Machine Learning Engineer Nanodegree

Capstone Project

Rahul Tanwani

January 19, 2018

I. Definition

Project Overview

Traffic sign detection and classification is one of the preliminary requirement for autonomous driving to be useful in the real world context. Prior work in this field has mainly focussed on building the solution using image processing techniques, and hand coded features followed by a classification model[2][3] with few exceptions[4]. The relevant work in the field has been well summarized in [1]. With the recent advancements in machine learning, specifically in the areas of computer vision using deep learning techniques, we aim to build the robust traffic sign detection system that can accurately identify the traffic sign in real time without requiring the prior knowledge of traffic sign locations.

Problem Statement

Traffic signs recognition has direct real world applications in the autonomous driving and in driver assistance systems. Traffic signs are intended to be visible for drivers and have very little variability in appearance. Natual variations and calamities such as viewpoint variations, lightning conditions, sun glare, occlusions, physical damage and color fading etc contribute additional complexity and make the problem even more challenging. In this project, we propose to build the system for detecting the correct traffic sign given the complex natural images. Given we have more than 2 images, we will build the model for multi-class classification.

To build the solution, we will use deep learning techniques, specificically, the convolutional neural networks and define this as the computer vision problem. The details on the algorithm, overall solution, and the results obtained are summarized below. To gain the most of the solution, we have used the pre-processing techniques and data augmentation to build the robust model, that can handle the natural variations in the traffic sign images.

Metrics

There are multiple ways to evaluate the model performance for the multi-class classification model. The dataset used for this project comes from the German Traffic Signs Recognition Benchmarks (GTSRB) that uses CCR to rank the submissions. CCR is widely known as accuracy in the community. Though accuracy may gives us idea on the general model performance, it may not give very precise information for the imbalanced datasets. The GTSRB dataset is indeed the imbalanced dataset and hence we use other evaluation metrics in addition to accuracy. In specific, we would look at f1-score to incorporate both precision and reall balancing, and log loss. Furthermore, analyzing the confusion matrix would be very important to understand the class wise performance.

F1-Score:

Unlike accuracy, the F-Score is a balanced measure that takes into account both the precision and recall to compute the final score.

- Precision - is the measrure of how accurate the model is in predictions (True Positives / Total Predicted Positives)
- Recall - is the measure of capture / coverage (True Positives / Total Real Positives)

F1-score is the harmonic mean of precion and recall rates and is defined as:

$$F_1 = 2 \cdot \frac{1}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

Log Loss:

The general idea behind Log-Loss is to capture the goodness of the model not be based on the true predictions but also incorporating the confidence the model has on each of its predictions. Intutively, an ideal classifier is a one, that not only classifies all the samples correctly (accuracy 100%) but is also 100% confident in each of the predictions (log loss - 0). Mathematically, it is defined as:

$$-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} \log p_{ij}$$

- N = Number of samples
- M = Number of classes
- yij = 1 if j is the true label for sample i, 0 otherwise
- pij = Jth class probability of the classifier for sample i.

The notion of confidence is captured in terms of probability. This is good, as the neural network models (including ours) has the final softmax layer to get the probability distribution across each of the class.

II. Analysis

Data Exploration

The dataset used in this project comes from GTSRB - German Traffic Sign Recognition Benchmark, the traffic signs dataset hosted and maintained by INI benchmarks[5]. This dataset represent a multi-class classification challenge with more than 40 classes. The dataset has more than 50,000 images in total.

Dataset specifications:

- 43 traffic signs in total
- 51839 images in total
- The images contain one traffic sign each

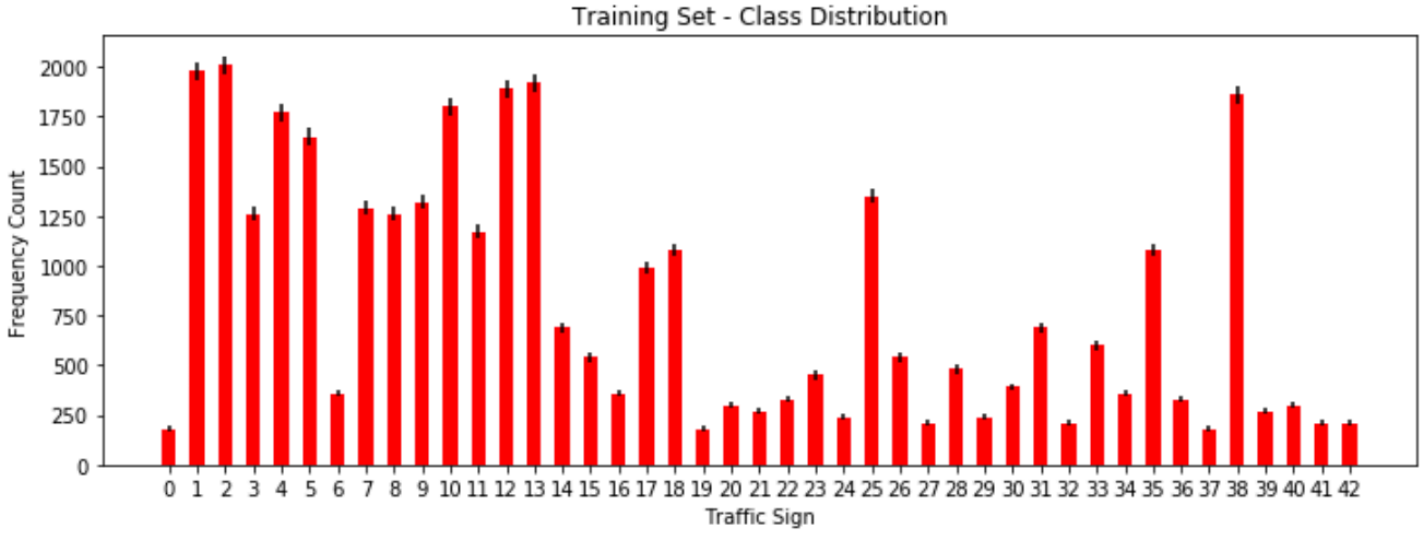
We used the dataset that was reshaped to 32x32 size each. The entire dataset was divided into training, validation and test sets.

