Capstone Project

Machine Learning Engineer Nanodegree

Diaa Ahmed
March 3, 2019

Classify Traffic Signs Using Deep learning

Introduction

Autonomous vehicles have been gaining speed over the last couple of years. The Darpa Grand Challenge a prized competition for autonomous vehicles a decade ago sowed the seeds for developing autonomous vehicles. The aim of this competition was to bring out the research work that was happening in only the academia and big research labs into main stream implementation. Autonomous vehicles utilizes a combination of advanced hardware and software in order to perform the driving task.

It involves the usage of mechatronics a combination of mechanical and electronic components. Some of the mechatronics components are integrated cameras, laser sensors, radars, lidars, and feedback loop control systems for braking, acceleration and steering mechanisms. In order to integrate all these hardware, software techniques involving computer vision, Natural Language Processing, robotics and advanced Artificial Intelligence is required. Artificial intelligence involves Machine Learning, Deep Learning and Reinforcement learning where in the vehicle learns based on the existing data and also from the new data that the vehicle gathers in operation.

Problem Statement

A requirement for an autonomous vehicle is that the vehicle responds to traffic signs posted on the road so that it can adjust to speed, brake if needed, steer automatically or perform other necessary tasks that a human driver would perform. This involved the techniques of computer vision where the cameras of the vehicle detects the features of the images in the traffic sign and Machine Learning, Deep Learning in order to let the vehicle know what the traffic sign means and respond accordingly. Based on this requirement we can train a computer to detect the traffic signs and respond accordingly. This would help in reduction of accidents and improve the safety of driving. In our project I propose to use the Convolution Neural Network (CNN) for multi class classification of the traffic signs classification. The main purpose of the CNN is to classify different traffic images into their respective label classes. For example if the image is a stop sign the CNN model should correctly classify it as a stop sign label. In order to this I intend to follow the below procedure.

- Explore the dataset so as to find if the dataset is balanced or unbalanced. Perform image processing techniques such as resizing to a uniform size and removing noise if any.
- Create a validation set from the data set.
- Create a CNN using a buildup similar to the Lenet.
- Evaluate the model on the test dataset to see performance on unseen images.
- Testing the model on images that are not part the test data set to see how well our model generalizes on completely new data.
- Based on the performance of the model we could deploy the model onto the autonomous vehicles.

Datasets and Inputs

I plan to use the **German Traffic Sign Recognition Benchmark (GTSRB)** data set for my final capstone project. This data set is a benchmark for computer vision and machine learning. This data set is a large multi- category classification data set. The data set contains approximately 39209 training data, 12569 test data and 43 classes as labels. The link to the dataset is given below.

http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset#Downloads

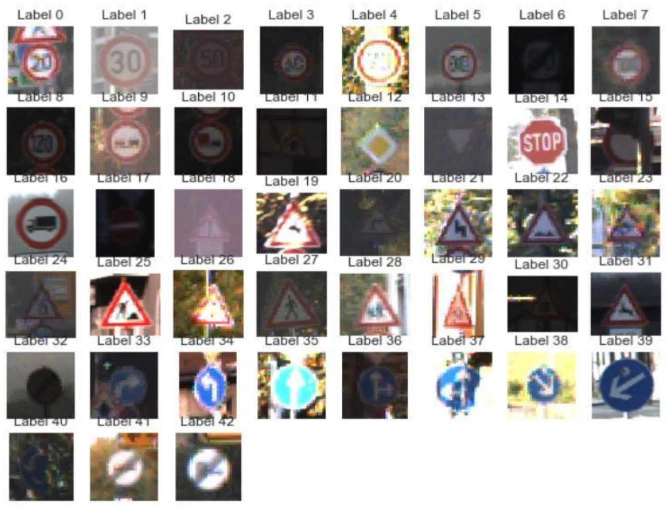
Training Data Set is the official data set that contains 39209 training images and 43 classes.

Test Data set (online-competition only) is the test used in the online competition stage of GTSRB. The test data set contains 12,569 test images and 43 classes.

Data Exploration

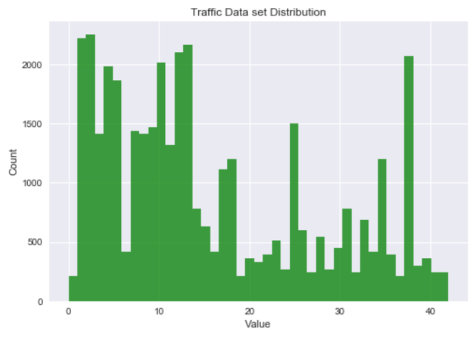
- Images and Respective Labels

The basic data is explored as to view the different images and the respective labels associated with the images.



- Histogram of the Labels

We plot the histogram of the labels to see the basic distribution of the dataset. We see that the dataset is unbalanced.



- Data Set Details

Dataset	# Of Images	Image Size
Train Set	39209	32 x 32
Test Set	12569	32 x 32

Metrics

In order to check the results of our model I use the classification accuracy. Basically we calculate the accuracy score as the percentage of correct predictions. Since our data set is unbalanced we also use the confusion matrix so as to find how accurate our model classifies the different classes of the traffic signs. Confusion Matrix indicates which off the classes is accurately classified and which not. This helps us in improving and tuning the model as to increase the accuracy of the model. Confusion matrix is a square matrix of size equal to the number of labels basically describes the performance of a classification model. For a multiclass classification the correctly predicted labels are the diagonals of the confusion matrix.

