Machine Learning Engineer Nanodegree

Computer Vision Capstone Project - Traffic signal classification

Thirupathi Pattipaka

May 07, 2017

I. Definition

Project Overview

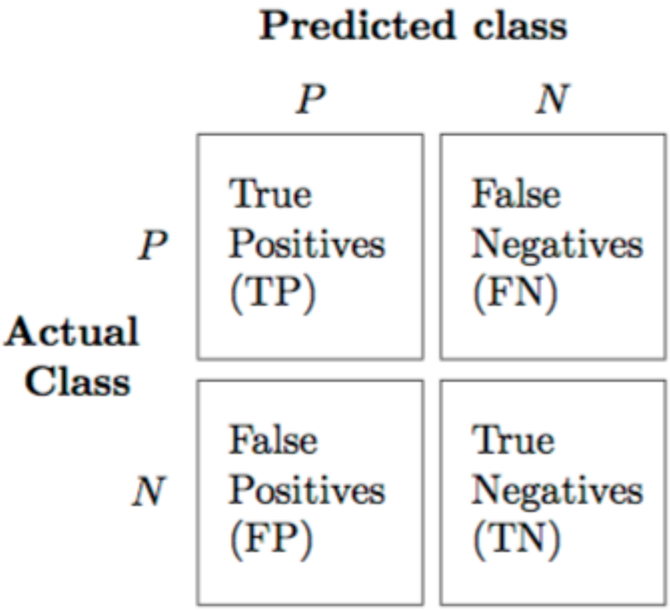
There are various companies are designing autonomous vehicles or self-driving cars. For autonomous vehicles, it is essential paramount to automatically detecting/classifying the traffic signs. It is very important to classify the traffic signs to avoid any accidents. Traditionally, computer vision and machine learning algorithms are used to classify these signs. Since the availability of GPU, deep neural networks able to produce better accuracy within limited time.

This project aimed to classify German traffic sign data by using deep convolutional neural networks using Keras and TensorFlow.

In this project, I have developed a classification algorithm based on deep learning algorithms, especially Convolutional Neural Networks (CNN) to automatically recognize traffic signs. The German Traffic sign data is publically available dataset (http://benchmark.ini.rub.de) which contains training and test data. The CNN model was trained with training data and validated with test data. The developed model performance was compared it with pre-exited classification algorithms.

Metrics

The main metrics to evaluate this classification of images in this project is accuracy. Classification accuracy is a ratio of the number of correct predictions out of all predictions that were made. It is often presented as a percentage between 0% (the worst possible accuracy) and 100% (the best possible accuracy). Other performance metrics will also be evaluated like Precision, Recall, F1 score.



		Predicted class	
		P	*N*
Actual Class	*P*	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

$$PRE = \frac{TP}{TP + FP}$$

$$REC = TPR = \frac{TP}{P} = \frac{TP}{FN + TP}$$

$$F_1 = 2 \cdot \frac{PRE \cdot REC}{PRE + REC}$$

PRE=precision, REC=recall, F1=F1-Score

Accuracy is defined as total number of correct predictions (True positives and true negatives) divided by total number of predictions made (TP + FP+FN+TN). Precision is the number of True Positives divided by the number of True Positives and False Positives. Put another way, it is the number of positive predictions divided by the total number of positive class values predicted. It is also called the Positive Predictive Value (PPV). Recall is the number of True Positives divided by the number of True Positives and the number of False Negatives. Put another way it is the number of positive predictions divided by the number of positive class values in the test data. It is also called Sensitivity or the True Positive Rate. The F1 Score is the 2*((precision*recall)/ (precision + recall)). It is also called the F Score or the F Measure. Put another way, the F1 score conveys the balance between the precision and the recall.

It is hard to judge that different traffic signs may have importance relative to other traffic signs. So it is a non-trivial judgment of which metric is the best, so I am using all metrics in this project.

