Udacity Self-Driving Car Nanodegree - Project 2
Traffic Sign Classifier

Mark Moyou

March 26, 2017

1 Introduction

The purpose of this project was to design a deep neural network to do classification of German traffic signs. By going through this exercise, one would gain familiarity with large datasets, network architecture, optimization algorithms, hyperparameter selection, model generalization, internal model state visualization using the Tensorflow library [1]. The remainder of this report details the datasets used, the architecture, training of the network and classification accuracy. The (Section: *) refers to a numbered section in the code notebook.

2 Dataset Summary and Exploration

The German Traffic Sign dataset was chosen to train this neural network [2]. The dataset consists of 34799 training images, 4410 validation images, and 12630 testing images spread over 43 classes (Section: 1.1). Each image is 32 by 32 pixels and is a standard 3 channel color RGB image. As seen in Figure 1 and Figure 2, the distribution of the number of exemplars per class is almost exactly the same. This means that we can expect our accuracy to be relatively high because the network would have trained well on the classes that contained more data, and the corresponding test data would have a higher weight in the accuracy calculation. So, the accuracy score is not a good measure to use in this case, something like the F1 score which takes into account the number of samples per class would give a much better representation of the classification. The accompanying notebook (Section: 1.4) contains a table with the detailed class names and their number of samples per class in the training set.

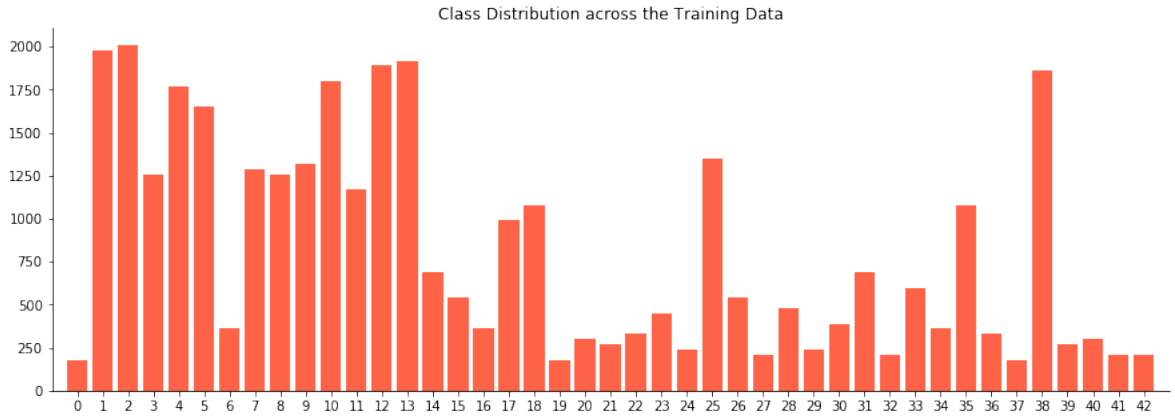

Figure 1: Training Set Class Distribution (Section: 1.3)

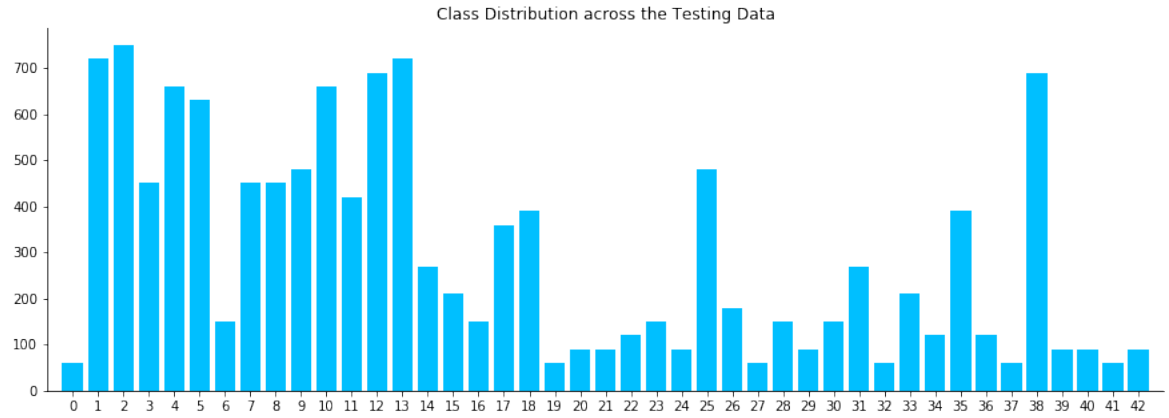

Figure 2: Testing Set Class Distribution (Section: 1.4)

