Tazwar Rahman: Research, project (experiment, error analysis), paper writing (abstract, introduction, related work, future work, appendix), paper review, slides, video.

8 Code

https://github.com/stanford-cs230-traffic-sign/stanford-cs230-traffic-sign.

References

[1] Pierre Sermanet and Yann LeCun. Traffic sign recognition with multi-scale convolutional networks. In *The 2011 International Joint Conference on Neural Networks*. IEEE, July 2011.

[2] Kwangyong Lim, Yongwon Hong, Yeongwoo Choi, and Hyeran Byun. Real-time traffic sign recognition based on a general purpose GPU and deep-learning. *PLOS ONE*, 12(3):e0173317, March 2017.

[3] Rongqiang Qian, Bailing Zhang, Yong Yue, Zhao Wang, and Frans Coenen. Robust chinese traffic sign detection and recognition with deep convolutional neural network. In *2015 11th International Conference on Natural Computation (ICNC)*. IEEE, August 2015.

[4] Xue Yuan, Xiaoli Hao, Houjin Chen, and Xueye Wei. Robust traffic sign recognition based on color global and local oriented edge magnitude patterns. *IEEE Transactions on Intelligent Transportation Systems*, 15(4):1466–1477, August 2014.

[5] Junqi Jin, Kun Fu, and Changshui Zhang. Traffic sign recognition with hinge loss trained convolutional neural networks. *IEEE Transactions on Intelligent Transportation Systems*, 15(5):1991–2000, October 2014.

[6] Yi Yang, Hengliang Luo, Huarong Xu, and Fuchao Wu. Towards real-time traffic sign detection and classification. *IEEE Transactions on Intelligent Transportation Systems*, 17(7):2022–2031, July 2016.

[7] Lihua Wen and Kang-Hyun Jo. Traffic sign recognition and classification with modified residual networks. In *2017 IEEE/SICE International Symposium on System Integration (SII)*. IEEE, December 2017.

[8] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In *IEEE International Joint Conference on Neural Networks*, pages 1453–1460, 2011.

[9] Jingwei Cao, Chuanxue Song, Silun Peng, Feng Xiao, and Shixin Song. Improved traffic sign detection and recognition algorithm for intelligent vehicles. *Sensors*, 19(18):4021, September 2019.

[10] Soumith Chintala. Deep learning with pytorch: A 60 minute blitz, 2017.

[11] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[12] Min Lin, Qiang Chen, and Shuicheng Yan. Network In Network, 2013.

[13] Sungbin Lim, Ildoo Kim, Taesup Kim, Chiheon Kim, and Sungwoong Kim. Fast autoaugment, 2019.

[14] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks, 2016.

[15] bkkm16. What we should use $align\_corners$ = false, 2019.

A Network Layer Info

Our network layer type with output shape and number of parameters are shown in Table 3. It includes a total of 92,397 parameters.

B Classification Results

Table 4 shows the classification comparing results between the baseline network and our final network.

C Spatial Transformer

Figure 9 shows the architecture of a spatial transformer module. The input feature map U is passed to a localization network which regresses the transformation parameters θ. The regular spatial grid G over V is transformed to the sampling grid $T_\theta(G)$, which is applied to U producing the warped output feature map V ([14]).

For our experiments, we use $align\_corners = False$ on functions $affine\_grid$ and $grid\_sample$ functions. It would allow both functions to be agnostic to the size of the sampled image when we are doing affine transformation When $align\_corners = True$, pixels are regarded as a grid of points where all the points at the corners are aligned. When $align\_corners = False$, pixels are regarded as 1x1 areas where area boundaries, rather than their centers, are aligned (Figure 10).

Table 3: Our network layer type with output shape and number of parameters.

Layer	Output Shape	Param #
Conv2d-1	[-1, 8, 26, 26]	1,184
MaxPool2d-2	[-1, 8, 13, 13]	0
ReLU-3	[-1, 8, 13, 13]	0
Conv2d-4	[-1, 16, 9, 9]	3,216
MaxPool2d-5	[-1, 16, 4, 4]	0
ReLU-6	[-1, 16, 4, 4]	0
Linear-7	[-1, 84]	21,588
ReLU-8	[-1, 84]	0
Linear-9	[-1, 6]	510
Conv2d-10	[-1, 8, 28, 28]	608
PReLU-11	[-1, 8, 28, 28]	8
Conv2d-12	[-1, 8, 28, 28]	72
BatchNorm2d-13	[-1, 8, 28, 28]	16
PReLU-14	[-1, 8, 28, 28]	8
Dropout2d-15	[-1, 8, 28, 28]	0
MaxPool2d-16	[-1, 8, 14, 14]	0
Conv2d-17	[-1, 16, 10, 10]	3,216
BatchNorm2d-18	[-1, 16, 10, 10]	32
Dropout2d-19	[-1, 16, 10, 10]	0
MaxPool2d-20	[-1, 16, 5, 5]	0
Linear-21	[-1, 120]	48,120
Linear-22	[-1, 84]	10,164
Linear-23	[-1, 43]	3,655

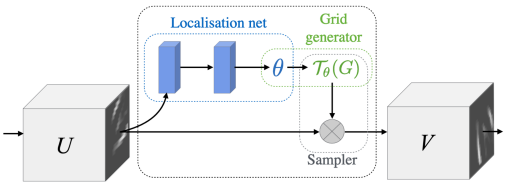

Figure 9: A spatial transformer module.