II. Analysis

Data Exploration

I have imported the German Traffic sign data (http://benchmark.ini.rub.de). Before performing any exploratory analysis, the original data exists as a zipped file which once extracted contains the raw images as training and testing data. I must familiarize with data and German sign data consists of many traffic signs as shown below.

Image name: 70_speed,
True: 4

Image name: 80_lifted,
True: 6

Image name: right_of_way_general,
True: 12

Image name: lifted_no_overtaking_general,
True: 41

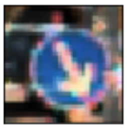


Image name: turn_right_down,
True: 38

Image name: no_overtaking_general,
True: 9

Image name: attention_bottleneck,
True: 24

Image name: attention_traffic_light,
True: 26

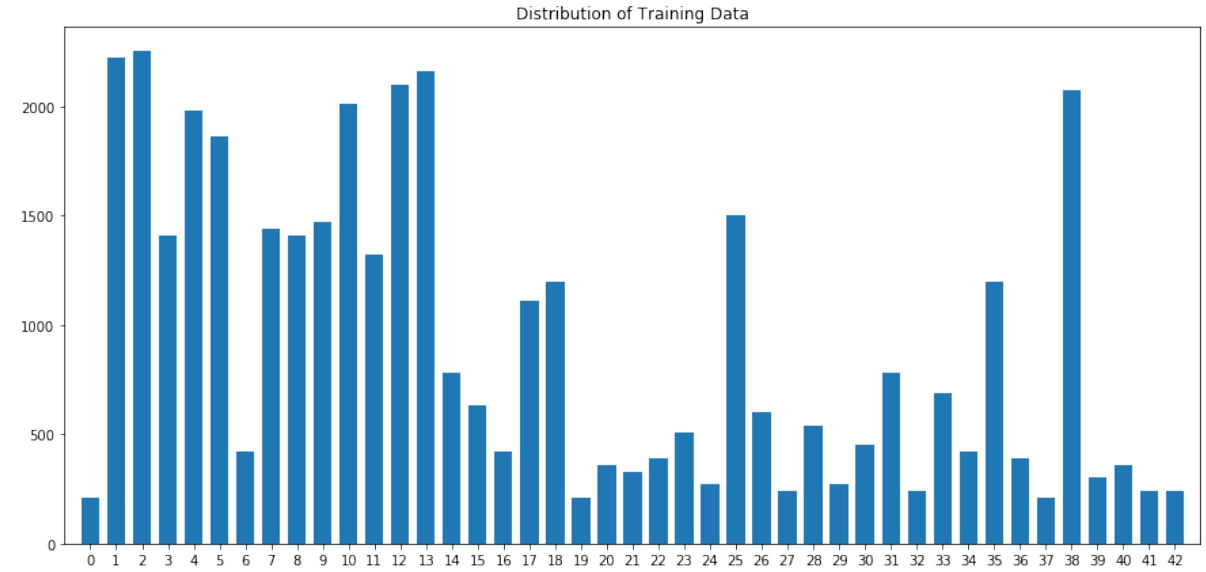
Image name: attention_deer,
True: 31

This data consists of more than 50,000 images in total and 43 classes in total. The downloaded the dataset has training dataset which contains 39,209 images in 43 classes. The training dataset contains 12,630 images with corresponding pre-calculated features in random order. The images contain various sizes of images.

