Udacity Self-Driving Car Nanodegree
Term One: **Computer Vision** & **Deep Learning**
Project Two: **Traffic Sign Classification**

Nishan Srishankar
Worcester Polytechnic Institute

2017

1 Introduction

One of the fundamental aspects in implementing a self-driving car is perception and deals with how a car would identify aspects of interest that are necessary for its functioning or could impede its performance. This could include traffic signs, signals, neighboring vehicles, lane lines, and even chaotically behaving humans or cyclists. This project deals with classifying traffic signs using supervised learning. In real-life, traffic signs would have to be detected in a region of interest before being classified, and then acted upon.

2 Methodology & Implementation

Three pickled datasets (train, valid, and test) from the German Traffic Sign dataset are given as a starting point, which are split according to a 0.67-0.09-0.24 ratio. The datasets contain traffic sign images with dimension (32,32,3) and belong to one of 43 classes as seen in Table 1. The training dataset contains 34,799 such images which is used to train a custom Convolution Neural Network, tuned on the valid dataset, and verified on the test image dataset and new images.

Table 1: Analysis of traffic sign training set

Class Index	**Sign Name**	**Count**
0	Speed limit (20 km/h)	180
1	Speed limit (30 km/h)	1980
2	Speed limit (50 km/h)	2010
3	Speed limit (60 km/h)	1260
4	Speed limit (70 km/h)	1770
5	Speed limit (80 km/h)	1650
6	End of Speed limit (80 km/h)	360
7	Speed limit (100 km/h)	1290
8	Speed limit (120 km/h)	1260
9	No passing	1320
10	No passing for vehicles over 3.5 metric tons	1800
11	Right-of-way at the next intersection	1170
12	Priority road	1890
13	Yield	1920
14	Stop	690
15	No vehicles	540
16	Vehicles over 3.5 metric tons prohibited	360
17	No entry	990
18	General Caution	1080
19	Dangerous curve to the left	180
20	Dangerous curve to the right	300
21	Double curve	270
22	Bumpy road	330
23	Slippery road	450
24	Road narrows on the right	240
25	Road work	1350
26	Traffic signals	540
27	Pedestrians	210
28	Children crossing	480
29	Bicycles crossing	240
30	Beware of ice/snow	390
31	Wild animals crossing	690
32	End of all speed and passing limits	210
33	Turn right ahead	599
34	Turn left ahead	360
35	Ahead only	1080
36	Go straight or right	330
37	Go straight or left	180
38	Keep right	1860
39	Keep left	270
40	Roundabout mandatory	300
40	End of no passing	210
42	End of no passing by vehicles over 3.5 metric tons	210

2.1 Preprocessing

The datasets are simply preprocessed by conversion to grayscale, followed by normalizing to have zero mean, and a small standard deviation. The dataset is also repeatedly shuffled. A grayscale colorspace was chosen rather than leaving images in RGB from literature review into the LeNet architecture.In addition to grayscaling and normalizing, adaptive histogram equalizing was also looked into for the initial iterations of this project but was then removed.

2.2 Data Augmentation

If classes are severely underrepresented, images are randomly rotated, randomly translated, sheared, and scaled. In addition to this, some classes can be further augmented as seen in Table 2. Images in some classes are invariant in which they can be either flipped vertically or flipped horizontally and still belong to the same class, or even be flipped vertically and used to augment another class (Flip Labels).

Table 2: Secondary augmentation

Augmentation	**Class Index**
Flip Vertical	9,10,11,12,13,15,17,18,21,22,23,25,26
,	27,28,29,30,31,32,35,40,41,42
Flip Horizontal	1,5,7,9,10,12,15,17,32,38,39,40,41,42
Flip Labels	[19,20],[33,34],[36,37],[38,39]