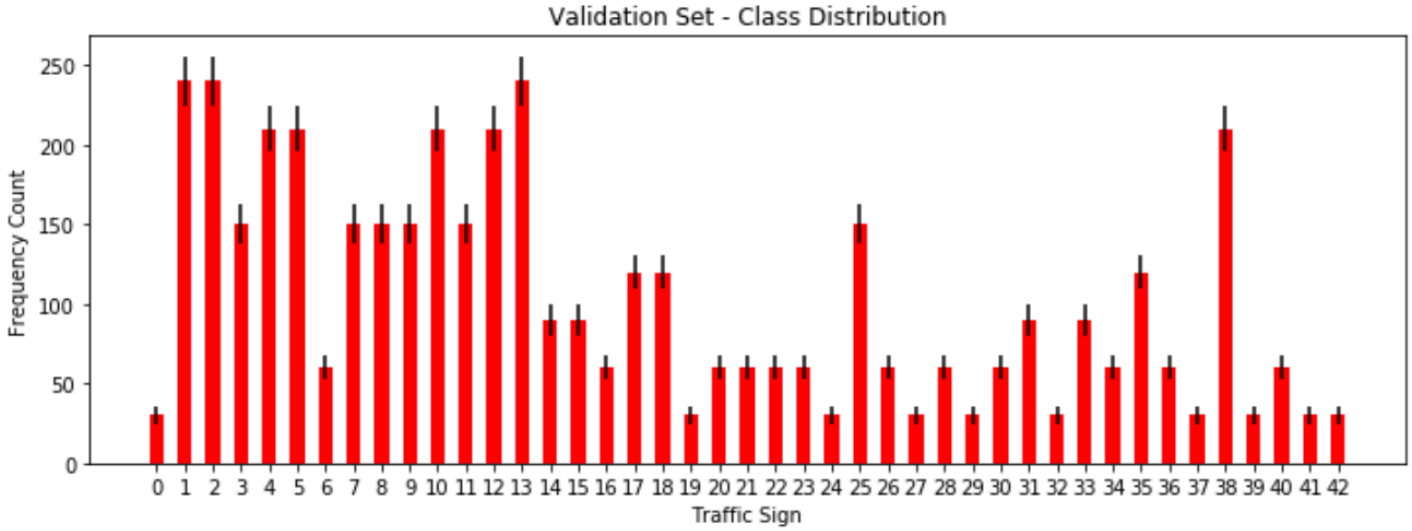
Data split:

- 34799 images for training
- 4410 images for validation
- 12630 images for testing
- 51839 images in total

Exploratory Visualization

Class Distribution (Training and Validation Data):





We can clearly see the distribution is uneven / imbalanced. This suggests we should pay attention on how to effitiently use the data for training and validation set, need for balancing and the effective measure for model evaluation.

Looking at the graphs above, we can make following observation:

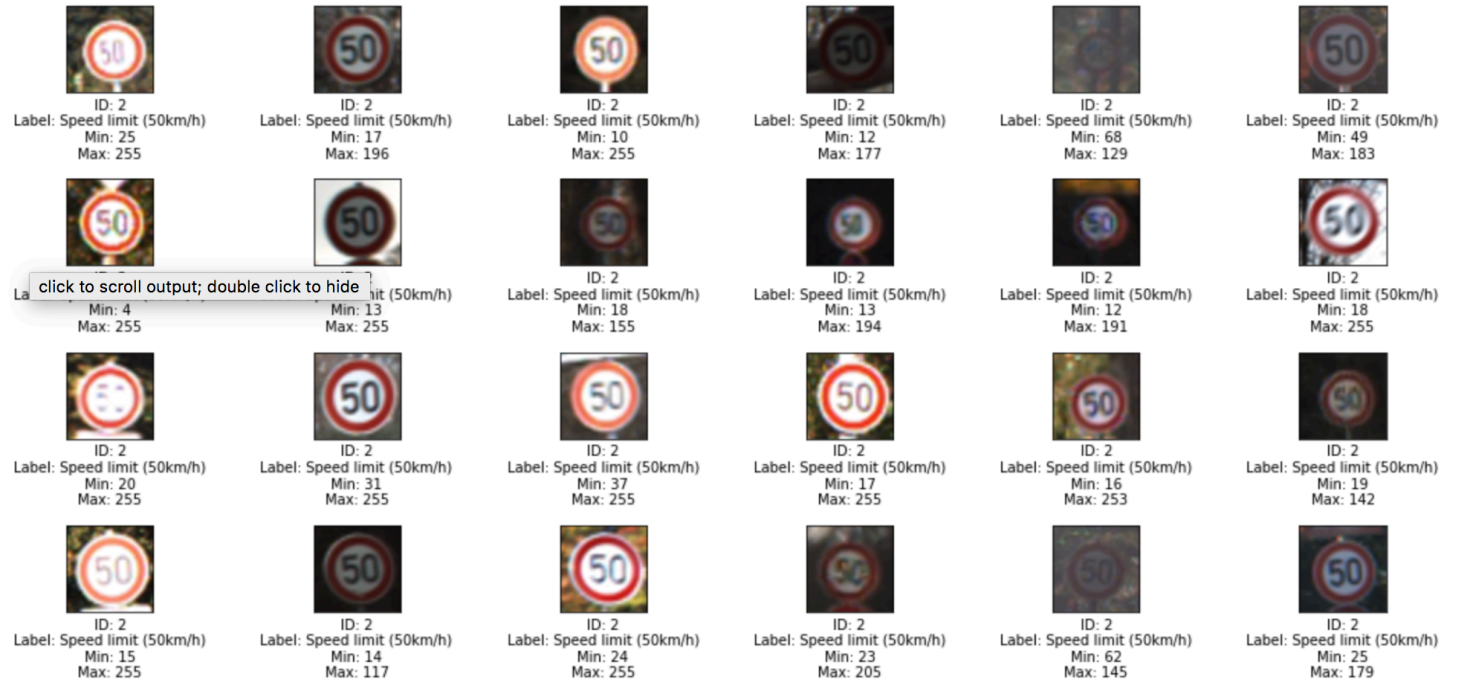
- The training set and the validation set distribution is similar
- All the classes are covered / represented in the validation set.

