Machine Learning Engineer Nanodegree

Capstone Project (Traffic Sign Classification: Deep Learning)

Rajesh Kumar
March 29th, 2017

Link to my project code

I. Definition

Project Overview

Self autonomous vehicles have gained a lot of attraction is last few years. One of the tasks that these vehicles have to perform to drive successfully in real traffic is to correctly recognize traffic signs.

In this project, I aim to create a traffic sign recognizer, which enables the self autonomous vehicle to correctly recognize the traffic sign captured by it's camera. This project builds a classifier trained using GTSRB Dataset [2]. The problem statement is inspired from Udacity's traffic sign classification project for carND [3].

Problem Statement

Fundamentally, this is **image classification problem**. An image classification pipeline gets an image as input and generates probability distribution of different predefined classes. For instance, in this figure, image recognition algorithm receives an image of a traffic sign and generates probability distribution as illustrated on the right side of the figure where correct category (class label 1 here) has the highest probability, while other classes have lower probabilities. Main goal is to improve the probability of correct class in this scenario (referred as true positive) while reducing the probability of all other classes (referred as false positive). The classification pipelines outputs the predicted class based on the probabilities obtained.

We approach this problem using some image processing and a popular supervised learning technique used for image classification (convolutional / deep neural networoks) [4, 5]. We follow the following approach.

- Download and preprocess the German Traffic Signs Data.
- Train a classifier that can predict the sign of input traffic sign image.
- Test the accuracy on *test set* and re-tune the classifier.
- Finalize the classifier and validate results on *validation set*.

Metrics

Like any other n-class image classification problem, the metric of central importance for us is *accuracy*. The final goal is to improve *accuracy* on the *validation set*.

*Accuracy = (Number of test samples correctly classified / Total number of test samples) * 100*

Please note that we have defined the metric accuracy for the test set (not for validation set and obviously not for train set). This is because, as we tain our model, we tend to use the outcomes of the vaidation set to make our model better. This leads to an indirect bleeding of validation data into training set. So, validation set accuracy should not be used as the final metric to judge how good our model is. For this purpose, we keep a separate isolated test set for final evaluation. To put it simply,

- Training set is used to fit the parameters [i.e., weights]
- Validation set is used to tune the parameters [i.e., architecture]
- Test set is used to assess the performance [i.e., generalization and predictive power]

Accuracy (fomula defined above) is the most relevent metric for us because, it tells us how accurately the model performs on the unseen data. i.e. how many percentage of test samples are recognized correctly. For example, Lets say, while the autonomous vehicle is driving on a road for an hour and it encountered 100 traffic signs. And it predicted 90 out of these 100 signs correctly. Then its accuracy is 90%. The model should hence focus on increasing this metric to make sure less traffic violations are made.

Other metric which could be of importance is the training time. But this is not super critical since, training is a one time effort. Once the model is deployed, it is accuracy which matters (predictions are typically superfast because they are just one round of mathematical calculations based on already learned parameters.)

II. Analysis

Data Exploration

Basic summary of the dataset is as follows.

- Each data point in this dataset is a color image of a traffic sign(3 channels).
- The shape of a traffic sign image is (32, 32, 3)
- The size of training set is 34799
- The size of test set is 12630
- The size of validation set is 4410
- The number of unique classes/labels in the data set is 43.

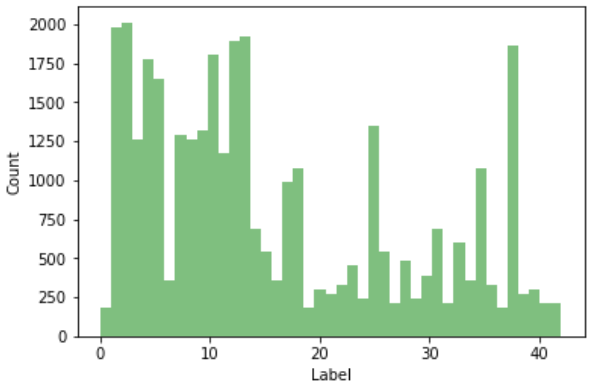
ClassId	SignName
0	Speed limit (20km/h)
1	Speed limit (30km/h)
2	Speed limit (50km/h)
3	Speed limit (60km/h)
4	Speed limit (70km/h)
5	Speed limit (80km/h)
6	End of speed limit (80km/h)
7	Speed limit (100km/h)
8	Speed limit (120km/h)
9	No passing
10	No passing for vehicles over 3.5 metric tons
11	Right-of-way at the next intersection
12	Priority road