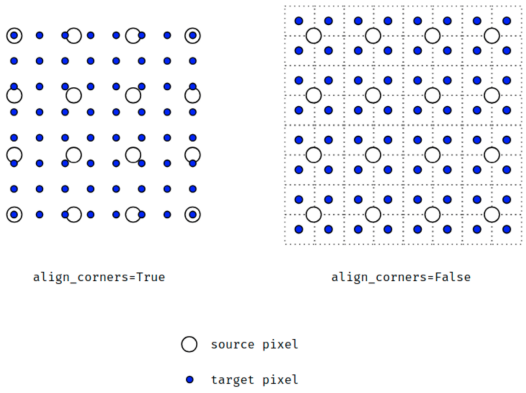

Figure 10: $align\_corners = True$ vs $align\_corners = False$ [15].

Table 4: Classification results.
The worst top 5 results for each precision, recall, and F1 score are marked as red color.

		Baseline Network				Final Network			
	Count	**Precision**	**Recall**	**F1 Score**	**# Mislabel**	**Precision**	**Recall**	**F1 Score**	**# Mislabel**
100 Speed	288	0.9825	0.9757	0.9791	7	1.0000	0.9965	0.9983	1
120 Speed	282	0.9858	0.9858	0.9858	4	1.0000	0.9965	0.9982	1
20 Speed	42	0.9512	0.9286	0.9398	3	1.0000	1.0000	1.0000	0
30 Speed	444	0.9820	0.9842	0.9831	7	1.0000	0.9977	0.9989	1
50 Speed	450	0.9800	0.9822	0.9811	8	0.9978	1.0000	0.9989	0
60 Speed	282	0.9556	0.9929	0.9739	2	0.9930	1.0000	0.9965	0
70 Speed	396	0.9949	0.9899	0.9924	4	1.0000	1.0000	1.0000	0
80 Lifted	84	0.9881	0.9881	0.9881	1	1.0000	1.0000	1.0000	0
80 Speed	372	0.9783	0.9677	0.9730	12	0.9973	0.9973	0.9973	1
Attention Bikes	54	0.9455	0.9630	0.9541	2	0.9815	0.9815	0.9815	1
Attention Bottleneck	54	0.9455	0.9630	0.9541	2	1.0000	1.0000	1.0000	0
Attention Bumpers	78	1.0000	1.0000	1.0000	0	1.0000	1.0000	1.0000	0
Attention Children	108	1.0000	0.9907	0.9953	1	1.0000	1.0000	1.0000	0
Attention Construction	300	0.9834	0.9900	0.9867	3	1.0000	1.0000	1.0000	0
Attention Curvy	66	0.9275	0.9697	0.9481	2	1.0000	0.9848	0.9924	1
Attention Deer	156	0.9745	0.9808	0.9776	3	1.0000	1.0000	1.0000	0
Attention General	240	0.9671	0.9792	0.9731	5	1.0000	0.9958	0.9979	1
Attention Left Turn	42	1.0000	0.9524	0.9756	2	1.0000	1.0000	1.0000	0
Attention Pedestrian	48	1.0000	0.9167	0.9565	4	1.0000	1.0000	1.0000	0
Attention Right Turn	72	1.0000	0.9306	0.9640	5	1.0000	1.0000	1.0000	0
Attention Slippery	102	0.9900	0.9706	0.9802	3	0.9903	1.0000	0.9951	0
Attention Snowflake	90	0.9457	0.9667	0.9560	3	1.0000	0.9889	0.9944	1
Attention Traffic Light	120	0.9573	0.9333	0.9451	8	1.0000	0.9917	0.9958	1
Give Way	432	0.9931	0.9931	0.9931	3	0.9977	1.0000	0.9988	0
Lifted General	48	0.9796	1.0000	0.9897	0	1.0000	1.0000	1.0000	0
Lifted No Overtaking General	48	1.0000	0.9792	0.9895	1	1.0000	1.0000	1.0000	0
Lifted No Overtaking Trucks	48	0.9796	1.0000	0.9897	0	1.0000	1.0000	1.0000	0
No Overtaking General	294	0.9898	0.9864	0.9881	4	1.0000	1.0000	1.0000	0
No Overtaking Trucks	402	0.9925	0.9925	0.9925	3	1.0000	1.0000	1.0000	0
No Way General	126	0.9688	0.9841	0.9764	2	1.0000	1.0000	1.0000	0
No Way One Way	222	1.0000	0.9865	0.9932	3	1.0000	0.9955	0.9977	1
No Way Trucks	84	0.9880	0.9762	0.9820	2	1.0000	1.0000	1.0000	0
Right Of Way Crossing	264	0.9776	0.9924	0.9850	2	0.9925	1.0000	0.9962	0
Right Of Way General	420	0.9929	0.9952	0.9941	2	1.0000	1.0000	1.0000	0
Stop	156	0.9936	1.0000	0.9968	0	1.0000	1.0000	1.0000	0
Turn Circle	72	0.9600	1.0000	0.9796	0	1.0000	1.0000	1.0000	0
Turn Left	84	0.9881	0.9881	0.9881	1	0.9882	1.0000	0.9941	0
Turn Left Down	60	1.0000	0.9833	0.9916	1	1.0000	1.0000	1.0000	0
Turn Right	138	1.0000	0.9783	0.9890	3	1.0000	1.0000	1.0000	0
Turn Right Down	414	0.9928	0.9928	0.9928	3	0.9976	1.0000	0.9988	0
Turn Straight	240	0.9958	0.9875	0.9916	3	1.0000	1.0000	1.0000	0
Turn Straight Left	42	1.0000	1.0000	1.0000	0	1.0000	1.0000	1.0000	0
Turn Straight Right	78	0.9872	0.9935	0.9935	1	1.0000	0.9872	0.9935	1
Average	182.4				2.9				0.3
Median	120				3				0
Total	7842				125				11
Macro Average		0.9822	0.9833	0.9827		0.9985	0.9980	0.9982	
Weighted Average		0.9855	0.9853	0.9854		0.9986	0.9986	0.9986	
Accuracy				0.9853				0.9986	