Sample Images:

The schematic above shows the various classes and corresponding images. Here, the min and max captures the minimum and maximum pixel value observed for the image. The values between 0 and 255 indicate we are using the 8 bits to capture the R/G/B channel values and that the images are are not normalized.

Let us analyze the images belonging to the same class to understand how similar / disimilar they look.

Variability for the same class:



Here, we analyze the different representations for the same class. We see that the traffic signs are captures with various lightning conditions, sun glare, size and horizontal & vertical placements. This also indicates that the test set would have different variations for the same class. This suggests, the model needs to be invariant and robust to these variations.

Algorithms and Techniques

Most of the prior algorithms and solutions for this problem have employed traditional image processing techniques, where the relavent features are hand-coded, and global signs are first detected using various heuristics and color thresholding techniques, the detected windows are then feed into the classifier for the final classification [2][3][6][7]. However, we will follow a learning based approach to effitiently learn the relevant features and detect the appropriate traffic signs. We plan to use the convolutional neural network based model to detect and classify the traffic signs.

Convolutional Neural Network:

Unlike the typical feed forward neural nets, the convolutional neural nets are specifically designed for the image data. In a staight FFN, each feature from the input is given to each of the nueros in the first layer, leading to fxn weights (f = feature dimensions, n = neurons in the layer) to be assigned and learned. This doesn't quite scale well when the input is an image. Even a low resolution image can typically be 200x200x3 pixel in size. This makes the input dimensions alone 120000. If we add many nuerons this number would quickly grow up. Too many weights pose the scalability challenge. We discuss below on how convnets deal with this challenge in an effitient way.

In addition to this, FFN doesn't preserve the spatial representation of the input which is very important when learing from the image. The pixel sourrounding a particular pixel has a spatial meaning and this must be preserved. Furthermore, FFN expects the input data to be a flat vector, which is very unnatural when we are working with images. Images capturing the physical objects are usually represented in 3 dimensions width x height x depth (color channels). Due to these challenges, FFNs are not very suitable for computer vision tasks in general.

CNNs instead of treating the input image as a flat feaure vector, they divide the image into regions and extract the usful information from those regions. These regions are designed to preserve the spatial information. With CNN, the general idea is that, the final image is built of multiple sub-elements which in turn are made up even smaller geometrical representations (Car -> (wheels, engine, doors, bonet), wheels -> (circular objects with some design in between)).CNNs learn this hierarchical representations by building multi layer convolution layer. Each conv layer acts on a region at a time (2d patch) and applies the series of filters which are passed to the next layer. The next layer builds on these first level features (extracted through filters) to learn the higher level abstractions / concepts. A typical CNN consists of following layers:

- **Convolution layer** applies the series of filters on a 2d patch of image giving us the output of 3 dimensions (patch*w * patch*h * d_filters). They are passed through activation function (usually relu) for non linear transformation by thresholding the output at 0.
- **Pooling layer** applies the downsampling operation by preserving only the most relevant aspects (max or avg pooling) of the patch, thus reducing the output dimensions by a factor of stride.
- **Fully connected layer** takes all the extracted features from the images and learns the representation to classify the final class from these features.

CNNs deal with the curse of dimensionality of weights by a concept called weight sharing. The intution behind weight sharing is that, since we are focused on learning the features, it doesn't matter where do they occur in the image. So far we can learn the weights to detect them in a patch, we can continue to use those weights for all the patches of an image. This greatly reduce the dimensions that the network has to learn. The dimensions now depend on the patch width, height and filters rather than the input width and height. The patches usually are very small as compare to the original image.

We build our final solution on the same concepts. The implementation below describe the details on the overall network arhitecture and individual layers to understand how these are being used in the context of traffic sign detection.

To make sure the model is robust, we will enrich the data with various data/image augmentation techniques before learning the model.

Benchmark

Before we dive deeper into building the solution, it was important to start from some base line and build on top. For this project, we used LeNet architecture as a base line model. The LeNet architecture was first introduced by LeCun et al. in their 1998 paper, 'Gradient-Based Learning Applied to Document Recognition'.

To build this model, we didn't do any data pre-processing. We used all 3 color channels for each of the images. **The benchmark model gave us an accuracy of 80% on the validation dataset**. After setting up the base line model, we made the following observations:

Observations:

The LeNet5 architecture was super fast as far training time is concerned. It also gave us pretty good accuracy of 0.8 from the get go. The LeNet5 architecture was objectively designed and architected to recognize hand written digits, which is quite different from the traffic signs recognition. We are going to improve upon this base model to increase the accuracy of the solution. To be able to do so, we will perform the following:

- Data pre-processing
- Deeper convolutional models

- Regularization using drop outs.
- Data augmentation.

It may also be important to know that the human accuracy for this dataset stands to be around 98.3%. Any machine learning model to be effective / useful in this task, should strive to beat this benchmark.