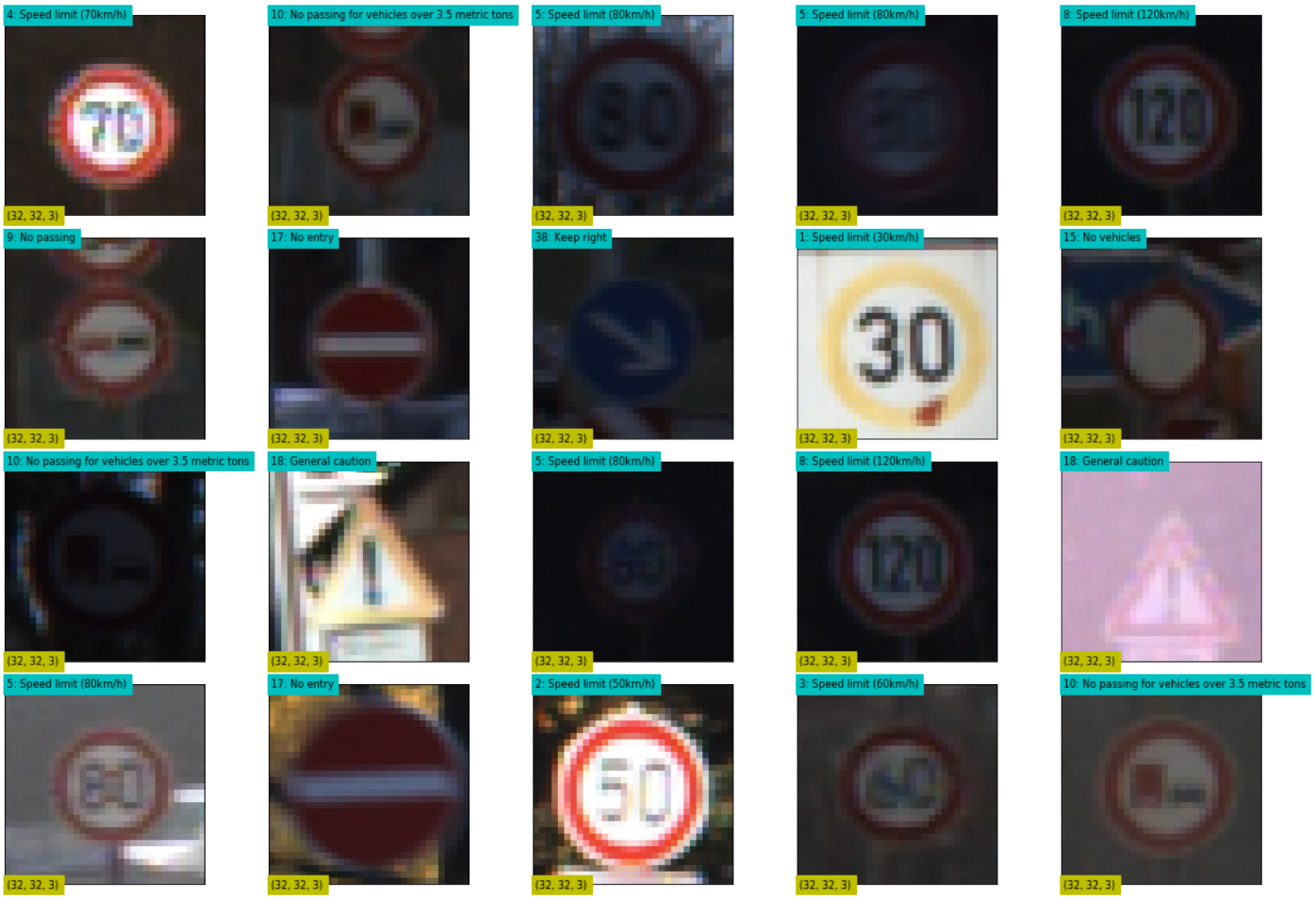

Figure 3: Example Training Images (Section: 1.5)

As can be seen in Figure 3 there is a significant amount variation between images per class. Also, by restricting the sizes of the images to be 32x32 or constant, this causes a loss of information through downsampling or upsampling of the raw image data. This prompted the need for data augmentation to create more varied training data, so the network could learn to generalize better.