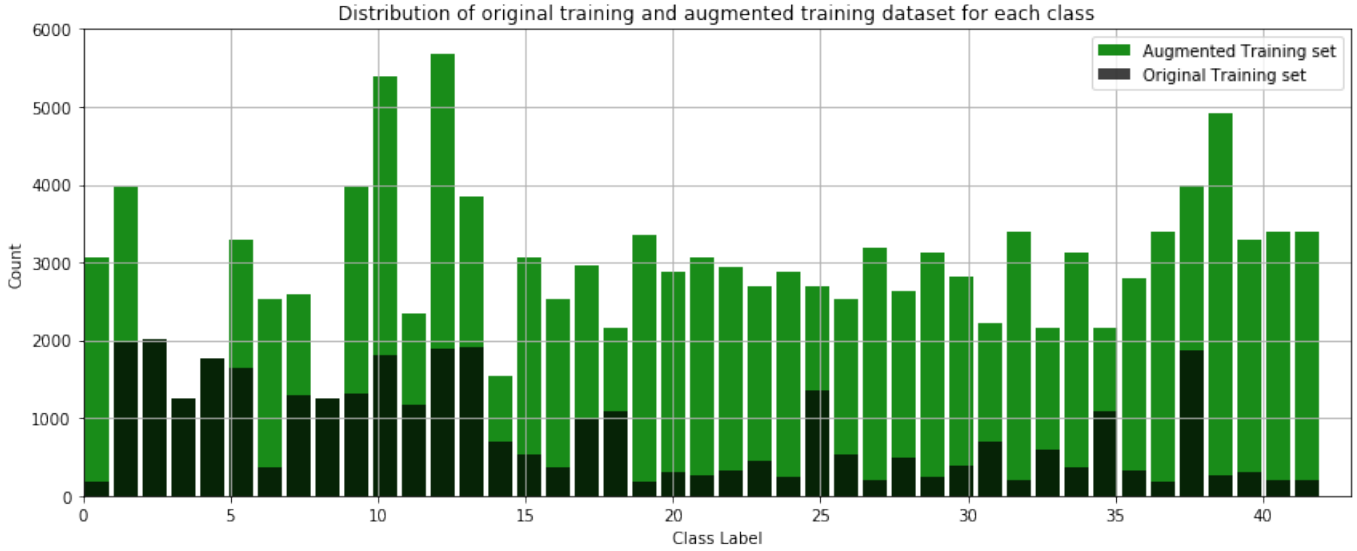


Figure 1: Augmented class representations

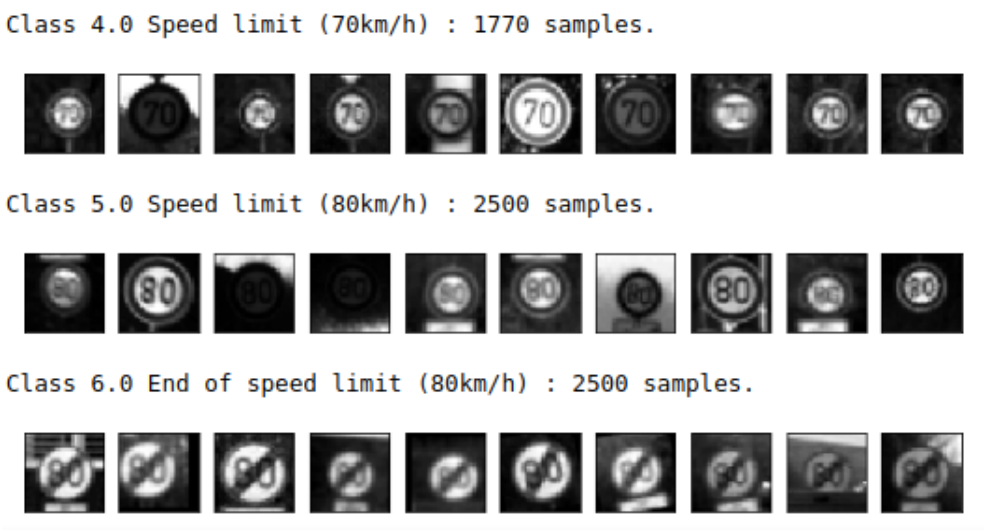


Figure 2: Sample augmented images

Augmentation increases the dataset length from 34,799 images to 128,192 images (which is almost a factor of three!).

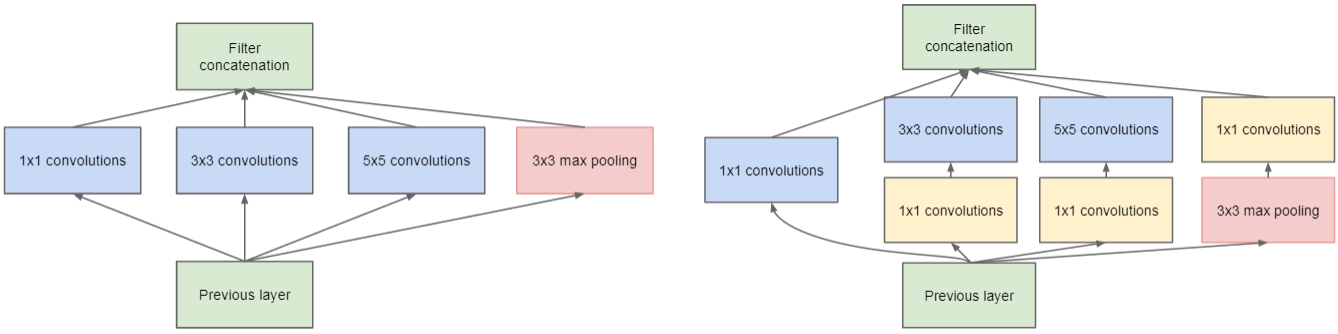
2.3 Balancing Classes

Data augmentation is also done such that classes won't be misclassified due to under-representation as seen in Figure 1, however it can be seen as the dataset is still unbalanced. In an attempt to balance classes, the augmented dataset is pruned to have approximately 2500 images per class for better representation which brings down the training dataset to 101,373 images. By hashing each of the images in the datasets, it was seen that there are no duplicate images of the training dataset in the validation, or test datasets which could result in skewed accuracies.