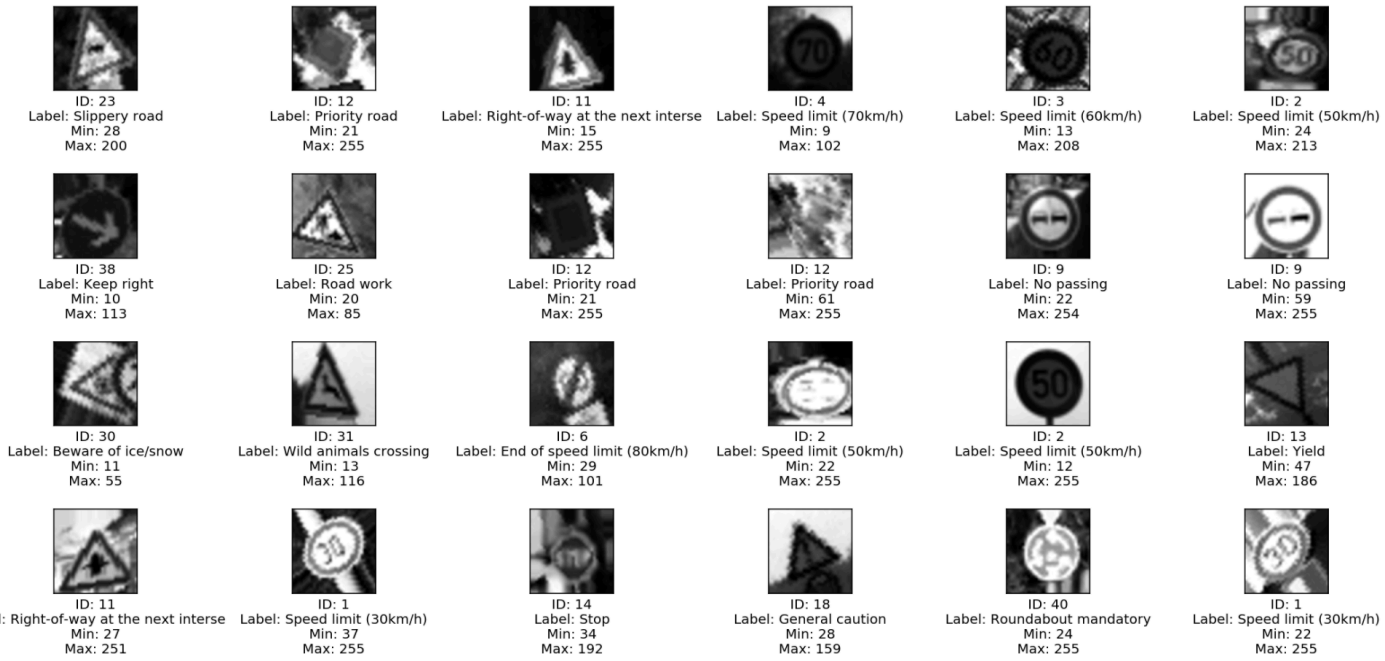
III. Methodology

Data Preprocessing

Data processing is one of the very important step in the life-cycle of machine learning problem solution. For this particular project, we did the following for data pre-processing:

Grey Scaling:

As we may notice from the images shared above, some traffic signs have different colors than the others. While images for classes like 'Keep Left', 'Ahead Only' are primarily represented in blue, the other classes like 'Yield' and 'Speed Limit' are primarily represented in red. Though colors may be somewhat helpful for the human eye, it may not be very useful for building the models in this particular task. This was also tested and concluded in [4] where adding all 3 color channels didn't improve upon the accuracy achived from grayscale images. For the model to be reliable, we want the models to learn the general patterns that define the traffic sign, rather than basing their predictions based on the colors. To make our models independent of the colors, we converted the RGB images into gray scale images. This also has the added advantage of lesser memory foot print. While the RGB image would need 32x32x3 bytes each, the gray scale images would be stored in 32x32 bytes, a reduction of x3. Sample images below show some of the images after gray scaling:



Normalization:

We performed a sample / image wise normalization as part of the pre-processing step. This is important to capture the relative intensity for each of the pixel.

Random Shuffling:

The data is not randomly shuffled by default. This would be a problem since we are doing a batch wise learning. For the learning to converge to the right parameters, it is important that all the classes are presented in each of the batch pass.

Implementation

To improve upon the base model, we start by building a deeper model with more convolution filters. This is important to learn the high level abstractions from the images, as described in detail by Yoshua Bengio in [10].

We built a network architecture with 3 convolution layers followed by 2 fully connected layers. Initially, without applying any drop outs in the architecture. The model didn't do as well as we may expect. We got an accuracy of about 82% on validation set. However, the quick observation was that, even though the training cost went close to zero, the validation didn't improve in accordance. After adding the drop outs, the peformance improved to close to 94% on validation set. This reinforces the importance of regularization when building complex / deep solutions. Table below show the architecture details after adding the drop outs.

Layer	Type	Activation	Drop Out
1	Conv - (5 x 5) @ 32 Max Pool - (3 x 3 / 2)	Relu	0.9
2	Conv - (5 x 5) @ 64 Max Pool - (3 x 3 / 2)	Relu	0.8
3	Conv - (5 x 5) @ 128 Max Pool - (3 x 3 / 2)	Relu	0.6
4	Fully Connected - (512)	Relu	0.6
5	Fully Connected - (192)	Relu	0.6
6	Output / Softmax Layer	Identity	

We first started with 2 conv + max pool layers with 32 and 64 filters respectively. However, given the number of classes we have in the dataset and their varied represention, we immediately felt that total of 64 filters may not be enough for the classifier to accurately predict the right outcome. We then added 3rd conv layer with 128 filters to capture extract more higher-level features from the data.

For the fully connected layer, we tried with various different shapes: (128 -> 128), (256 -> 256), (512 -> 192). These changes didn't have significant impact on the final scores. However, the (512 -> 192) architecture shape was marginally better than the other options we considered.

We also played with the varying drop out strategies. We started off with flat keep-prob of 0.7 for final conv layer and fully connected layer. We then switched to 0.6 which performed better. Later we added, drop outs to first two conv layers as well, however keep_prob=0.6 was too aggresive of a drop out for the initial layers given the limited filters. We then added progressive drop outs for each layer that gave us the best results.

Though accuracy of 94% is much better than the base model we started of with, it is still far from the human accuracy. Also, the GTSRB benchmarks shows the top performance of 99.8% on the same dataset. We will continue to build on top of the existing solution to improve the performance.