Along with the confusion matrix I am going to use the precision, recall and f1-score to evaluate the performance of the model. Methods for calculating accuracy, confusion matrix, Precision, Recall and F1 –scores for a binary classification are shown below. The same could be scaled to be used for a multi class problem.

Confusion Matrix

		Predicted Labels	
Actual Labels		1	0
	1	TP	FN
	0	FP	TN

TP- True Positive
FN- False negative
FP- False Positive
TN- True Negative

Accuracy: $\dfrac{TP + TN}{Total}$

Precision: $\dfrac{TP}{TP + FP}$

Recall: $\dfrac{TP}{TP + FN}$

F1-Score: $2\dfrac{\Pr ecision + \operatorname{Re} call}{\Pr ecision + \operatorname{Re} call}$

Benchmarking

In order to compare the results of our model we compare the results to the benchmark provided in GTSRB website. The link to the benchmark is shown below.

http://benchmark.ini.rub.de/?section=gtsrb&subsection=results

Algorithms and Techniques

In order to train a model on the traffic sign dataset, we used Convolution Neural Network (CNN) for the classification. The details of different layers used in defining the CNN model are described in this section.

Input Layer: Input to a CNN is a 2D image data with either RGB or YUV or gray scale (one channel). In our case our input image is a 2D image in gray scale format.

Convolution Layer: This layer computes the output of the neurons connected to the local regions of the input. It computes the dot product between the weights and a small region connected to in the input volume. This might result in a shape as [32x32x16] if we use 16 filters.

RELU Layer: The RELU layer applies an element wise activation function. A linear rectified function such as max(x,0) thresholding at zero is one of the activation functions. The relu function does not change the shape of the volume.

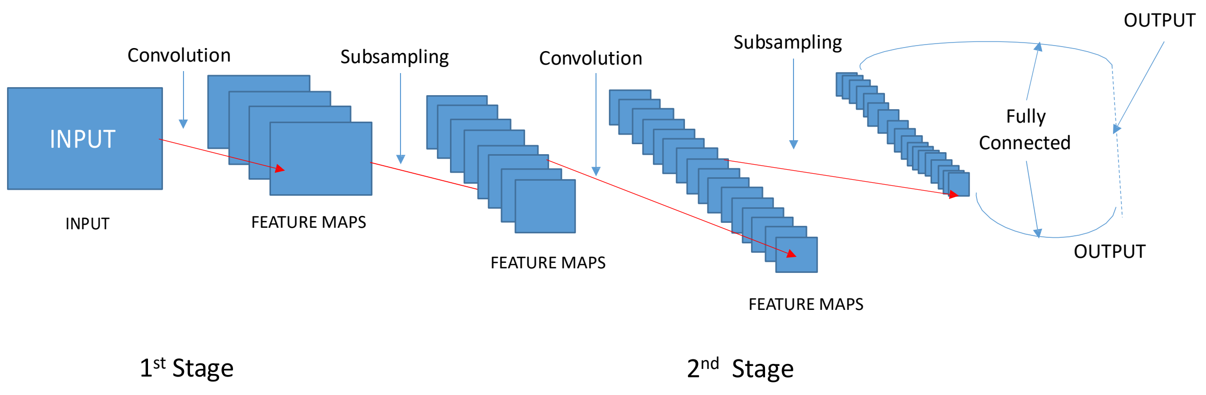
POOL Layer: The POOL layer performs the down sampling operation on the along the spacial dimensions (width, height). If we have a convolution layer size of [32 x 32 x16], after the POOL operation with a down sampling of 2 the resulting volume would be of shape [16x16x16].

Fully Connected Layer: The fully connected layer computes the class scores. This results in volume size of [1x1x43]. Each of the 43 numbers correspond to a class score of the 43 labels of the traffic signs data set.

Based on the above layer definitions the CNN transforms the original image layer from the original pixel values to the final class scores. The convolution layer and the fully connected layer performs transformations as functions of activation functions, weights and biases.

The RELU and POOL layers implement fixed functions such as the activation functions. The weights and the biases of the convolution layer and the fully connected layers are trained using optimization methods such as the gradient descent. The weights and biases trained using the optimization methods computes the class scores that are consistent with the class labels in the training set of each image.

Below we show a 2 stage CNN pipeline is shown below.



Data Preprocessing

- Load Data

  The train and the test data is loaded into python as a list of images. The length of the train images list is 39209 and test images is 12569.

- Resize images

  The images in the train and test data are not of uniform size that is have different pixel sizes. Image size is generally defined by the #of pixels in rows and columns of the image. In order to have all images to be uniform, the train and test images set are resized to have a constant # of pixels. All the images in the train and test set are resized to 32 x 32 pixel images.

- Gray Scale images

  The images in the train and test data are in the RGB format. The images are converted to gray scale format for training the data using the CNN algorithm.

- Normalize images

  After the images are converted to gray scale the images are normalized so the images have a mean of zero and variance of 1.

- Reformat to CNN format

  The CNN network in tensor flow needs the data in cube form. The images after normalization are in the square format of size 32 x 32 representing the pixels in the image. The images are converted to a cube format by including the number of channels as the third dimension. As the images are converted into gray scale the images now have only one channel. Hence the images are reformatted to be of shape 32 x32 x1.