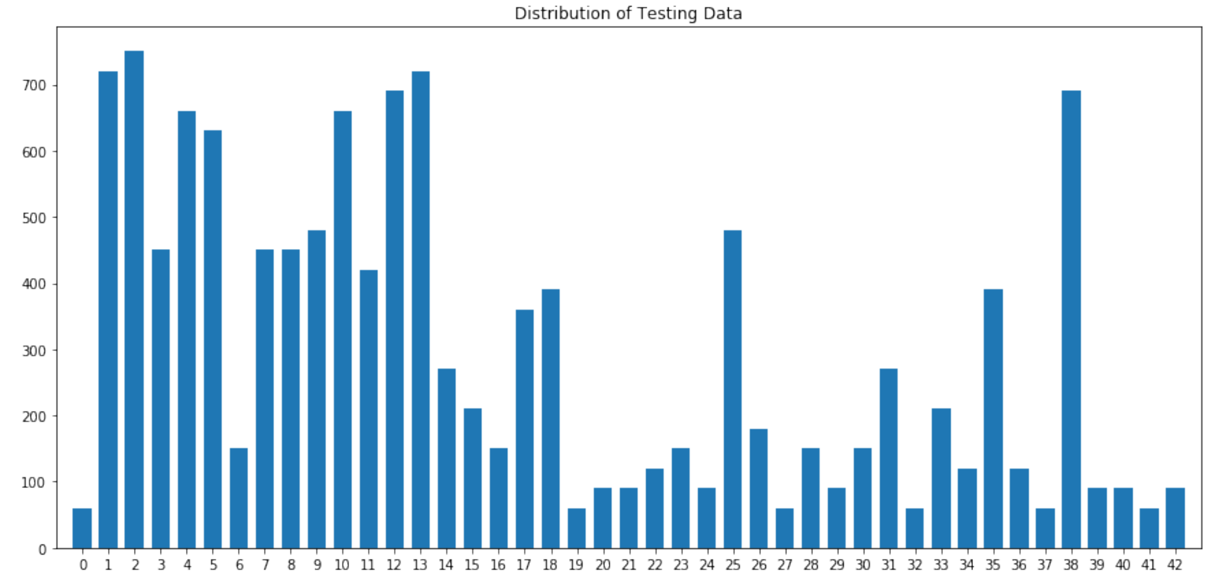
Exploratory Visualization

The below plots show that number of images per class in training and testing data. The distribution of training and testing data between different classes in training and testing data seem be similar and there is no imbalance found between them.

Number of images per class in training data



Number of images per class in test data



Algorithms and Techniques

The popular method for classifying images in recent years is that CNN. Therefore, I am using the CNN deep learning algorithm to classify images and predict the sign of image. This is largely due to the network being able to identify images at different scales. Another advantage here is that a CNN can directly work on raw pixel images.

My final deep convolutional neural network contains as below:

Images / Input Layer

This layer holds the input image which is equivalent to a store of the pixels of all images.

The first module in the model above is comprised of 3 1X1 filters. This filter will change the effect of color maps.

Convolution Layer

Convolutional layers are neural networks that share their parameters across space. It works by sliding over the vector with depth, height and width and producing another matrix (called a convolution) that has a different weight, depth height. Essentially, rather than having multiple matrix multipliers, we have a set of convolutions. For example, if we have an image of size 1024x1024px in 3 channels, we could progressively create convolutions till our final convolution has a size of 32x32px and a much larger depth.

Pooling

Pooling is a technique that can be used to reduce the spatial extent of a convolutional layer. Pooling finds a way to combine all information from a previous convolution into new convolution. For our example, we'll be using max-pooling, takes the maximum of all the responses in each neighborhood. We choose it because it does not add to our feature space and yet gives accurate responses.

Fully Connected Layer

This layer connects every neuron from the previous convolutions to every neuron that it has. It converts a spatial like network to a 1D network, so we can then use this network to produce our outputs

The Output layer

The output from this layer are logits which represents matrix showing the probabilities that of having a character in a position.

The drop outs are to avoid overfitting and force the network to learn multiple models for the same data.

This network contain 3 modules of 32, 64 and 128 convolutional layers (respectively) 3X3 filters followed by maxpooling and dropouts.

Benchmark

The primary benchmark here would be the accuracy of prediction with testing data. Accuracy score is the percentage of correctly predicted data. There are various algorithms which already implemented with this data has produced accuracy of ~98%.

III. Methodology

Data Preprocessing

We performed the following steps to preprocess data:
1. Downloaded the data from German Traffic image data website (http://benchmark.ini.rub.de/)
2. Generate new images using the bounding boxes in the previous image.
3. Resize our new images to 48x48 pixels.