3 Data Augmentation

Significant data augmentation was implemented in order to combat overfitting due to the relatively small training dataset. The majority of the data augmentation ideas were adopted from Krizhevsky et al. [3]. More specifically, I adopted the data augmentation and spacial and color based augmentation schemes. This data augmentation increased the training set by a from 34799 to 173995 images which significantly helped the model training and test accuracy and generalization. The data augmentation consisted of the following steps:

- Translation -$\pm$10 pixels in the x and y directions
  - Rotation - $\pm 45°$
- Flipping - Horizontal flipping
- Rotation + Translation - Take rotated images and apply the same translation above.
- Color-based - Adopted from [3]. The method employs a variation of PCA where the common eigenbasis of the RGB pixel intensity values are found and then stochastic noise is added to each pixel in proportion to the representation of that color eigen basis.

$$\begin{bmatrix} R'_{i,j} \\ G'_{i,j} \\ B'_{i,j} \end{bmatrix} = \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix} \begin{bmatrix} \alpha_1\lambda_1 \\ \alpha_2\lambda_2 \\ \alpha_3\lambda_3 \end{bmatrix} + \begin{bmatrix} R_{i,j} \\ G_{i,j} \\ B_{i,j} \end{bmatrix}$$

  Where λ and v are the eigenvalue, eigenvector pair for each RGB channel and α is the stochastic noise. The eigenvalues and eigenvectors were calculated via each image's correlation matrix. The correlation matrix was found by flattening each R, G, and B channel into a single dimension vector and then appending them together to form a $N^2 \times 3$ matrix which outputted the correct dimensionality of a 3×3 correlation matrix. The reasoning for implementing this alteration method is that an image will maintain the same classification under different lighting conditions and small variations in the exact pixel intensities.

3.1 Pre-processing

This network performed surprisingly well without pre-processing or normalization. The average training time for 150 epochs was 45 minutes with an accuracy of 90+%. In the interest of time, I did not explore the further benefits of using normalization, although I knew that it may influence the gradient descent significantly.

4 Design and Learning of the Model

4.1 Model Architecture

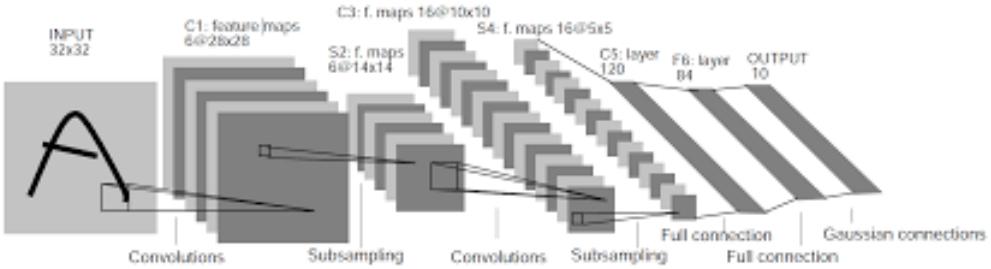

Figure 4: Model Architecture

The model is almost entirely taken from LeCun et al. [4] and can be seen in Figure 4. This architecture was chosen because of the relative small complexity of the dataset which required few learned features in order to correctly label each image, but I did make some adjustments. The first layer was the start of the convolutional layer with filters of size 5x5x6 and stride of 1. I then added in a localized response normalization term from [3] in order to penalize

high weights in order to reduce overfitting. Then the features were passed through a standard ReLU (Rectified Linear Unit) activation function.

Next a max pooling was applied with kernel's of size 2x2 and stride 1. Then the above layers were applied again in the same order including all dimensions and strides. The output of the second max pooling layer was then flattened and passed through two fully connected layers each with a ReLU activation unit and dropout in order to combat overfitting. The output of the second fully connected layer was fed into a a last fully connected layer to constitute the final output of the network. The optimization method used was ADAM [5] and it proved to be an effective learning algorithm for optimizing the mean cross entropy of the validation set.