2.4 Convolution Neural Network

Various convolution networks were tried out on the balanced-augmented training set described above. The LeNet-5 architecture was initially tested as a benchmark and was seen to obtain a 93 % accuracy on the training dataset. Additional dropout or regularization techniques could improve this accuracy. As a means of understanding the material taught in this course, two additional architectures were analyzed: VGG-16, and Inception-module. The VGG-16 architecture resulted in a training accuracy of approximately 94 %. The inception architecture that was tested out consisted of a two convolution layers as a stem, followed by an inception module seen in Figure 3 (Inception-V3 uses around 8), followed by two full-connected layers and a classifier as the root. The issue with the two additional architectures was the sheer time it took for training parameter-heavy architectures, with even the simple, dimensionally-reduced Inception architecture taking approximately 300 s- 500 s for an epoch (one forward followed by back-propagation pass) on an AWS instance. This

meant that learning-rate, batch-size and additional hyper-parameters couldn't be trained effectively.



(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 3: Inception Modules

The final CNN architecture that was chosen was relatively simple as seen in Table 3 and takes in grayscale images with hyper-parameters seen in Table 4. The CNN is better generalized by implementing dropout layers to reduce over-fitting and help in generalizing, and L2-regularization to penalize large weights and biases. An Adams optimizer was used instead of vanilla Gradient-Descent since this intrinsically implements learning-rate decay as well as momentum and is very useful for better convergence and to prevent overshooting.

Table 3: Chosen CNN architecture

Layer No.	**Detail**	**Description**
0	Input Layer	Shape= (32,32,1)
1	Convolution w\RELU activation	Kernel: (5,5) Stride: (1,1,1,1) Padding: Same Input Feature Map: 1 Output Feature Map: 16
2	Max-Pooling Layer	Kernel: (1,2,2,1) Stride: (1,2,2,1) Padding: Same
3	Convolution w\RELU activation	Kernel: (3,3) Stride: (1,1,1,1) Padding: Same Input Feature Map: 16 Output Feature Map: 32
4	Max-Pooling Layer	Kernel: (1,2,2,1) Stride: (1,2,2,1) Padding: Same
5	Convolution w\RELU activation	Kernel: (3,3) Stride: (1,1,1,1) Padding: Same Input Feature Map: 32 Output Feature Map: 64
6	Max-Pooling Layer	Kernel: (1,2,2,1) Stride: (1,2,2,1) Padding: Same
7	Flatten Layer	
8	Full-Connected Layer w\RELU activation	Input Feature Map: 1024 Output Feature Map: 1024
9	Dropout Layer	Keep Probability: 0.50
10	Full-Connected Layer w\RELU activation	Input Feature Map: 1024 Output Feature Map: 512
11	Dropout Layer	Keep Probability: 0.50
12	Full-Connected Layer w\RELU activation	Input Feature Map: 512 Output Feature Map: 256
13	Full-Connected Output Layer w\Softmax	Input Feature Map: 256 Output Classes: 43

Table 4: Tuned Hyper-parameters

Hyperparameter	**Detail**
Epochs	75
Batch Size	64
Learning Rate	7.5e-4 or 1e-3
L2 Regularization Penalty	1e-6

Once adequate performances were obtained from tuning learning rates and batch sizes, further optimization wasn't performed.

3 Results

As seen in Figure 4, the training loss plateaus at around 99 % accuracy, and validation (as well as test) accuracy reaches around 95-96 % accuracy. The stopping epoch right now is random, but methodologies can be implemented to stop when training accuracy is high and loss is minimal after a patience of a few epochs (Early Stopping).