Followed above steps for both training and testing datasets.

My implementation is covered in Jupyter notebook: German_Traffic_Sign_Data.ipynb. This notebook included with all implementations in this project.

Convolution model

To train, we read the already preprocessed data into the model and train it in batches. During the training, we try to minimize loss and log the accuracy we are achieving so we can keep track of how well our model is improving. Once the model is trained, we evaluate it using our test step. One step further would be to save the model and load the model so it can be used in other applications such as an android app.

There are various ways we can improve the model by Hyper-parameter optimization. Here is a brief list of hyper-parameters:

- Learning rates
- Batch size
- Training epochs
- Image processing parameters
- Number of layers
- Convolutional filters
- Convolutional kernel size
- Dropout fractions

IV. Results

Model Evaluation and Validation

The initial model accuracy, I could achieve as following:

- Minibatch accuracy: 97.48%
- Validation accuracy: 98.14%
- Test loss: 2.673838
- Test accuracy: 93.59%

According to our benchmark based on previous published papers, this model produced less accurate predictions when compared with the base classifier, although not as good as humans do. This is because my CPU computer which I am using quite slow and it is taking vast amounts of time.

The final model made of the following

- Max pooling to the hidden layers
- convolutions for depths at 32,64 and 128
- created 43 classifiers
- learning rate decay at 1e-06
- Stochastic gradient descent optimizer
- Accuracy score as our benchmark

I have tried to improve the model accuracy by increasing number of epochs and changing the learning rate decay.

When I change the number of epochs to 20, the model accuracy on training data is about 99.60% and accuracy with test data is about 95%. The model accuracy has not improved much with learning rate decay of 1e-07. I have used GridSearchCV method to select the best learning rate and momentum. I have tried these learn_rate [0.001, 0.01, 0.1, 0.2, 0.3] and momentum = [0.0, 0.2, 0.4, 0.6, 0.8, 0.9].

With final model with 95% accuracy is robust enough to check this algorithm with any external images by creating an app.

Justification

Final model here produced 95% accuracy while training on ~37000 images, which is not the optimal. There are several ways we can improve the model performance. One of the way is increasing the epoch number to 30. If we have over 200,000 more images to train on, we can expect the accuracy to continue to improve moving closer to human recognition.

V. Conclusion

Free-Form Visualization

To test how well our model is working, we created random images and used it for our prediction. Here is the actual images and their labels:

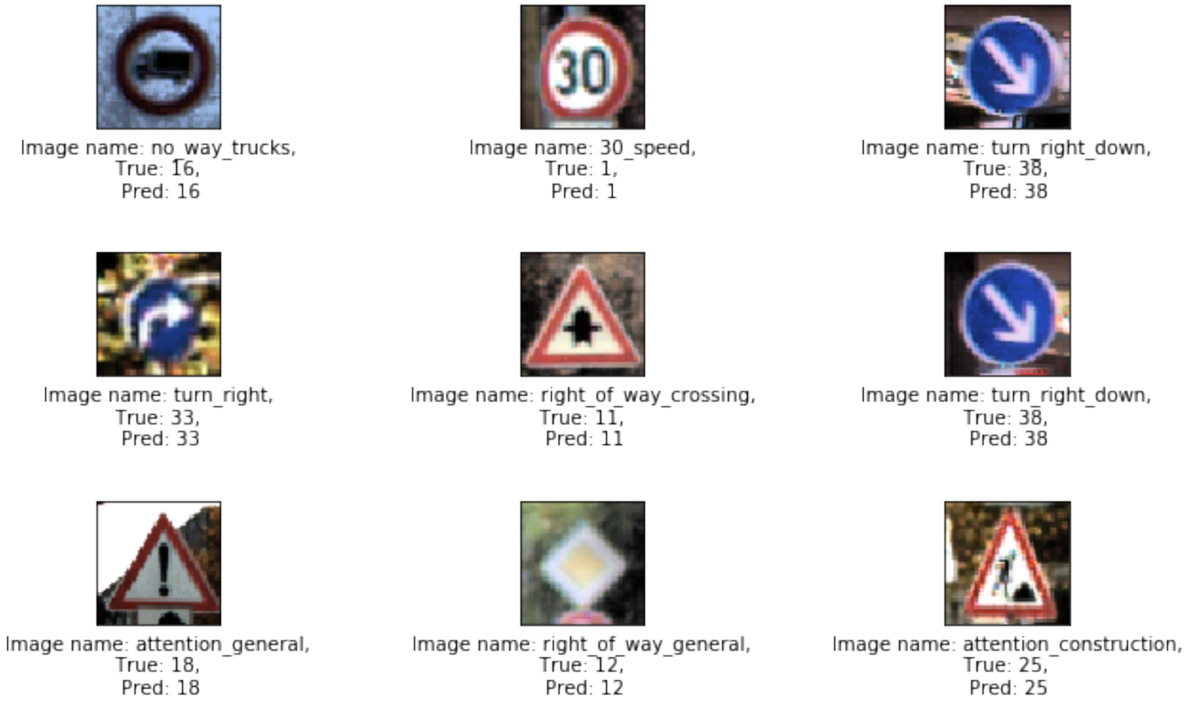


Image name: no_way_trucks,
True: 16,
Pred: 16

Image name: 30_speed,
True: 1,
Pred: 1

Image name: turn_right_down,
True: 38,
Pred: 38

Image name: turn_right,
True: 33,
Pred: 33

Image name: right_of_way_crossing,
True: 11,
Pred: 11

Image name: turn_right_down,
True: 38,
Pred: 38

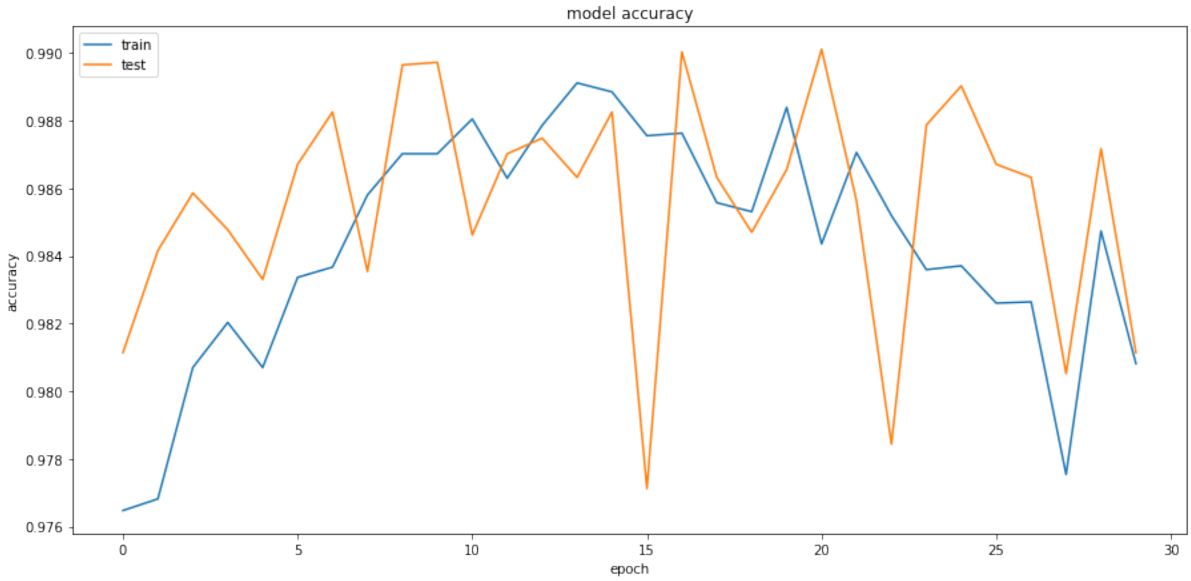
Image name: attention_general,
True: 18,
Pred: 18