  The labels vector was converted to a one-hot encoding format for classification purposes. For example we converted the label 5 to a vector [0 0 0 0 0 1 ....0]. The vector has a length of 43 as there are 43 labels in the data set. Hence if the label vector y

  $y \in R^{n}$ After encoding we get a matrix of the form

  $y \in R^{nx43}$ 43 is the number of unique class labels in the data set.

Convolution Neural Network (CNN) Model Architecture

In order to train the traffic signs data set, I experimented with 2 different architectures. The two architectures are shown below.

Model 1 Architecture:

Two convolutional layers, followed by a fully connected network.

Convolution Layer 1:

- Input Size : [batch size X 32x32x1]
  - Batch size is the size of the feature matrix for training purposes
- Output Size : [5 X 5x1x16]
  - 5x5 is the patch size
  - 1 - # of Input channels
  - 16- # of Output channels

Sub Sampling Layer 1:

- Input Size : [5 X 5x1x16]
- Output Size : [batch size X 16x16x16]

Convolution Layer 2:

- Input Size : [batch size X 16x16x16]
- Output Size : [batch size X 8x8x16]

Features are down sampled.

Fully Connected Layer Weight shape

- Input Size : [batch size X 8 X 8 X 16]
- Output Size : [1024x 64]
    - 64 - # of Hidden units in the fully connected layer

Output Layer, Weight Shape: [64 x43]

- 43- # of class labels

For optimization of the cost function no regularization was used.

Model 3 Architecture:

In this model we build a more detailed CNN. This architecture is very similar to the LENET architecture. In this model I have 2 convolution layers with max pooling and two fully connected layers. In this model I also have implemented the drop out and added L2 regularization so improve accuracy of the model and also prevent over fitting.

Convolution Layer 1:

- Input Size : [batch size X 32x32x1]
    - Batch size is the size of the feature matrix for training purposes
- Output Size : [5 X 5x1x16]
    - 5x5 is the patch size
    - 1 - # of Input channels
    - 16- # of Output channels

Sub Sampling Layer 1:

- Input Size : [5 X 5x1x16]
- Output Size : [batch size X 16x16x16]

Convolution Layer 2:

- Input Size : [batch size X 16x16x16]
- Output Size : [batch size X 10x10x32]

Features are down sampled and # of output channels is doubled.

Sub Sampling Layer 2:

- Input Size : [batch size X 10x10x32]
- Output Size : [batch size X 5x5x32]

Drop off Fully Connected Layer Weight shape

- Input Size : [batch size X 8 X 8 X 16]
- Output Size : [batch size x256]
    - 256- # of Hidden units in the fully connected layer

Drop off Fully Connected Layer Weight shape

- Output Size : [batch size x256]

    - 256- # of Hidden units in the fully connected layer

Output Layer, Weight Shape: [batch sizex43]

- 43- # of class labels

For optimization of the cost function L2 regularization was used.

Results

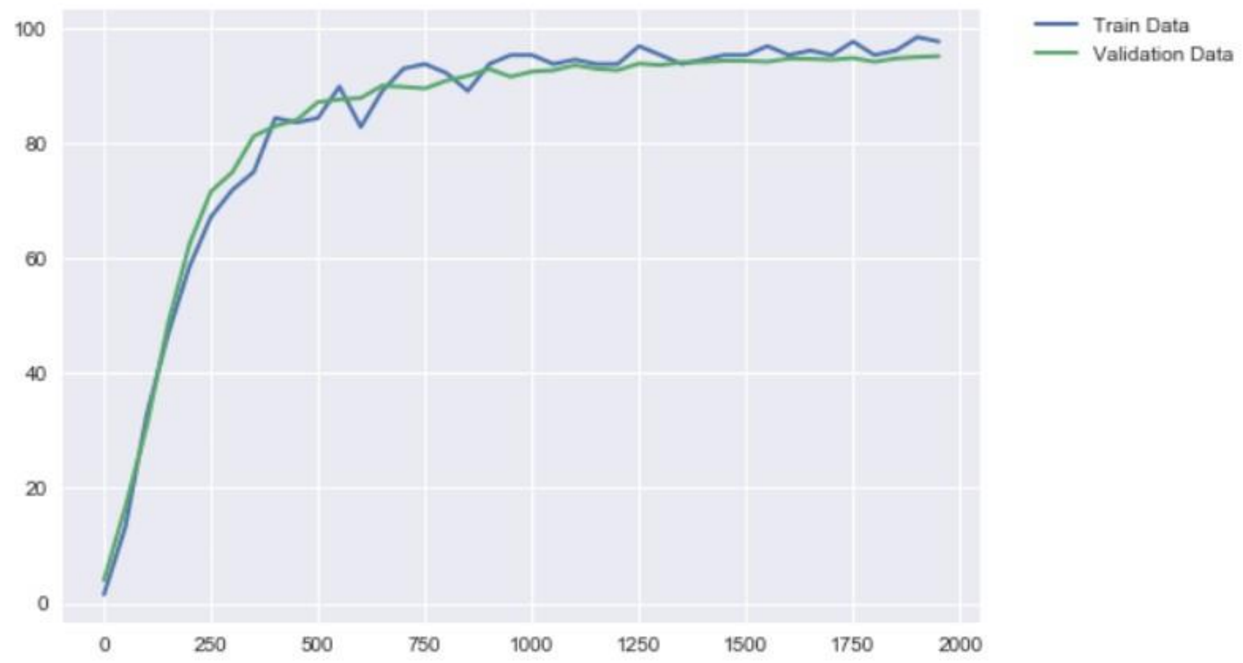
After training the dataset with 2 different models we have the following results.