4.2 Hyperparameter Selection

There were three main hyperparameters used in my framework: batch size, number of epochs, and the learning rate. The values for the hyperparameters were

- Batch size: 300
- Number of epochs: 150
- Learning rate: 0.0001

As can be seen in Figure 5, the validation accuracy was still rising slightly after 150 epochs, along with the training accuracy. The training accuracy was lower than the validation accuracy which meant that there was not as much memorization occurring the training of the model. Possibly, if I increased the number of epochs to 200, I may have hit 94% on the validation set. The batch size was small compared to the number of images in the model. Due to memory constraints, I did not test batch sizes beyond 500 and found 300 to be a suitable value that yielded the best validation accuracy.

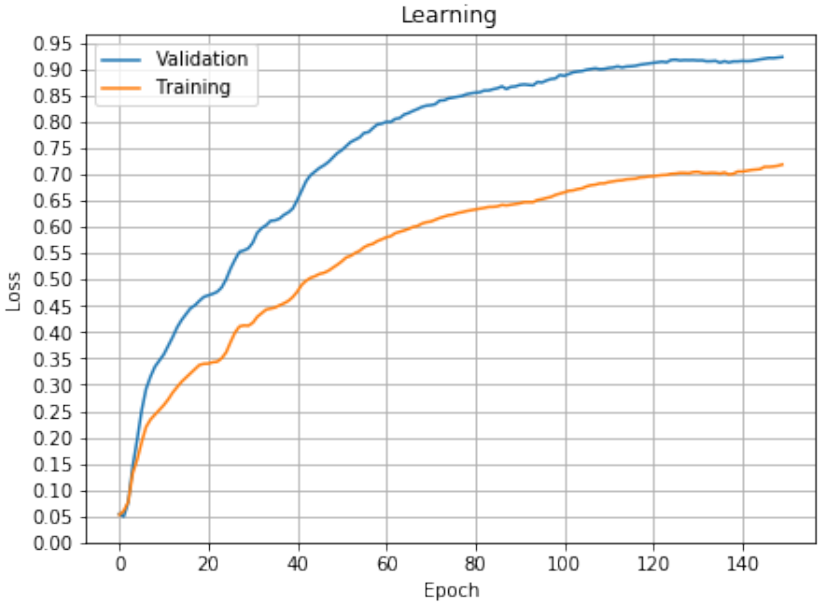

Figure 5: Validation Error vs. Training Error

5 Results

5.1 Testing Set Results

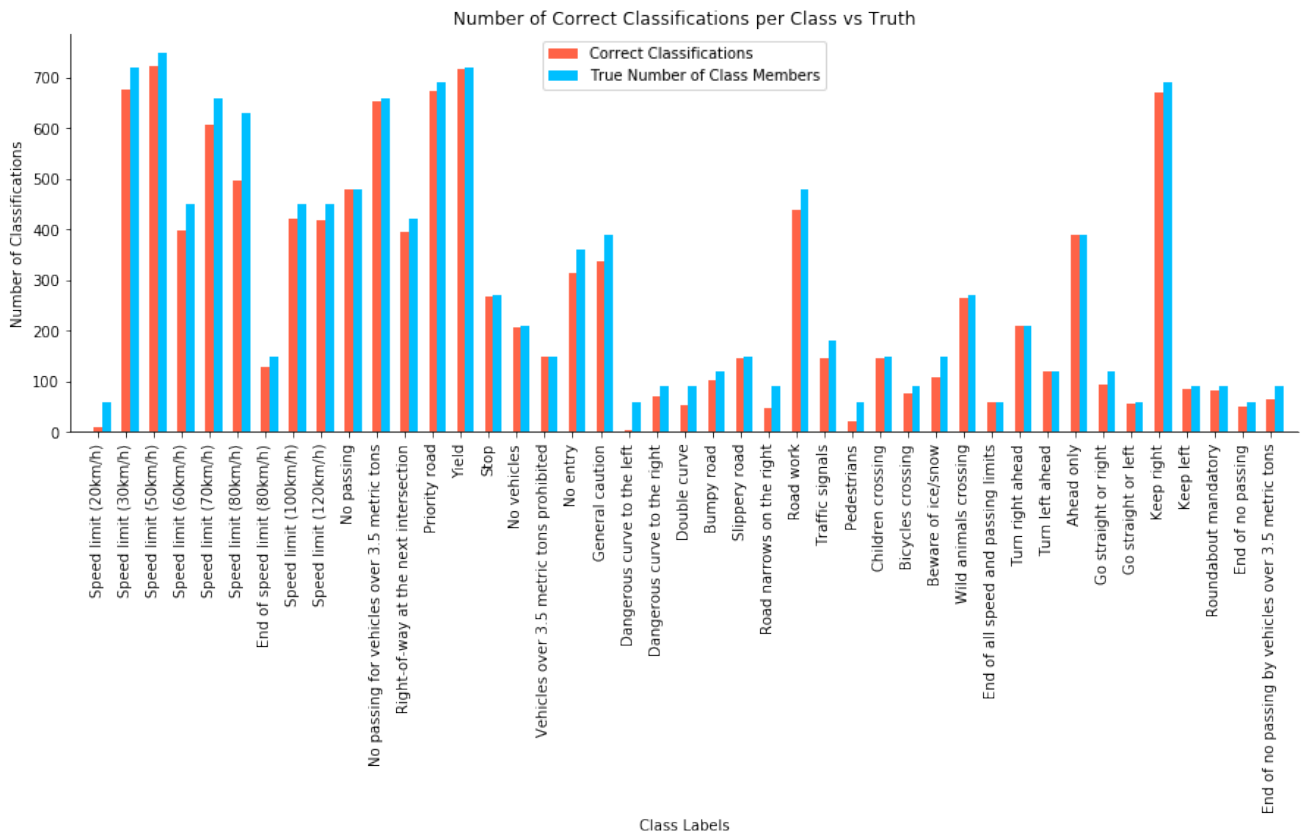

Figure 6: Number of correct and incorrect classifications per class of the testing set.

Note, that the overall classification accuracy was 91.7%. It is very evident from Figure 6, that there was no class that had a large majority of the misclassifications. However, the number of misclassified images is dependent on the number of images per category. A category with more images has a higher chance of error. Overall, from the results we see that the classifier is able to identify the different categories of the traffic signs fairly well.

5.2 Classification of New Traffic Signs

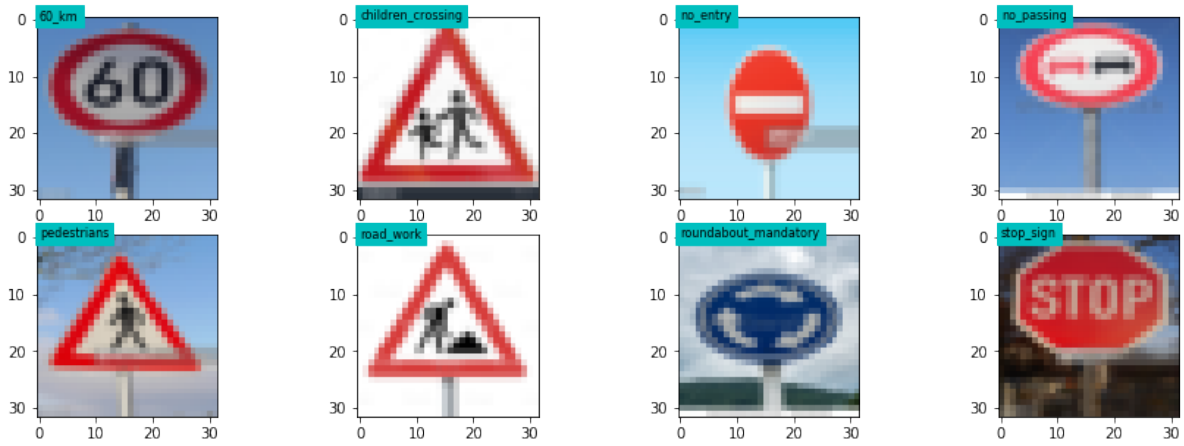

Figure 7: Eight new traffic images.