ClassId	SignName
13	Yield
14	Stop
15	No vehicles
16	Vehicles over 3.5 metric tons prohibited
17	No entry
18	General caution
19	Dangerous curve to the left
20	Dangerous curve to the right
21	Double curve
22	Bumpy road
23	Slippery road
24	Road narrows on the right
25	Road work
26	Traffic signals
27	Pedestrians
28	Children crossing
29	Bicycles crossing
30	Beware of ice/snow
31	Wild animals crossing
32	End of all speed and passing limits
33	Turn right ahead
34	Turn left ahead
35	Ahead only
36	Go straight or right
37	Go straight or left
38	Keep right
39	Keep left
40	Roundabout mandatory
41	End of no passing
42	End of no passing by vehicles over 3.5 metric tons

Exploratory Visualization

Some examples from the dataset are given below. These labels can be verified using the above table.

The following histogram shows the count of training examples for all the 43 classes present in the dataset. This is to get an intuition of over-representation / under-representation of a given class. It is important to make sure that the dataset is not biased towards a few classes. As we can see in the histogram, many classes have far less number of samples than mean number of samples. This should be handled before working with the data to avoid any bias.



Algorithms and Techniques

Before we start looking at how the classifier is implemented, it is important to understand the algorithms and techniques which will be used in the implementation.

Artificial Neural Networks and Deep Neural Networks

Artificial neural networks (ANN) is a computational model based on the structure and functions of biological neural networks. It is used in computer science and other research disciplines. It is basically a large collection of simple neural units (artificial neurons). Each neural unit is typically joined to many others, and links can boost or weaken the activation state of neural units joined to this unit. Each neural unit does computation using summation function. On the top of which a threshold is applied. Threshold is used such that the signal must surpass the limit before getting propagated to other neurons. They differ from a traditional computer program in the sense that they exhibit 'self-learning behaviour'.

A deep neural network (DNN) is basically an artificial neural network (ANN) with many hidden layers between the input and output layers. Like ANNs (which are shallow), DNNs can also model non-linear relationships. DNN architectures, for example, for object detection , generate compositional models where the object is expressed as a layered composition of image

primitives. The extra layers enable composition of features from lower layers, giving the potential of modeling complex data with lesser units than a similarly performing not-deep network.

- *Backpropagation*: We can discriminatively train a DNN with the standard backpropagation algorithm. Backpropagation (The backward propagation of errors), is a popular method for training ANNs and used in conjunction with an gradient descent (or other optimization methods). Propagation and weight update are the two main tasks involved. When an input feature vector is fed to the network, it is propagated forward, until it reaches the output layer. We then compare the output of the network to the desired output, using a loss function, and calculate an error value for each of the neurons in the output layer. The backpropagation of the error values takes place, starting from the output, until each neuron has an associated error value (roughly represent its contribution to the original output).
- *Gradient Descent*: Gradient descent is a simple and popular first-order iterative optimization algorithm. In gradient descent, one takes steps proportional to the negative of the gradient of the function at the current point, to find a local minima of the function under consideration. Positive of the gradient leads to local maxima.

CNNs (Convolutional Neural Networks) and their architecture

It is well known that the CNN outperforms many image recognition algorithms in ImageNet dataset [6]. Convolutional Networks are similar to Neural Networks where neurons are replaced by the convolutional operation at the initial layers. We can do such a replacement if we consider focus on a similar features at different positions of the image which is achieved by convolutional filters. At the later stages, neurons in the Neural Network are also used in Fully Connected layer of Convolutional Network. CNN algorithm typically consists of several convolution operations followed of the image sequentially which is followed by pooling operation to generate the neurons feed into fully connected layer.