Model 1 Accuracy.

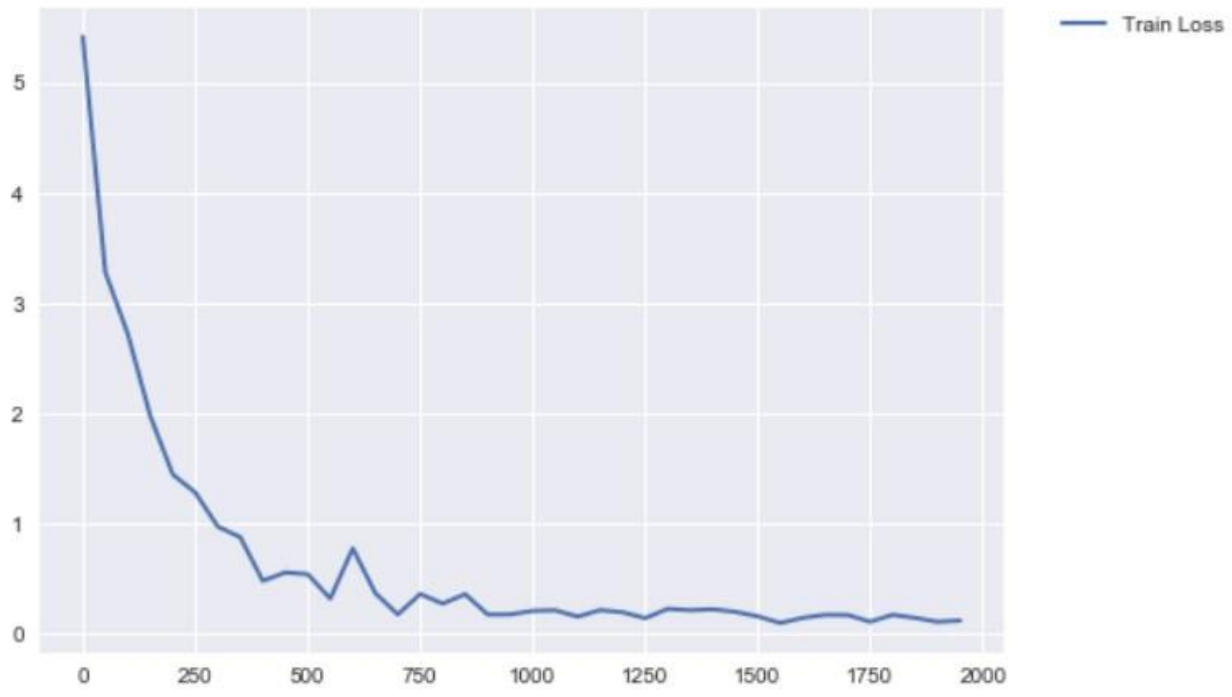
Model 1 was run for 2000 epochs, batch size = 128, patch size =5, depth =16, hidden units=64

Data Set	Accuracy
Train Set	97.5%
Valid Set	95.2%
Test Set	96.9%

Model 1 Train and Valid Data Set Accuracy Curves:

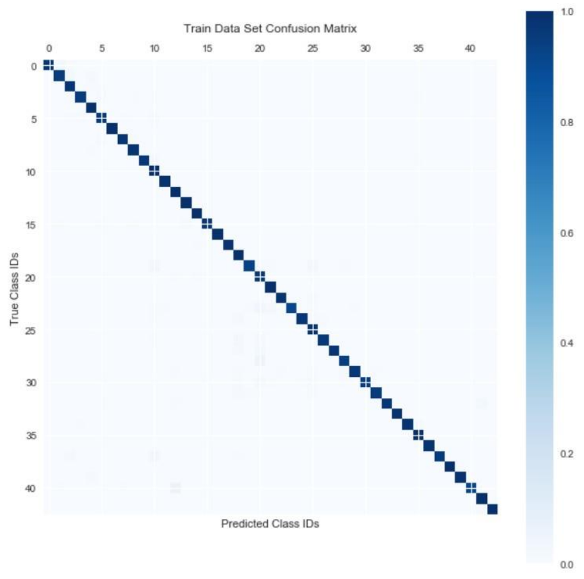


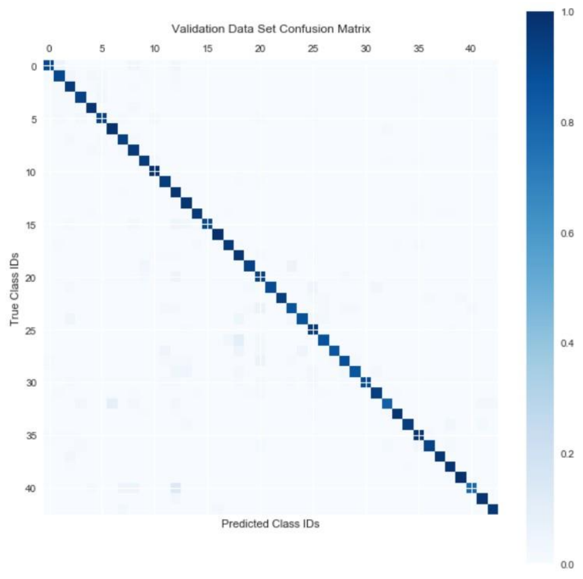
Model 1 Train Data Set Loss Curve:

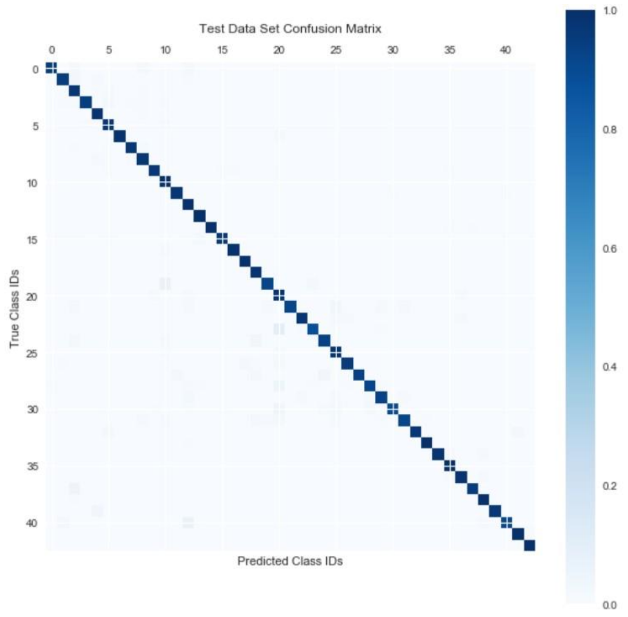


Model 1 Confusion Matrix:

The confusion matrix of the train, valid and test set is shown below







Looking at the confusion matrix our model does classify the labels well although the dataset is unbalanced.

We also list the classification of the results, total number of images classified correctly by the model (true positives) and percentage of true positives and the sum of true positives and false negatives.

Table 1: Precision, Recall and F1-Score Model 1

Labels	precision	recall	f1-score	support
0	1.00	0.82	0.90	60
1	0.97	0.96	0.96	720
2	0.96	0.96	0.96	750
3	0.98	0.93	0.95	450
4	0.94	0.99	0.97	660
5	0.93	0.93	0.93	600
6	0.99	0.98	0.99	120
7	0.96	0.97	0.96	480
8	0.97	0.93	0.95	450
9	0.97	0.98	0.98	480
10	0.98	0.98	0.98	660
11	1.00	0.96	0.98	420
12	0.96	0.99	0.98	690
13	0.99	0.98	0.99	720
14	1.00	1.00	1.00	240
15	0.95	0.99	0.97	210
16	1.00	0.99	1.00	120
17	0.99	1.00	0.99	360
18	0.99	0.98	0.99	390
19	0.91	0.97	0.94	60

20	0.87	0.97	0.91	120
21	0.81	0.98	0.88	90
22	0.98	0.99	0.99	120
23	0.99	0.93	0.96	150
24	0.93	0.94	0.94	90
25	0.98	0.99	0.98	480
26	0.96	0.96	0.96	180
27	0.95	0.90	0.92	60
28	0.99	0.91	0.95	180
29	0.90	0.94	0.92	90
30	0.89	0.99	0.94	150
31	0.98	0.93	0.95	240
32	1.00	0.98	0.99	60
33	1.00	1.00	1.00	209
34	1.00	0.99	1.00	120
35	0.99	0.98	0.98	390
36	0.98	1.00	0.99	120
37	0.92	1.00	0.96	60
38	1.00	0.99	0.99	690
39	0.95	0.97	0.96	90
40	0.97	0.91	0.94	120
41	0.95	1.00	0.98	60
42	0.97	0.95	0.96	60
avg / total	0.97	0.97	0.97	12569

