Udacity Self Driving Car Nanodegree –

Project: Traffic Sign Classifier

By:

Saurabh Sohoni

Udacity profile: <https://profiles.udacity.com/p/u183059>

Contents

1. Introduction 3

2. Project Goals 3

3. Implementation of Project Rubric Points 3

3.1 Submission of code and output files 3

3.2 Dataset Exploration 4

3.3 Design and Test a Model Architecture 5

3.4 Model Test on New Images 9

4. References 12

# Introduction

This document acts in support with code and output model created in order to implement ‘Traffic Sign Classifier’ project. In this project, a deep learning network is trained on traffic images taken from the German Traffic Sign Dataset [1] using TensorFlow [2]. The trained model is then used to determine accuracy on validation and test datasets. Also, the trained model is used to predict classification on traffic sign images taken from the internet and the results are summarized.

# Project Goals

Following were the goals of this project:

1. Load traffic sign images taken from the German Traffic Sign Dataset [1]. These images were provided by Udacity in files represented in pickle (.p) file format. These pickle files contained images and information on the class to which an image belonged. In total, there were 43 different classes used for classifying an image.
2. Double check if each image in the dataset is labelled with a class.
3. Explore the dataset by checking the number of samples present in training, validation and test sets. Also, plot few images and its respective class to verify that the labelled data is correct for a few samples. Also, explore into the distribution of dataset across different classes.
4. Pre-process the images to ensure training of the network is fast and that the network doesn’t stuck in local minima.
5. Define and implement a neural network which includes convolutional layers, fully connected layers, pooling layers and activation functions to identify intricate features in traffic sign images.
6. Train the network on training dataset and evaluate its performance on validation dataset. Tune different parameters such as the learning rate, number of epochs, batch size, weight initialization parameters, etc. to obtain maximum accuracy.
7. Test the network on test dataset and 5 traffic sign images downloaded from the internet and summarize the results.

# Implementation of Project Rubric Points

Following section lists down various rubric points for this project and also details out the implementation strategy followed:

## Submission of code and output files

The submission includes following files:

1. Traffic\_Sign\_Classifier.ipynb – This iPython notebook contains code to build convolutional neural network (referred to as CNN, henceforth), train the model of training data set and test the model on test dataset and 5 traffic sign images from the internet. Output for code implemented in each step is displayed after corresponding cell.
2. Traffic\_Sign\_Classifier.html – This file is an export of the iPython notebook described above.
3. writeup\_report.pdf – Current file summarizing about the project goals and results obtained.

## Dataset Exploration

Traffic sign images taken from the German Traffic Sign Dataset [1] were provided by Udacity in pickle (.p) file format. These files contain images and class label for which the image belongs.

1. Images in the dataset can be classified into one of the 43 classes described below. Also, a sample image for each class was visualized to get some insight into different classes. This is shown below:



Figure 1: Image from training dataset for each class. Class ID starting from 0 from top left and going by each row with 42 for image in the last row [5a]

1. The complete dataset was divided into training, validation and test sets each with following number of samples:
   * Training set – 34,799 samples
   * Validation set – 4410 samples (Roughly 12% of number of training samples)
   * Test set – 12630 samples (Roughly 35% of number of training samples)
2. The distribution of samples per each class was not even. This is evident from the histogram shown below:

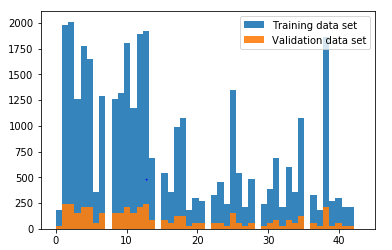


Figure 2: Distribution of samples across classes [5b]

It was found that there were fewer than 500 image samples for a few classes.