Refinement

Data Augmentation:

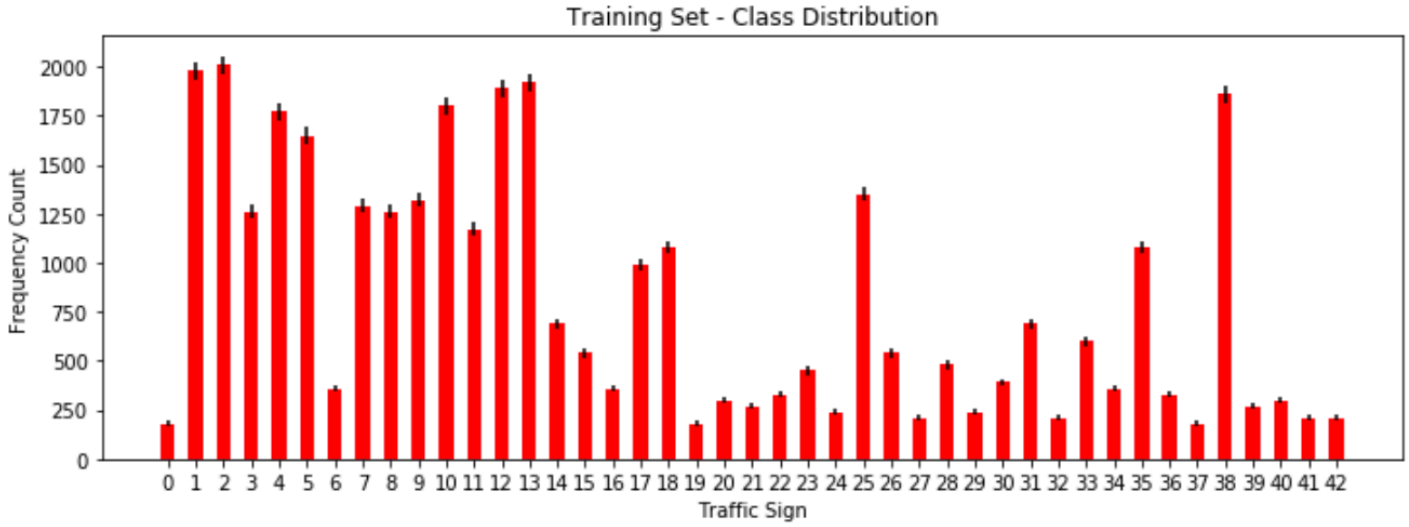
To make the model robust against the variations in images, we added about 60k augmented images with:

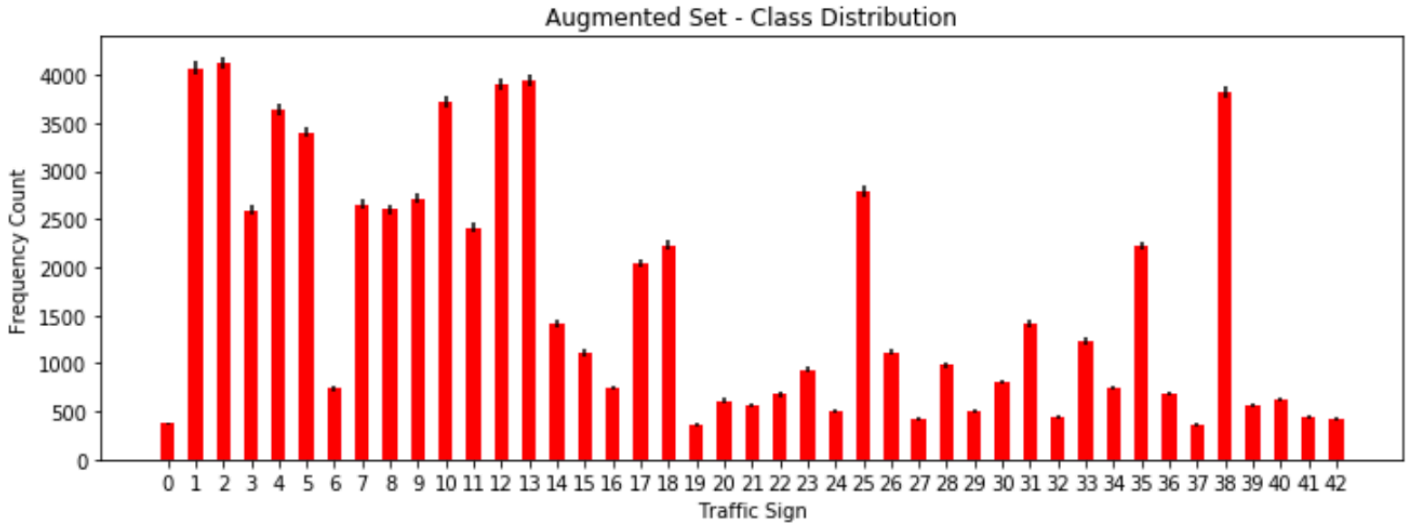
- Horizontal and Vertical shifts
- Rotation
- Shear

The picture below shows some of the augmented images for various classes.

It is also important to look at the class distirbution after augmentation and under how does it compare with the original class distribution.

Class distribution before and after image augmentation:





As we see above, the distribution after the augmentation is representative of the original distribution.

****Accuracy on validation set after augmentation: 96.2%****

Zooming and additional 10k images:

After seeing the impact of additional augmented data, we add additional 10k images. In addition to this, we also add the zooming transformation as we may expect to see the similar behaviour in the test data and also the real world.