Macro Averaging of Precision, Recall and F1 Scores

Precision	Recall	f1 score
0.96	0.96	0.96

Micro Averaging of Precision, Recall and F1 Scores

Precision	Recall	f1 score
0.97	0.97	0.97

Weighted Averaging of Precision, Recall and F1 Scores

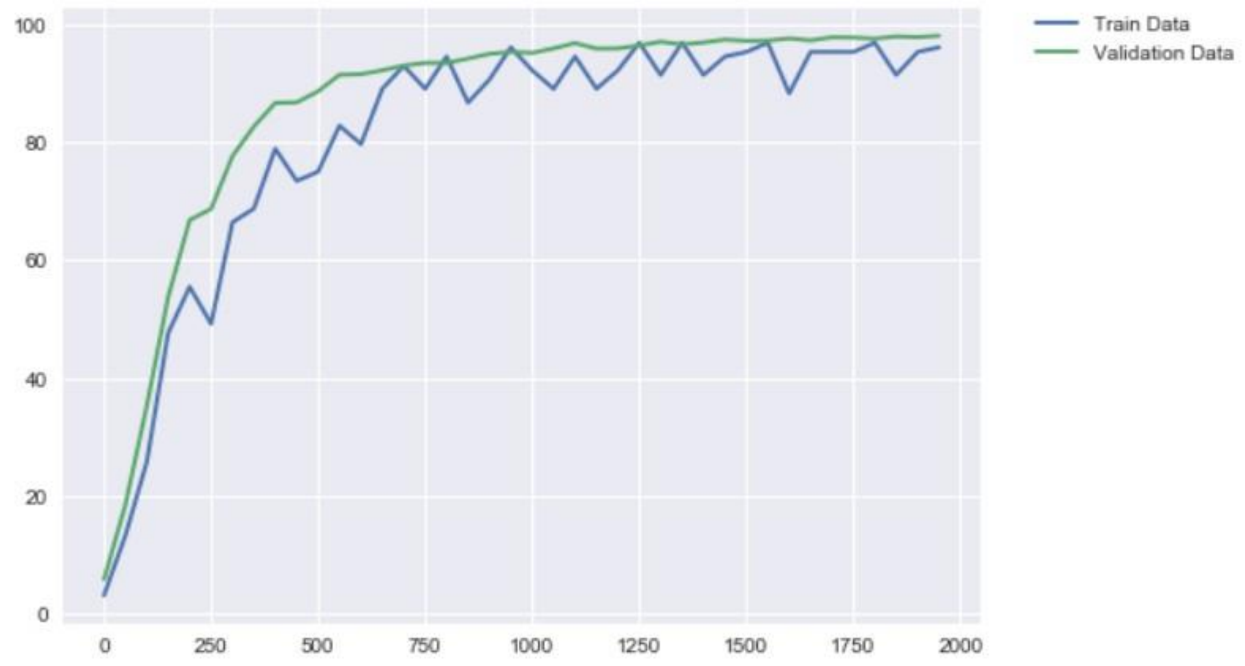
Precision	Recall	f1 score
0.97	0.97	0.97

Model 2 Accuracy.

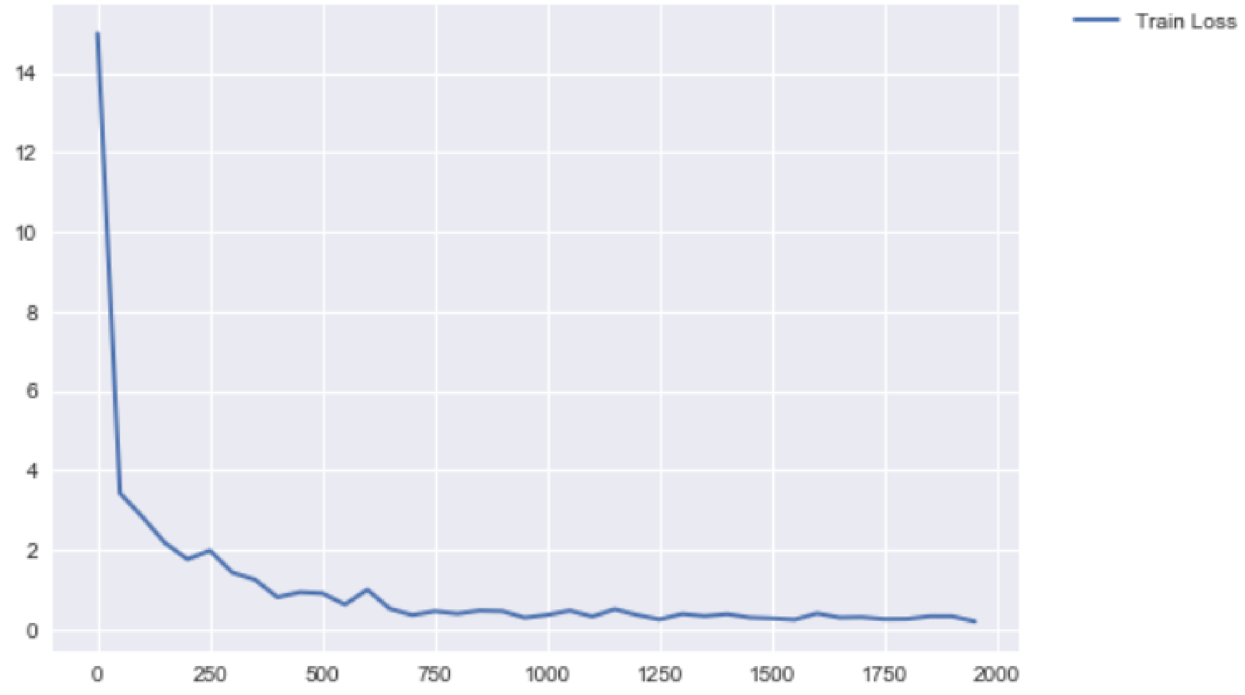
Model 2 was run for 2000 epochs, batch size = 128, patch size =5, depth =16/32, hidden units=256, drop out, L2 regularization.

Data Set	Accuracy
Train Set	96.4%
Valid Set	98.0%
Test Set	98.8%

Model 2 Train and Valid Data Set Accuracy Curves:

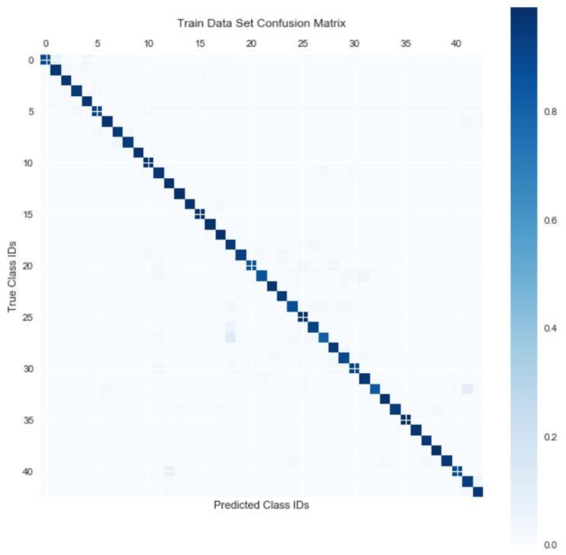


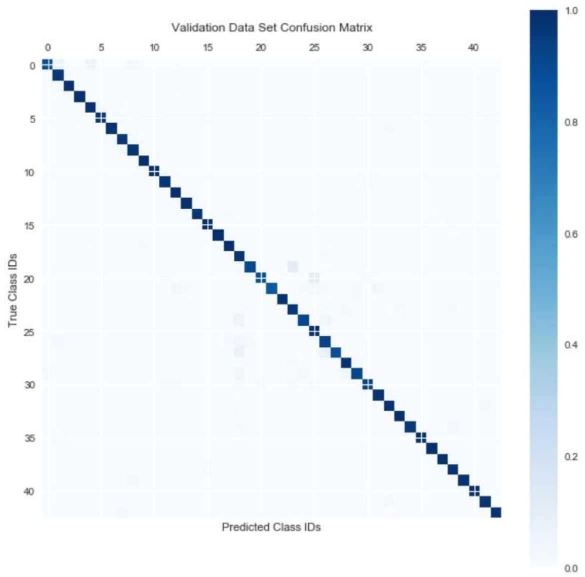
Model 2 Train Data Set Loss Curve:

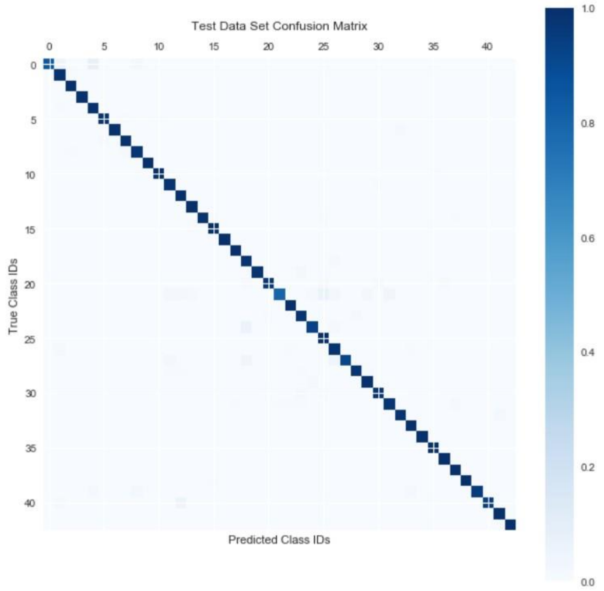


Model 1 Confusion Matrix:

The confusion matrix of the train, valid and test set is shown below







Looking at the confusion matrix our model does classify the labels well although the dataset is unbalanced.

We also list the classification of the results, total number of images classified correctly by the model (true positives) and percentage of true positives and the sum of true positives and false negatives.

Table 2- Precision, Recall and F-score for Model 2

Labels	precision	recall	f1-score	support

0	1.00	0.95	0.97	60
1	0.99	0.99	0.99	720
2	0.99	0.99	0.99	750
3	0.99	0.98	0.98	450
4	0.99	0.99	0.99	660
5	0.98	0.98	0.98	600
6	1.00	0.98	0.99	120
7	0.98	0.99	0.98	480
8	0.99	0.98	0.98	450
9	0.99	0.99	0.99	480
10	1.00	0.99	0.99	660
11	0.98	0.98	0.98	420
12	0.98	1.00	0.99	690
13	1.00	1.00	1.00	720
14	1.00	0.99	0.99	240
15	0.99	1.00	0.99	210
16	1.00	1.00	1.00	120
17	0.99	1.00	1.00	360
18	0.96	0.99	0.98	390
19	0.95	1.00	0.98	60
20	0.94	0.98	0.96	120
21	1.00	0.84	0.92	90
22	0.98	1.00	0.99	120
23	0.98	0.99	0.99	150
24	0.93	0.89	0.91	90
25	0.99	1.00	0.99	480
26	0.97	0.96	0.97	180
27	0.96	0.88	0.92	60
28	0.99	0.98	0.98	180
29	0.97	0.98	0.97	90
30	0.99	0.97	0.98	150
31	0.98	0.99	0.99	240
32	0.97	1.00	0.98	60
33	0.99	1.00	1.00	209
34	1.00	1.00	1.00	120
35	1.00	1.00	1.00	390
36	1.00	1.00	1.00	120
37	1.00	1.00	1.00	60
38	1.00	1.00	1.00	690

39	1.00	0.97	0.98	90
40	0.99	0.92	0.95	120
41	1.00	1.00	1.00	60
42	0.98	1.00	0.99	60
avg / total	0.99	0.99	0.99	12569

As compared to model 1 we see lot more classes classified correctly and we get a 100% accuracy on some of the labels in the test and valid set.