## Design and Test a Model Architecture

#### **3.3.1 Data Pre-processing**

Convolutional neural networks converge faster when the training data is centered around zero and has same covariance. This is achieved by using mean subtraction of pixel values in an image followed by normalization described below:

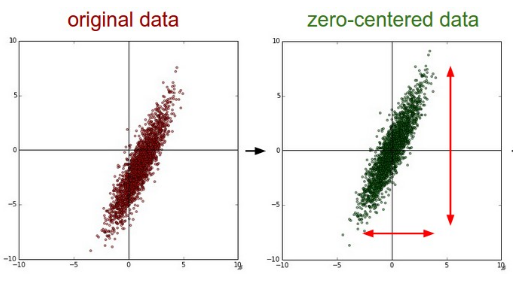
1. Mean subtraction is the most common form of data pre-processing. In this process, the mean of all features in an image is subtracted from every pixel. This process basically centers the cloud of data around zero, or origin. This is demonstrated below:

Figure 3: Original data on the left and mean subtracted data on the right. Image taken from Convolutional Neural Networks for Visual Recognition (CS231n) class notes [5c]

1. In case of feature extraction in an RGB image, pixel values for each channel range from 0 to 255. But when this data is fed to a CNN, each pixel value is just some raw data and no relation to visual color information. This results in biased judgement for pixels with higher values than for the ones with lower values. To avoid this bias, all features are fed to a normalizer. Normalization results in achieving approximately same scale for different data dimensions. Graphical representation is shown below:

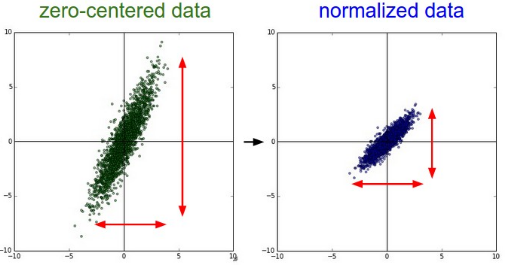


Figure 4: Mean subtracted data on the left and normalized data on the right. Image taken from Convolutional Neural Networks for Visual Recognition (CS231n) class notes [5d]

In this project, data was normalized around 0 or center.

#### **3.3.2 Model architecture**

For this project, LeNet architecture published in the paper ‘Gradient-Based Learning Applied to Document Recognition’ [3] was used along with use of dropout layer to avoid overfitting to training data. Model architecture is shown and described below:

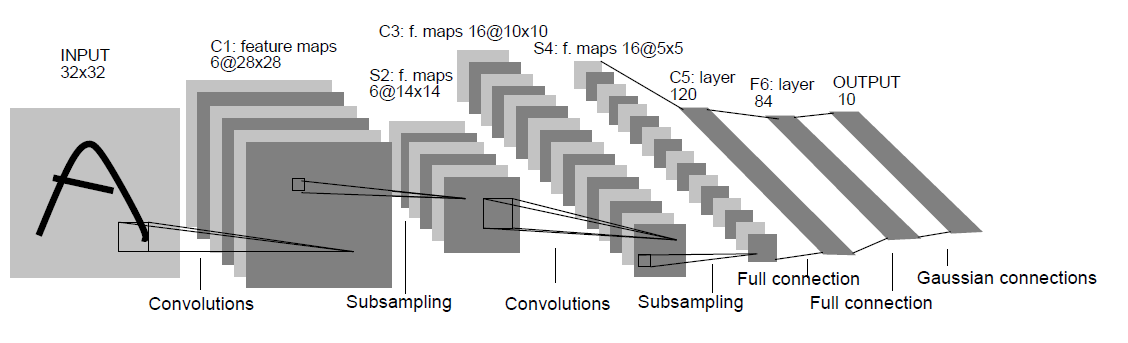


Figure 3: Image taken from original paper published by Yann LeCun ET. AL. in the paper ‘Gradient-Based Learning Applied to Document Recognition’ [5e]