****Accuracy on validation set after augmentation with zooming: 97%****

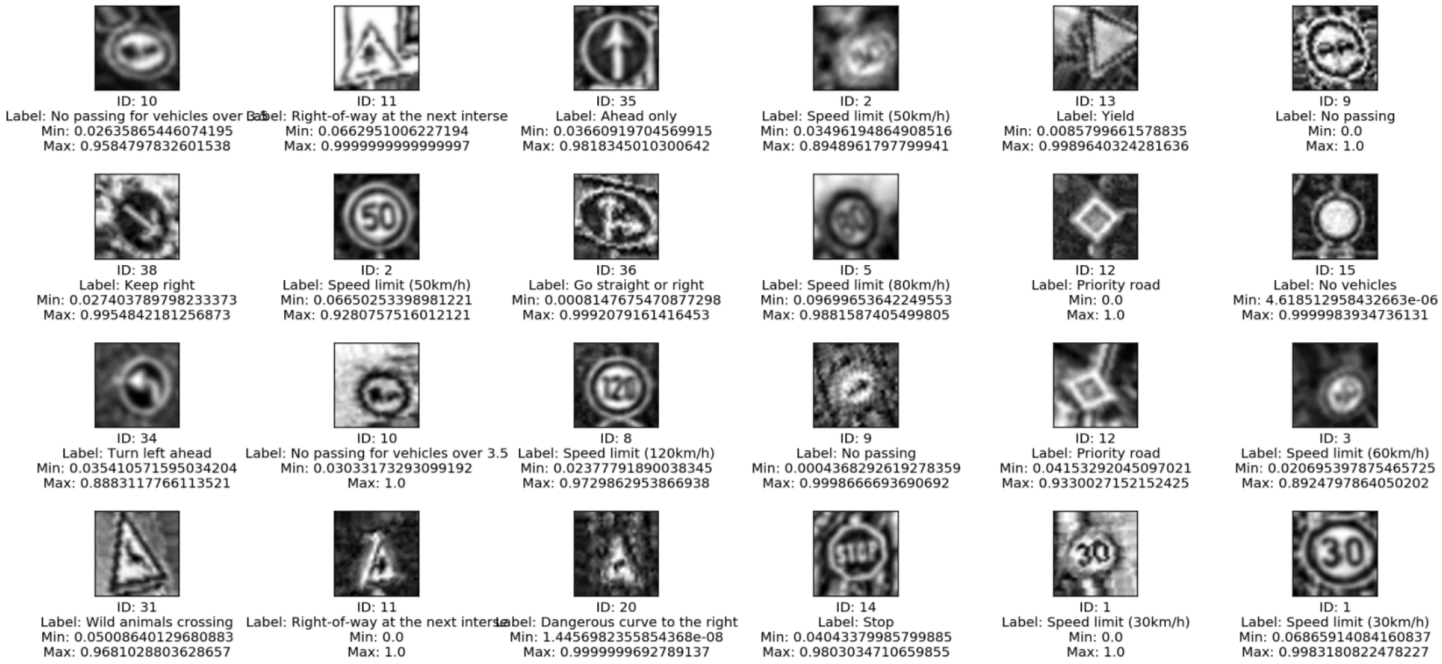
Image Contrast and Blurring:

Human perception is very sensitive to the image contrast as appose to the absolute luminance. The impact of image contrast for computer vision solution has also been recognised and discussed, as in [11]. We tried

multiple transformations for image contrasts, including:

- Histogram Equalization
- Contrast Stretching
- Adaptive Histogram Equalization (AHE).

AHE peformed better than the other two techniques. The image below shows some of the traffic signs after applying adaptive histogram equalization.



****Accuracy on validation set after constrast and blurring: 98.1%****

Multi-Scale features:

In a typical convolutions based neural network architecture, each layer is built on top the previous layer. The first layer learns the lower level entities - lines, circles etc, the second layer may learn the priliminary shapes like circle, ovals, squares etc, while the sub-sequent layers may learn the higher level abstractions useful to detecting relevant objects. The output of the final convolution layer is fed into the fully connected layer for the purpose of the classification. In such a setting, conv layers are typically used to extract the relevant features automatically from the images, so that classifier models can use those for effitiently classifying the outcome. The output of the final conv layyer is what the classifiers are fed for the final layer.

However, it may be useful to feed the lower level convolution filter outputs, in addition to the final layer, to the classifer. This has been suggested and discussed in detail by Pierre Sermanet and Yann LeCun in their solution to the same problem[4]. As is shown, using multi-scale features performed consistently better than

using the single scale features for the classifier.

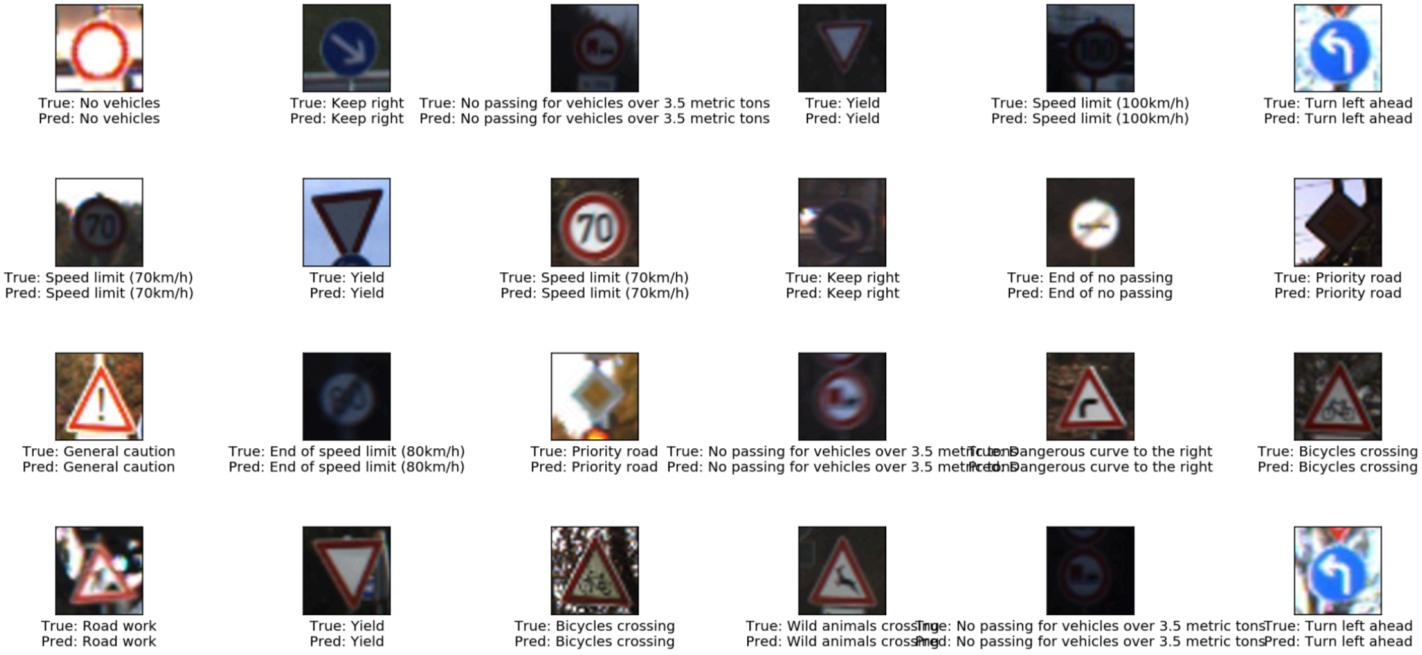
We used the similar approach to feed the output of 1st and 2nd stage after the pooling layer to the classifier in addition to the third layer filters. The first and second layer output has been pooled 2 times before it is fed into the classifier, as was followed in [4]. After making this change, the validation set accuracy reached to 99.22%.