- *Convolutional Layer*: Convolution is an operation of filtering the input data. In image recognition context, 2D filters are used to process data at different layer to generate the features. These filters are pre-defined and their coefficients are computed during training (see backpropagation and Gradient descent sections above).
- *ReLU and Pooling Layer*: This layer contains both the activation operation that is applied to each elements after convolution and subsampling of the data after activation. Activation operation is typically a nonlinear operation that is applied to each elements of convolution output such as max(0, x)(which is also called Rectifier Linear Unit) or $(1/ (1 + e^{x}))$ where x is the input data. Activation operation does not change the size of the input. Subsampling is applied after activation that reduced the size of the input to typically 1/2 at each dimension. Window size that is used during subsampling is also 2D such as 2x2 or x3. If the input data size is (W,H) then the size after pooling will be (W/2,H/2) if 1/2 subsampling is used.
- *Dropout Layer*: Dropout is a regularization technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data. It is a very efficient way of performing model averaging with neural networks. The term "dropout" refers to dropping out units (both hidden and visible) in a neural network.
- *Fully Connected Layer*: This layer will reduce the size of input data to the size of classes that the CNN is trained for by combining output of Convolutional Layer layer with different weights. Each neuron at the output of the Convolutional Layer layer will be connected to all other neurons after weighted properly, Similar to Convolutional layer, weight of these taps in Fully connected layer is found though backpropagation algorithm.
- *Classification Layer*: Classification Layer: This is the final layer of the CNN that converts the output of fully connected layer to probability of each object being in a certain class. Typically soft-max type of algorithms are used in this layer.

Benchmark

In the year 2011, a competition was held to recognize traffic sign and the same dataset was used [1]. The winning team achieved a recognition rate of 99.46 % which is higher than recognition rate achieved by human beings (reported 98.84 %)

Rank	Team	Method	Correct recognition rate
1	IDSIA	Committee of CNNs	99.46 %
2	INI	Human Performance	98.84 %
3	sermanet	Multi-Scale CNNs	98.31 %

Rank	Team	Method	Correct recognition rate
4	CAOR	Random Forests	96.14 %

These implementations are very advanced and its very difficult to achieve such a recognition rate with few months of effort. Keeping this in mind, I aim to achieve a test set accuracy of 90% (with higher validation accuracy) as a soft-target. Hard-target is beating human performance.

III. Methodology

Data Augmentation

I decided to generate the additional data because of following two reasons

- As we have seen in the frequency histogram, few classes have very less number of samples. This can lead to a bias towards the classes with much higher number of samples.
- Initially the accuracy of my model was low, so I tried augmenting synthetic data to see if it helps.

For all the classes which have less than mean number of training samples, I augmented synthetic samples to bring this number equal to mean.

These synthetic images were generated by following technique.

- Brightening -> Warping -> Scaling -> Translating the image by a random factor.

Shape of training data before augmentation: (34799, 32, 32, 1) Shape of training data after augmentation: (46714, 32, 32, 1)

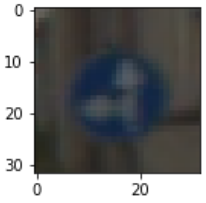
After augmenting the data with these images, data was shuffled.

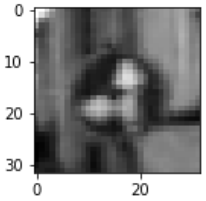
Data Preprocessing

Primarily, I used two preprocessing steps.

- **Grayscale conversion**: I converted the images to grayscale by averaging out values across the depth dimension. This results in change in image dimension from (32, 32, 3) to (32, 32, 1). Intuitively, the loss of information in this conversion should not impact the task of traffic sign classification much. Also, the reduction in size helps in reducing the training time.

Here is an example of a traffic sign image before and after grayscaling.





- **Normalization**: Ideally, we want out variables to have close to zero mean and equal variance (roughly). If this is not the case (badly conditioned problems), the optimizer has to do a lot of search to find the solution. So, we normalize our images in the range (-1, 1) using X_train_normalized = (X_train - 128)/128.

Implementation

The traffic sign images are typically composed of basic geometrical shapes, and hance convolutional layers can be very effective in squeezing out the in the semantics of the image. Broadly speaking, the CNN learns to recognize basic lines and curves, then shapes and blobs, and then increasingly complex objects within the image. Finally, the CNN classifies the image by combining the larger, more complex objects.