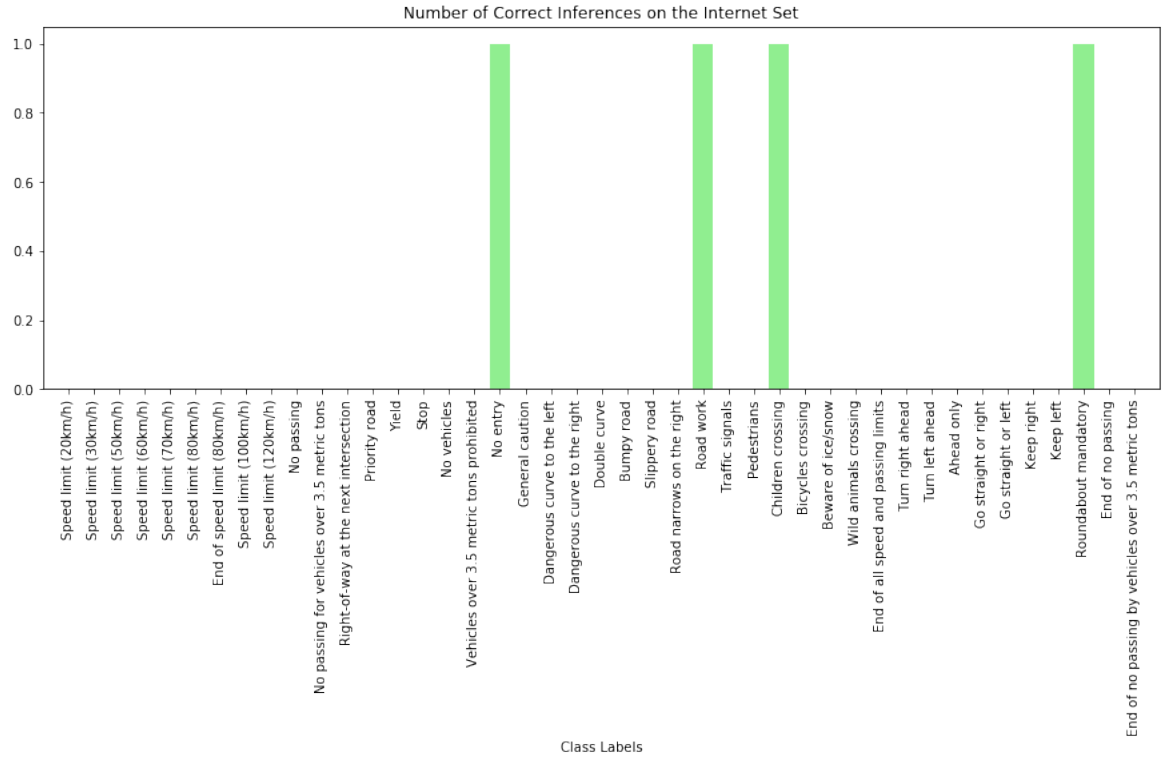

Figure 8: Correct classification of new traffic images.