Accuracy on validation set after adding multi-scale features: 99.22%

IV. Results

Model Evaluation and Validation

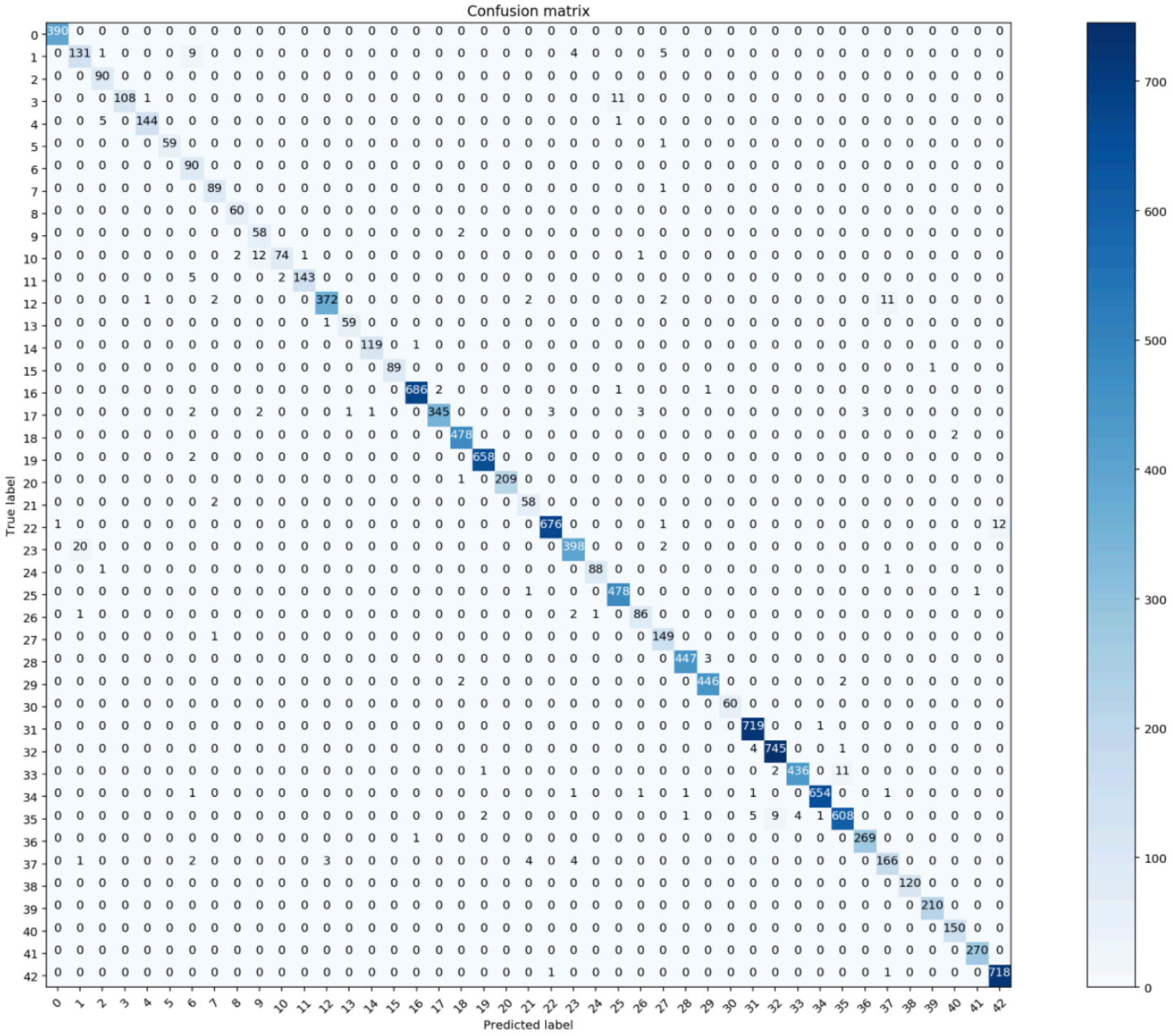
Using the solution as described in detail above, we got the **final test data accuracy of 98.14%**. The image below shows some of the sampled images from the test dataset with their true and predicted labels.



All the images shown above are actually the correctly classified data samples. Let us see the samples that were mis-classified to see if he could understand it better.