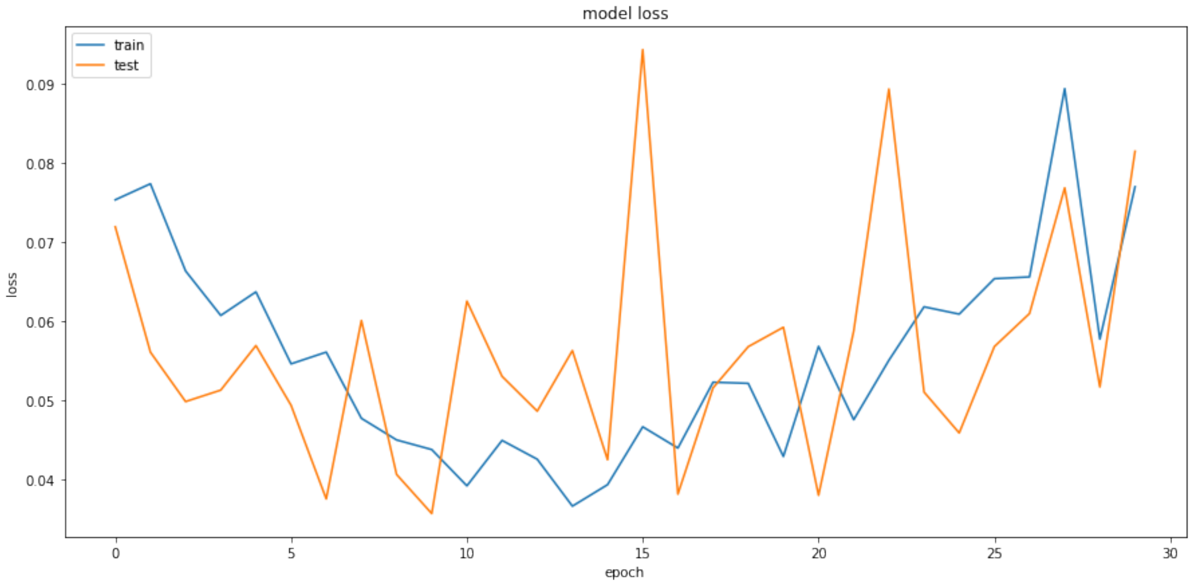
Image name: right_of_way_general,
True: 12,
Pred: 12

Image name: attention_construction,
True: 25,
Pred: 25

These images were then passed through our model and here is what our model predicted the values shown above.

We can see that our model did a pretty good job at predicting data and it did so with a training set of just over 30000. If we continue to train with the larger set of 200,000 images, we will achieve even better results, however, my computer might take a long time to do that, since I am training on a CPU.





Reflection

This was a very interesting project, and probably the first time I tackled deep learning problem from scratch to a stage where I could compare performance by changing various hyper parameters. There are several things that can be tried to further improve the model. One easy thing to do is to let the model train for longer time. As each training cycle was taking about 4 hours I did not perform any hyper parameter optimization to choose the best learning rate or regularization.

The difficult thing what I found in this project is that it is very difficult train CNN with CPU which has taken so much of time to run 10 epoch model and the very interest aspect of this project is that I am able to learn and tackle my first deep learning project and able to learn many aspects of convolutional neural networks and I am interested to learn Recurrent neural networks to apply on text data. The solution which I developed here is that an CNN model which is able to predict the Traffic signs with 95% accurate and there is immense chance of developing this algorithm further.

Improvement

There are several ways this implementation can be improved. One of which is to run this model on a GPU. Scholars have written on how this can provide up to 10x improvement in the training time for deep neural networks.

Other way is that data augmentation. Another improvement here would be to explore Recurrent neural networks and changing the learning rate

References

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

- https://www.tensorflow.org/versions/r0.10/tutorials/deep_cnn/index.html
- http://benchmark.ini.rub.de/
- https://keras.io/
- https://www.tensorflow.org/