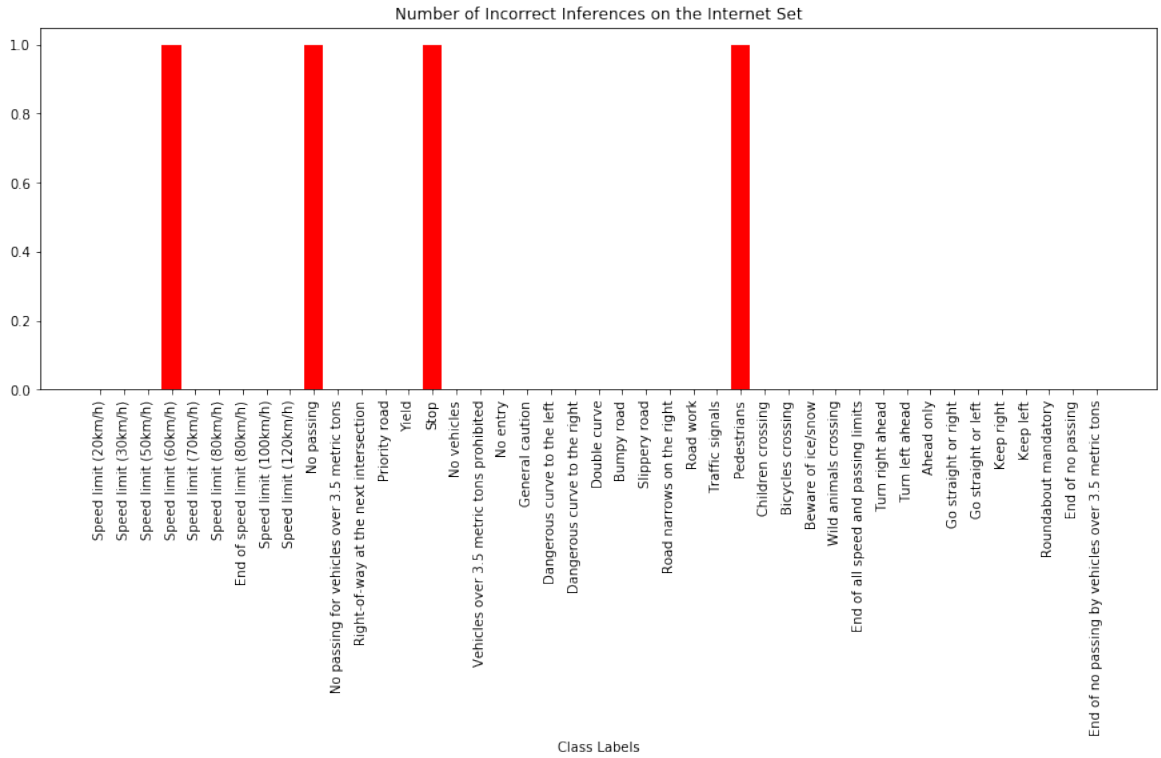

Figure 9: Incorrect classification of new traffic images.

The overall classification accuracy of the eight images was 50%. The four images classified were the "Speed Limit (60km/h)", "No passing", "Stop", "Pedestrians." One reason for the misclassification rate was due to the watermark on the images which were not contained in the original training images.

5.3 Accuracy Comparison Between New and Old Test Sets

The accuracy for the new test set of images was 50% while the original test set was 91.7%. This large gap in accuracy is due to the overfitting on the training data. Both the training and test data are very similar and were likely collected from the same sources, so this would explain the high accuracy on the original test set. In addition to the watermarking on the new images, the variation in them may have not been seen by the network, so its ability to recognized the object decreased. This suggests that more layers and more training data need to be employed.

5.4 Softmax Probabilities

The top five softmax probabilities for the eight images (Section: 3.5). Five of the images had their top probability over 74% and the other three images were around 52%.

5.5 Inference Time

The time of inference for a single forward pass averaged over all the new traffic images was 177 microseconds, which is suitable for a near real-time applications such as self-driving cars.

6 Visualizing the Models Learned Weights

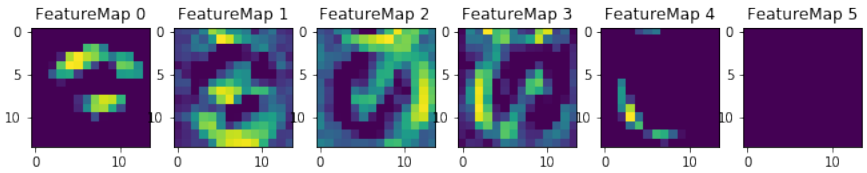

Figure 10: Layer 1 Visualization

Visualization from the first layer shown in Figure 10 reveals that the network was able to learn some circular features. Due to the size of the image the features are not very clear and are not easily interpreted. Note that we only have two layers in our network. If we wanted to see better defined features we would need more layers as this would extend the hierarchy of features that could be extracted.

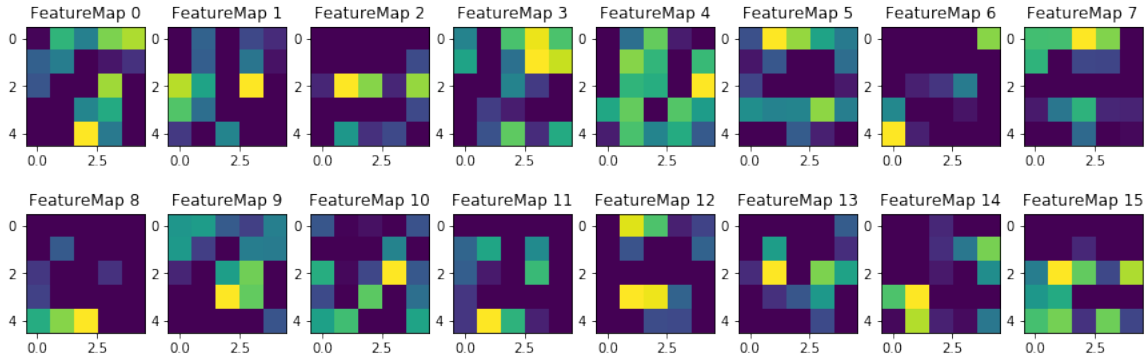

Figure 11: Layer 2 Visualization

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous systems," *tensorflow. org/. Software available from tensorflow. org*, 2015. https://arxiv.org/pdf/1603.04467.pdf.

[2] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural networks*, vol. 32, pp. 323–332, 2012. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.713.656&rep=rep1&type=pdf.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012. https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[4] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf.

[5] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014. https://arxiv.org/pdf/1412.6980.pdf.