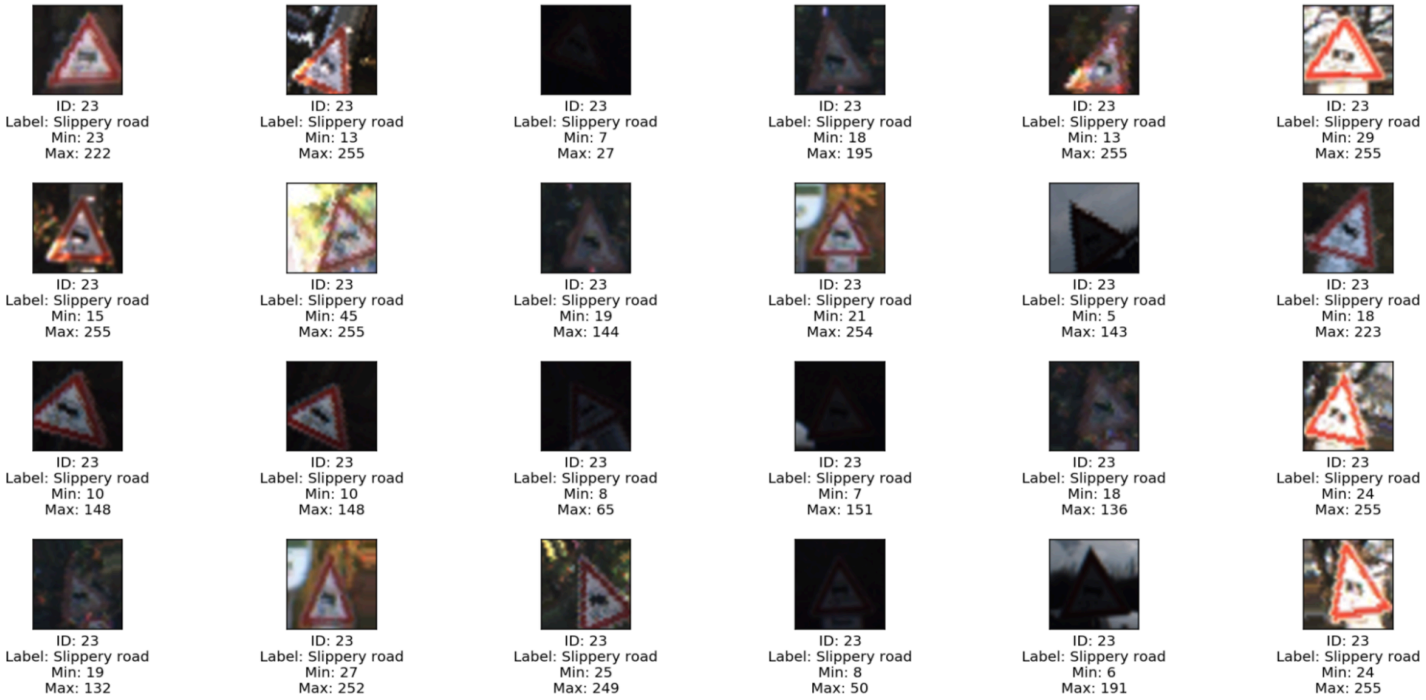
As we can see, some of these mis-classified images are indeed hard to classify. Some of them are occluded, while others are captured with very different lightning conditions.

Since we have an imbalanced dataset, lookig at the accuracy alone may not be the best idea to evaluate how good the model is. We also looked at the F1-Score for the model and we got the mean (unweighted) **F1-score across the classes of 0.972**. This is below the aaccuracy score, which indicates that not all the classes are predicted with equal performance. We then looked at the confusion matrix to understand performance for each of the class individually.



As we may expect, we mostly see the diagonal elements dominating the entire matrix. Though some of the classes in the original training dataset were not as well represented as others, we don't see any significant performance hit for those classes. This may be because, we generated the huge number of augmented images and after the augmentation, all the classes were significantly represented for the model to learn.

From the matrix above, we see that the performance for the label 23 needs to be looked into. We incorrectly classify it to be class 1 (Speed limit (30km/h)) for about 20 samples. The image below shows the traffic signs with label 23 (Slippery road) from the test dataset.



As we could see, some of the images are very difficult even for the human eye to confidently comment on the correct class for the sign.

Justification

The final accuracy of 98.14% on the test set and 99.22% on the validation set is significantly higher than the accuracy of the base model. However, as we discussed above, there are still the cases where the model mis-classified the traffic sign. Also, the benchmark results of 99.8% suggests that there is still the significant room for improvement. Some of the ideas to push the scores further are discussed below in the improvement section.

V. Conclusion

Reflection

In summary, we made a good progress on our solution from the initial base model we started with. Accuracy score on validation set jumped from 80% to 99.22% after all the solution refinements.

In the process, We realized the impact of regularization on the deeper and complex models. With adding in dropouts, we saw the jump of more than 10% accuracy on the exact same model.

The impact of data augmentation on the overall solution has been immense. It allowed us to represent some of the variations in the original data, to make it available for the model to learn from. This helped us achieve final

5% improvement in accuracy score.

Though 98.14% accuracy is not bad, we should keep in mind that for the automonous vehicles to be really effective and useful, they need to be able to recognize the traffic signs with 100% accuracy. The cost of recognizing a traffic sign wrongly would be too high and hence the performance needs to be pushed to close to 100%. However, given the work in the fields and recent advancements, we should be able to reach very close to this requirement. In the section below, we discussed some of the ideas to be able to get there.

Improvement

One of the very important aspect to improve further would be, to make sure that the test dataset is actually represented in the training set. Any exclusivity in the test set may lead to model mis-classification.

In additon to this, Smart Augmentation[9] suggests multiple techniques to improve upon the simple augmentation. It may also be important to automatically learn the required augmentations for the model to be effective.

Spatial Transformer Networks (STNs)[13] is another important idea to consider for improving the accuracy further. Some of the solutions using STNs on the same dataset published the accuray of more than 99% on the test data without any data augmentation.

Batch normalization could also be used for the models to converge much faster and save upon the training time as described in [14].

References:

1. Detection and Recognition of Road Traffic Signs - A Survey
2. Road and traffic sign detection and recognition
3. Traffic Road Sign Detection and Recognition for Automotive Vehicles
4. Traffic Sign Recognition with Multi-Scale Convolutional Networks
5. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition
6. Automatic detection and classification of traffic signs
7. Traffic sign shape classification evaluation I: SVM using distance to borders
8. The effectiveness of data augmentation in image classification using deep learning
9. Smart Augmentation - Learning an Optimal Data Augmentation Strategy
10. Learning Deep Architectures for AI
11. Understanding how image quality affects deep neural networks
12. On the limitation of convolution neural networks in recognizing negative images.
13. Spatial Transformer Networks

14. Batch Normalization - Accelerating Deep Network Training by Reducing Internal Covariance Shift****