1. The dimensions for image vector at the input layer is 32 x 32 (width x height). LeNet was designed to be used on gray scale images. However, in this project, RGB images were used. Hence, the dimensions of input layer were 32 x 32 x 3.
2. Input image was then fed to a convolutional layer with filter size of 5 x 5 and stride of 1 x 1 with valid padding. 16 output layers were created resulting in output of size 28 x 28 x 6.
3. This is followed by Exponential Linear Unit (referred to as ELU, henceforth) activation. Use of ELU activation introduces non-linearity in the network. ELU tries to make the mean activation close to zero and as it is an exponential function, it does not saturate. Also, ELU prevents the problem of vanishing gradients as it uses exponential function for activation. Hence, ELU was found beneficial over Rectified Linear Unit (ReLU) activation.
4. After activation, a subsampling layer is introduced in the network. This is nothing but an average pooling layer of filter size 2 x 2 and stride of 2 x 2 with valid padding. The resulting output layer has the size of 14 x 14 x 6. This layer helps in filtering out activations with low values.
5. The combination of convolutional layer-ELU-average pooling is employed again to get an output of shape 5 x 5 x 16.
6. Output obtained from last average pooling layer is then flattened out to a single row vector and is fed to 2 fully connected layers. Each fully connected layer is followed by ELU activation.
7. The first fully connected layer is followed by a dropout layer with 0.7 as keep\_probability. Dropouts ensure the network doesn’t overfit to training data.
8. Input layer accepting images of size 160 x 80 x 3 (3 planes for colored images).
9. Batch Normalization layer just after Input layer to normalize the dataset.
10. Three convolutional layers with 5 x 5 filter kernel, each layer followed by a ReLU (Rectified Linear Units) activation to introduce non-linearity.
11. Two convolutional layers with 3 x 3 filter kernel, each layer followed by a ReLU (Rectified Linear Units) activation.
12. Vector space was flattened to run fully connected layers here after.
13. Two fully connected layers, each followed by a ReLU activation and a dropout layer with 50% as keep probability.
14. This problem being a regression problem, last fully connected layer with 1 x 1 output predicting the steering angle measurement was employed at the output node.

#### **Model Training**

The model was trained using following hyper-parameters:

1. Weights initialization

Initialization of weights correctly plays an important role in faster convergence of model and optimized learning in each epoch. Since ELU activation was used, weights cannot be initialized as random with zero mean and zero sigma. In the initial stages of this project, each weight was initialized with random normal distribution with 0 mean and 0.1 covariance. In the later stages, Xavier initialization, described in the paper ‘Understanding the difficulty of training deep feedforward neural networks’ [4] was used as model would converge faster with same overall accuracy on validation dataset.

1. Adam optimizer

For training, Adam optimizer was used. Adam optimizer is an adaptive optimizer which decreases the learning rate as the number of epochs increases. This ensures that the model doesn’t get stuck in local minima and network loss is decreased gradually. As it uses moving averages of the parameters (momentum) it enables Adam to use a larger effective step size, and the algorithm will converge to this step size without fine tuning.

1. Number of epochs and batch size

With reducing the batch size, the network converged in less number of epochs. In this implementation, batch size of 32 was chosen training was done until 20 epochs.

#### **Solution approach**

Using the combination of hyper parameters described above, the network was able to achieve validation accuracy of roughly 93.5% or 0.935. The results for each epoch are shown below:

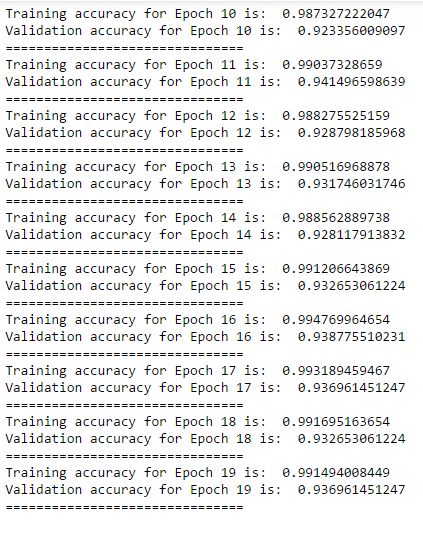
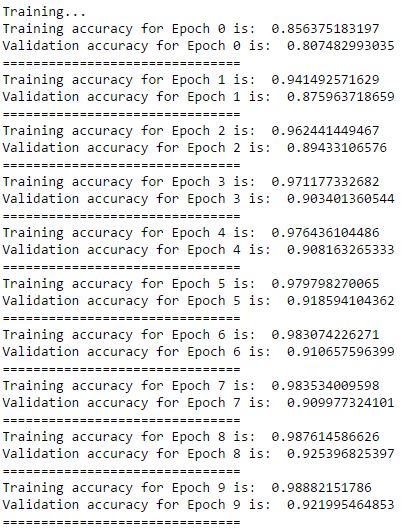


Figure 4: Network accuracy on training and validation dataset for each epoch [5f]

Also, the trained model was able to achieve an accuracy of 0.925 as shown below:

Figure 5: Result of model on test dataset [5g]

## Model Test on New Images

The model trained above was saved in file named model.ckpt. This model was then loaded into TensorFlow session and prediction was carried out on new test images. Following section explains on how the images were acquired and the prediction of class done by trained model for each image:

#### **3.4.1 Acquiring new images**

All five images were acquired from the internet. The size of each image was different and hence in pre-processing step, they were all cropped to 32 x 32 x 3 as required by Input layer of network architecture. Images with their correct classes are shown below:



Figure 6e: Stop

**Correct class ID – 14**

**[5l]**

Figure 6d: Ahead only

**Correct class ID – 35**

**[5k]**

Figure 6c: Yield

**Correct class ID – 13**

**[5j]**

Figure 6b: Speed limit (60km/h)

**Correct class ID – 3**

**[5i]**

Figure 6a: Turn left ahead

**Correct class ID – 34**

**[5h]**

Figure 6a and were difficult to classify as they were in blue color as opposed training images, which were in black with white in the background. Also, as these images belonged to classes 33 and 34 which had less image samples in the training dataset, classification will be a success if model generalizes effectively. Figure 6b was difficult to classify as the overall brightness in the image is low enough to mask the number 60 within the white circle.

#### **3.4.2 Performance on new images**

The trained model predicted correct classes for figure 6a and figure 6c, while it failed on all other images. Hence, an accuracy of 40% or 0.4 was achieved on new test images. Results are shown below:

Figure 7: Accuracy on new images and the predicted class for each image respectively [5m]

#### **3.4.3 Model Certainty – Softmax Probabilities**

For the predictions made by the trained model on new images, the top five softmax probabilities were extracted to measure its certainty in predictions. The model was very much certain in detecting a class for an image. The softmax probability for first prediction was much higher (almost close to 1) as compared to other four. The results for each image are summarized below:

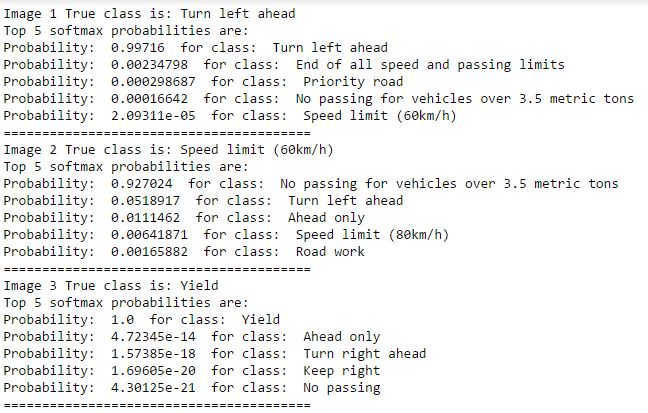
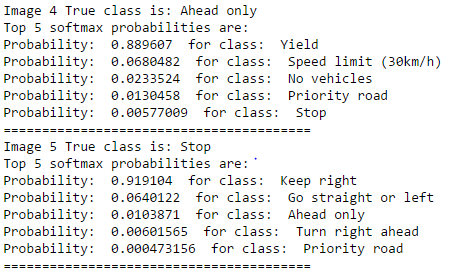


Figure 8: Top 5 softmax probabilities predicted by model for each image from new images [5n]

# References

1. [1] [German Traffic Sign Data](1.%09http:/benchmark.ini.rub.de/?section=gtsrb&subsection=dataset)
2. [2] [TensorFlow An open-source software library for Machine Intelligence](https://www.tensorflow.org/)
3. [3] Yann LeCun. ET. Al. , ‘Gradient-Based Learning Applied to Document Recognition’ , Proc. of the IEEE, November 1998
4. [4] Xavier Glorot and Yoshua Bengoi, ‘Understanding the difficulty of training deep feedforward neural networks’, DIRO, Universite de Montr ´ eal, Montr ´ eal, Qu ´ ebec, Canada
5. [5] Image references:
   1. /examples/image-class-sprite.jpg
   2. /examples/sample-distribution-histogram.JPG
   3. /examples/mean-subtraction.JPG
   4. /examples/normalization.JPG
   5. /examples/lenet-architecture.JPG
   6. /examples/traffic-accuracy.jpg and /examples/traffic-accuracy-2.jpg
   7. /examples/test-accuracy.JPG
   8. /test-images/german-test-1.jpg
   9. /test-images/german-test-2.jpg
   10. /test-images/german-test-3.jpg
   11. /test-images/german-test-4.jpg
   12. /test-images/german-test-5.jpg
   13. /examples/test-images-accuracy.JPG
   14. /examples/softmax-1.JPG and /examples/softmax-2.JPG