I have used two convolutional layers followed by three fully connected layers, where the length of output of last layer equals to the number of classes. It is inspired from LENET architecture [9]. It is well known that LENET architecure works well for images in MNIST data set [8].

For both the convolutional layers, the convolution is followed by ReLU (for adding non-linearity in the network in a simple way). ReLU is followed by dropout and maxpool.

To sensibly reduce the spatial extent of feature map of the convolutional pyramid, rather than using stride of size two (which is aggressive), using 2x2 maxpooling is less agressive and preserves more info than aggresive striding.

To prevent overfitting, we use a dropout after every layer with keep_prob=.50. Because of random dropout, the network can not rely on any activation to be present, because they might be destroyed at any given moment. So, it is forced to learn a redundant representation of everything.

Fully connected layers help in squeezing out the dimensions such that finally the length of the output of length of the class.

My final model consisted of the following layers:

- Layer : 1
    - 5x5 convolution (32x32x1 in, 28x28x6 out)
    - ReLU
    - Dropout with keep_prob 0.50
    - 2x2 max pool (28x28x6 in, 14x14x6 out)
- Layer : 2
    - 5x5 convolution (14x14x6 in, 10x10x16 out)
    - ReLU
    - Dropout with keep_prob 0.50
    - 2x2 max pool (10x10x16 in, 5x5x16 out)
    - Flatten. Input = 5x5x16. Output = 400
- Layer : 3
    - Fully Connected. Input = 400. Output = 120
    - ReLU
    - Dropout with keep_prob 0.50
- Layer : 4
    - Fully Connected. Input = 120. Output = 84
    - ReLU
    - Dropout with keep_prob 0.50
- Layer : 5
    - Fully Connected. Input = 84. Output = 43.

These 43 outputs are softmax probabilities corresponding to each of the pre-defined classes. The one with highest probability is considered the correctly predicted class.

I used **AdamOptimizer** [7] for optimization. Other settings I used are following.

- batch size: 64
- epochs: 50
- learning rate: 0.001
- mu: 0
- sigma: 0.1
- dropout keep_prob: 0.5

Refinement

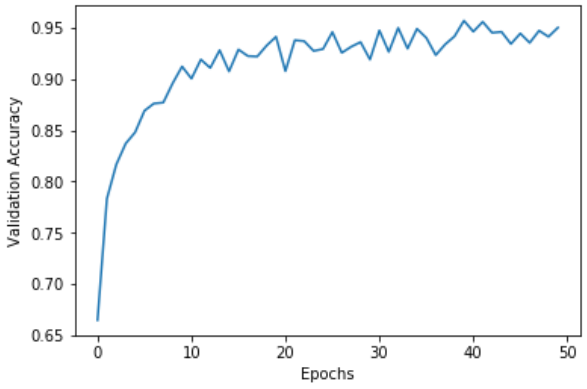
After trying various combination of parameters, i used the setting described above. Apart from parameter tuning, I used following approach.

- Ran the model with default parameters. (Validation Set Accuracy: 77%)
- Modified the model by adding dropouts after each layer (apart from the final layer) with keep_prob =0.50 (Validation set accuracy increased to 85%)
- Tried playing with the learning rate and batch size. (Froze them to the values specefied above, Validation set accuracy increased to 91%)
- Augmented training set with the synthetic images. (Validation set Accuracy: 95.5%)

IV. Results

Model Evaluation and Validation

I first evaluated the performance of the model on validation set. Validation accuracy immproves from 66% to 95.5% in 50 epochs.



Testing the same model on test set gives an accuracy of 93.2% which I think is pretty impressive.

Testing with new images and challenges involved

Apart from this test set, I downloaded 5 random traffic sign images from the web and analyzed the performance of our model on these images.

Here are the images I got.

Though images look good, it might be difficult to classify them because

- It has no borders. In contrast, images in GTSRB dataset has around 10% border.
- These images look brighter than the images in the GTSRB dataset.

Predictions on new traffic images

Here are the results of the prediction:

Image	Prediction
General caution	General caution
Turn left ahead	Turn left ahead
Speed limit (30km/h)	Speed limit (30km/h)
Priority road	Priority road
Keep right	Keep right

The model was able to correctly guess 5 of the 5 traffic signs, which gives an accuracy of 100%. This gives me confidence that the model does well on real life traffic signs. I analyzed 'With what confidence the predictions has been made' using softmax probabilities (discussed in conclusion section).