Macro Averaging of Precision, Recall and F1 Scores

Precision	Recall	f1 score
0.99	0.98	0.98

Micro Averaging of Precision, Recall and F1 Scores

Precision	Recall	f1 score
0.99	0.99	0.99

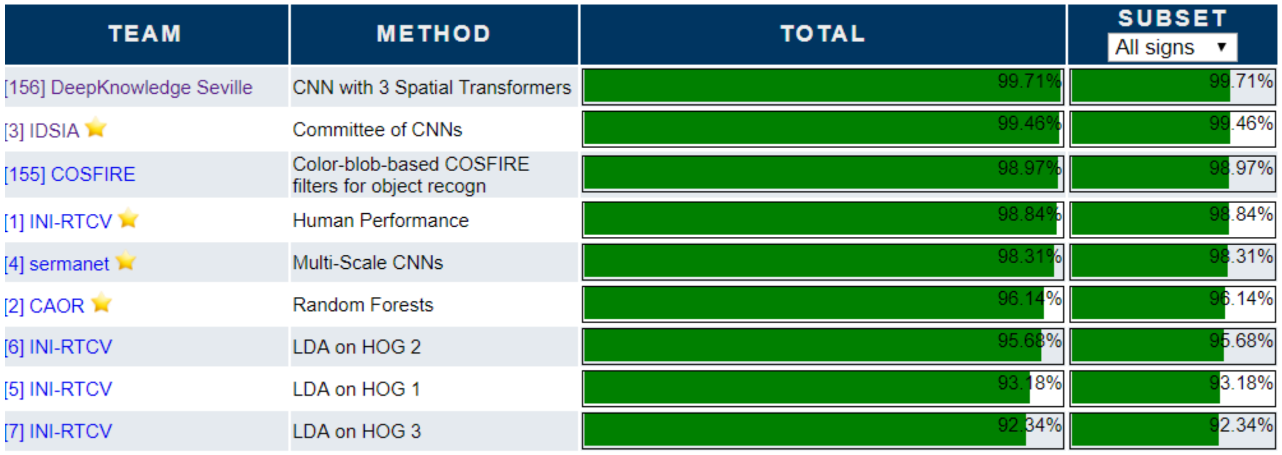
Weighted Averaging of Precision, Recall and F1 Scores

Precision	Recall	f1 score
0.99	0.99	0.99

Comparison Model 1 vs Model 2

Our detailed CNN model had an accuracy of 98.9% as compared to the 96.9% of the simple model 1.

Model 2 was able to classify better with the addition of drop out and usage of the L2 regularization. I used 2000 epochs for training the model, if I train the model for more number of epochs I could improve the accuracy. In the GTSRB website the benchmark results shows an accuracy of 99.71%. The snapshot of the results are shown below.



TEAM	METHOD	TOTAL	SUBSET All signs ▼
[156] DeepKnowledge Seville	CNN with 3 Spatial Transformers	99.71%	99.71%
[3] IDSIA ⭐	Committee of CNNs	99.46%	99.46%
[155] COSFIRE	Color-blob-based COSFIRE filters for object recogn	98.97%	98.97%
[1] INI-RTCV ⭐	Human Performance	98.84%	98.84%
[4] sermanet ⭐	Multi-Scale CNNs	98.31%	98.31%
[2] CAOR ⭐	Random Forests	96.14%	96.14%
[6] INI-RTCV	LDA on HOG 2	95.68%	95.68%
[5] INI-RTCV	LDA on HOG 1	93.18%	93.18%
[7] INI-RTCV	LDA on HOG 3	92.34%	92.34%

Ref: http://benchmark.ini.rub.de/?section=gtsrb&subsection=results

Conclusion

Free-Form Visualization

The model 2 is tested on new images in order to test random images to see how best our model predicts on new images. The images that are tested were are shown below.

Label 2

Label 22

Label 18

Label 33

Label 11

Label 23

Label 14

These images were passed through our model 2 and below is what our model predicted.

The actual labels are shown below

Y-Actual = [2, 22, 18, 33, 11, 23, 14]

The predicted labels are

Y-Predicted = [2, 22, 18, 33, 11, 23, 13]

We see that our model predicts all the images except the Stop Sign image or the image with label 14.

Our model has an accuracy of 0.86 on the new images that were not part of the original test data. If we generate more data by data augmentation image translation, image rotation, image warp and image shear would generalize our model better and end up increasing the accuracy.

Reflection

Deep Learning techniques have been able to solve lot of machine learning problems and the accuracy of these problems have also improved with the usage of deep learning.

Although Deep Learning techniques have solved a lot of tuff previously unsolved machine learning problems they are very computationally expensive. In order to support such expensive computations GPUs come to our rescue and since GPUs have become cheaper deploying our models will be very helpful.

The data set contained non uniform sized images and I had to preprocess images to a uniform size of 32x 32 and converted the images to gray scale. This helped in making my model accurate and also computationally less expensive as I reduced the 3 channel color images to 1 channel gray image.

Improvements

Using Deep learning techniques such as the Convolution Neural Network we developed a model for identifying and classification of traffic signs. More accurate and better classification would be very helpful for autonomous driving cars and other transportation systems.

Some of the techniques for improving the model would be to collect more data and generate synthetic data. We could also try normalized color channels as compared to the raw colors, but this would also increase the computational infrastructure. But with GPUs this techniques could be used so as to improve the classification of our model.

References:

http://benchmark.ini.rub.de/

http://ufldl.stanford.edu/

http://cs231n.github.io/

http://scikit-learn.org/stable/

https://www.tensorflow.org/

http://blog.revolutionanalytics.com/2016/03/com_class_eval_metrics_r.html

https://en.wikipedia.org/wiki/Confusion_matrix

https://en.wikipedia.org/wiki/Convolutional_neural_network