Comparison with test set prediction accuracy

If we compare this accuracy (100%) to with the test set accuracy (93.2 %), it seems to be performing better on newly acquired images. This might be due to the fact that number of images in the newly acquired image set is very low (only 5). Also, the model might be good at performing classification for these kind of images due to unknown factors such as adequate availability of similar images in the training set. In spite of 100% accuracy on new images, I doubt that the model is overfitting to some extent, because test set accuracy is lower than the validation set accuracy.

Comparison against the benchmark

Lets revisit our performance summary.

- Validation set Accuracy of 95.5%
- Test set Accuracy of 93.2%
- Accuracy on 5 random images (not from the dataset): 100%

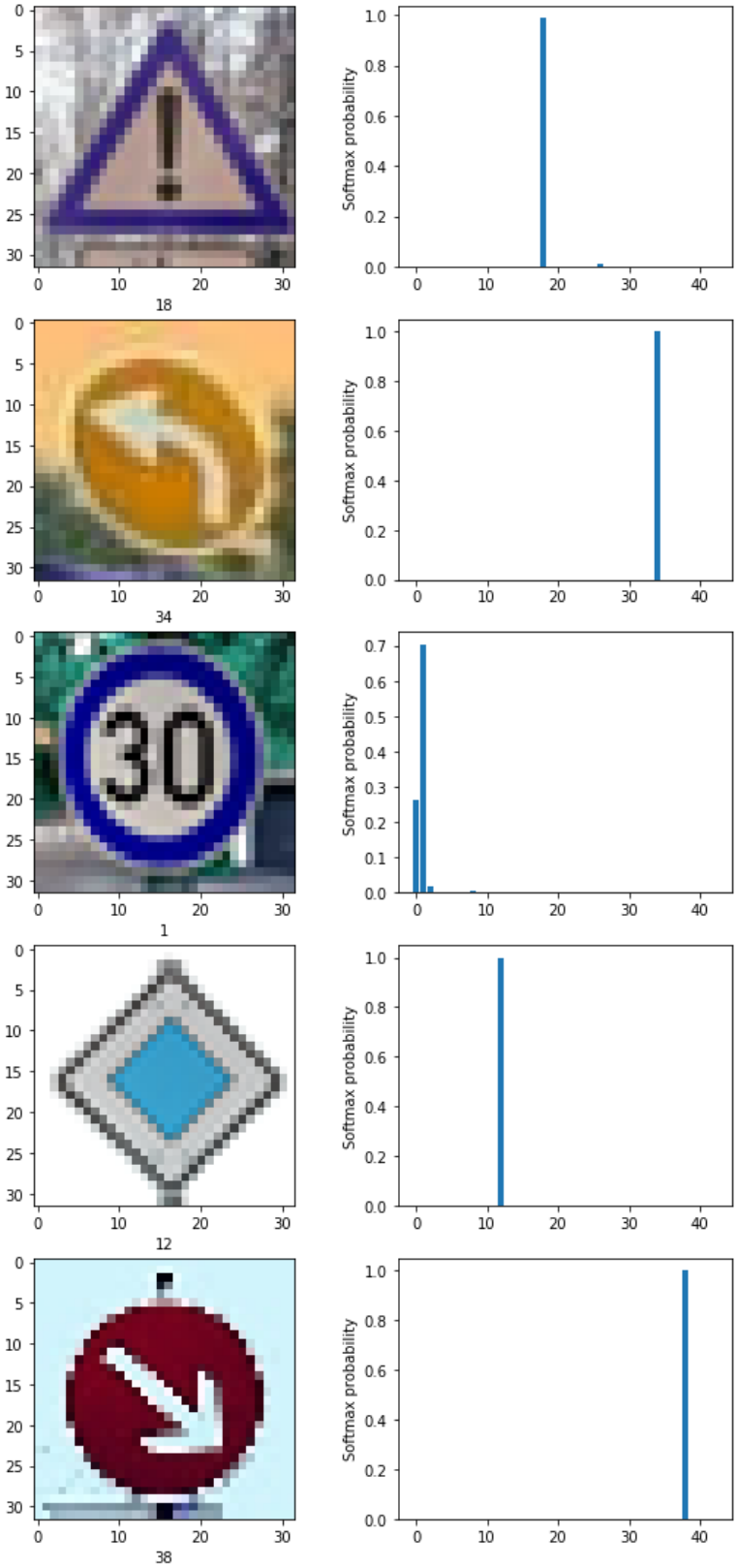
We had started with a soft aim of achieving 90% test set accuracy. We have achieved an accuracy of 93.2% on test set. While it is 5% away from human performance, it should be considered good because.

1. It performed extremely well on unseen images (not a part of original dataset).
2. In a real life driving scenario, to avoid traffic violations, some amount of extra code will help us sail through the deficiency in our performance. For example, if our classifier is not confident enough (say, less than 99% probability for the most probable class), it can take the safest option. Like a speed limit of 50 can be treated as speed limit of 10 (or whatever lowest speed limit is).

V. Conclusion

Free-Form Visualization

Lets use softmax probabilities to understand 'with what confidence the predictions has been made' and 'kind of confusion classifier might be facing'.



Classifier classifies 3 images (Turn left ahead, Priority road, Keep right) with softmax probability of 1, which means that it classifies these images with full confidence.

For General Caution Image, it classifies correcly with softmax probability of 0.98. With a probability of 0.02 it confuses this image with 'pedestrians' image. This is because these images look pretty similar.

For 'Speed limit (30km/h)' image it confuses it with Speed limit (20km/h) and Speed limit (50km/h) images. Again, these images are pretty similar in appearance.

Reflection

Implementation Summary

- We started with the realization that few classes are suffering from under-representation. We augmented synthetic data in the dataset to solve this.
- To reduce the size of the feature vector, we used grayscaling.
- To make the problem well-conditioned (to reduce optimizer's work), we normalized the inputs.
- We took inspiration from basic LENET implementation and started building upon it. We used two convolutional layers followed by three fully connected layers, where the length of output of last layer equals to the number of classes.
- We added ReLU units for introducing non-Linearlity in the network.
- To sensibly reduce the spatial extent of feature map of the convolutional pyramid, we used maxpooling.
- We used dropout for preventing overfitting.
- We experimented with different parameters, specially learning rate and batch size.
- We tested our model on validation set, test set and unseen image data.

What I found interesting

- Just by adding dropout layers, the validation set accuracy improved by 8%. Dropout seems like a trivial and too-simple concept, but is extremely effective. Dropping activations prevents the network from being overconfident.
- Initially, I tried the same process with the original dataset (without augmentation). Augmentation took care of bias and gave us an accuracy improvement of over 4%. This made me realize that, sometimes working on dataset is more important than working on processing/algorithms.

What I found difficult

- Training was very slow on my laptop. It used to take me 13 minutes on average to train the model. I decided to use GPU version of tensorflow. For this I setup Nvidia GTX 680 on a desktop system. This reduced the training time to 1.5 minutes.
- I wasted a lot of time in tuning parameters, and got stuck at a point where I was not seeing any improvement. It took me a while to figure out that simple techniques like dropout and augmentation work well.

Potential Improvements

- As seen in benchmarks section of this report, people have achieved better results with a committee of CNNs or multi scale CNNs. Though difficult to implement and tune, it is worth giving a try. Committee of CNNs might be useful in eliminating the doubt (less softmax probability) that one CNN might have. Different CNNs can be trained from different perspectives (shape, color, brightness etc.) and the majority vote can be taken to finally decide what the most likely solution is.

References

[1] http://benchmark.ini.rub.de/

[2] http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset#Overview

[3] https://github.com/udacity/CarND-Traffic-Sign-Classifier-Project

[4] https://en.wikipedia.org/wiki/Convolutional_neural_network

[5] https://en.wikipedia.org/wiki/Deep_learning

[6] http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neuralnetworks.pdf

[7] https://www.tensorflow.org/api_docs/python/tf/train/AdamOptimizer

[8] http://caffe.berkeleyvision.org/gathered/examples/mnist.html

[9] http://yann.lecun.com/exdb/lenet/