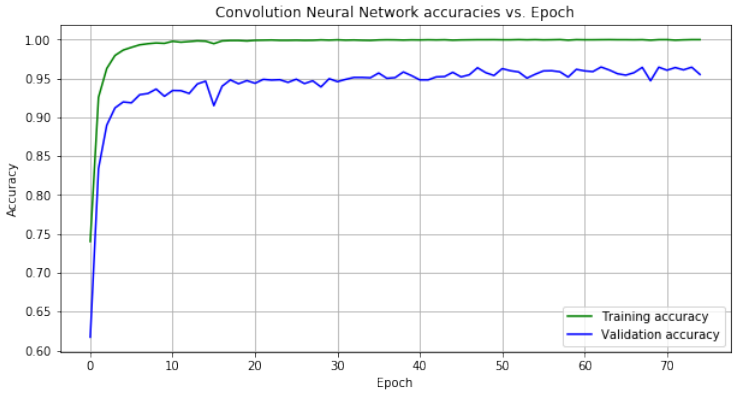


Figure 4: Training-Validation Accuracy vs. Epoch

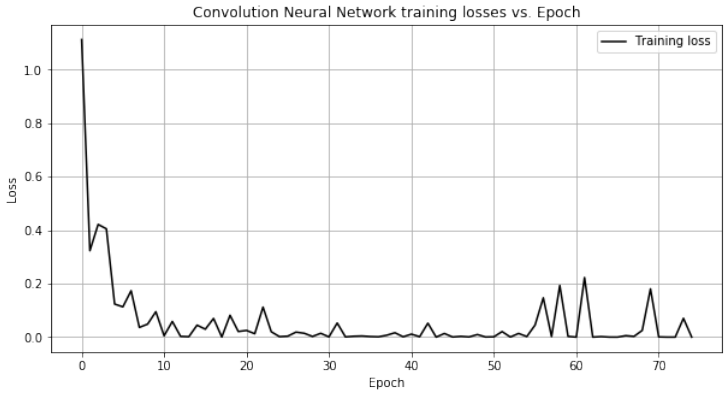


Figure 5: Training Losses vs. Epoch

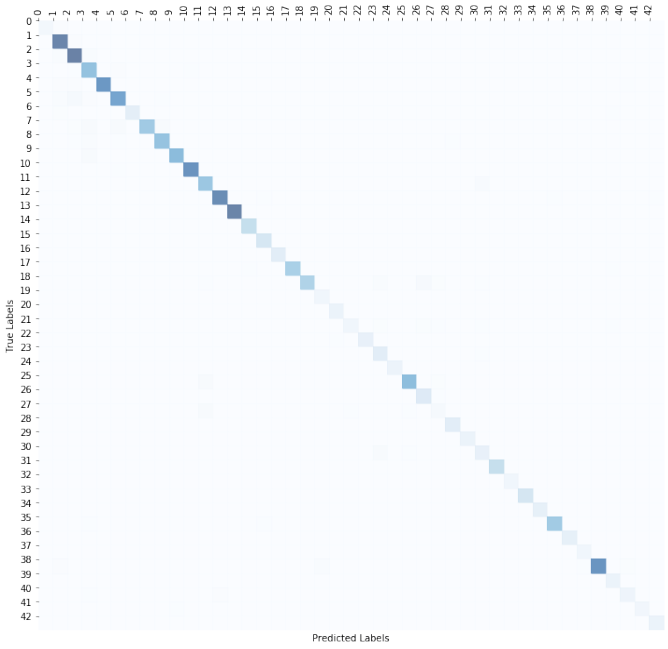


Figure 6: Confusion Matrix

An in-depth analysis of the Confusion Matrix (Figure 6) can be seen in the attached notebooks/ html files. The confusion matrix evaluates the accuracy of a particular classification where $C_{i,j}$ is equal to the number of observations in class i, but are predicted to be in class j. Ideally, the Confusion matrix heat map would show a high intensity on the main diagonal corresponding to correctly predicted observations. Some classes are seen to be easily misclassified (lower accuracy obtained from Pandas confusion matrix), but could be improved by further data augmentation or creating a deeper network.

The CNN was also tested out on images found on the net, as well as from the Belgian traffic sign dataset with varying successes over different iterations. The main issue seen is identifying images that haven't been trained on (leading to misclassification), and also to identify signs of different shapes/colors but have the same class index as seen in the Belgian dataset. There are two main notebooks attached with different learning rates- a learning rate of 1e-3 performs slightly better on the validation and test sets as well as on the new images (has very large belief in the correct class, or has a probability of 1.0 for the correct class).

In addition to this, the activations of the layers are shown in Figure 7 which reveal how the CNN chooses which identifying characteristics it deems important for classification during feature extraction.

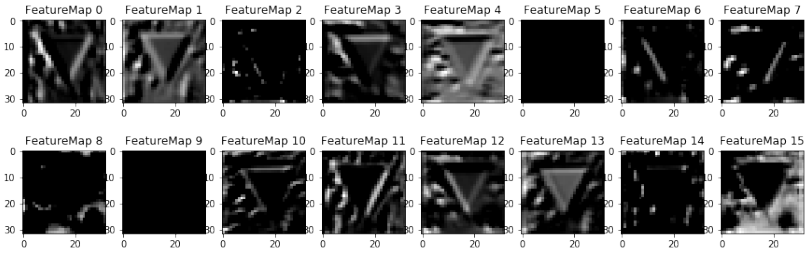


Figure 7: Feature Map of an activation layer

4 Conclusion & Further Improvements

As can be seen from the Results section, the CNN successfully classifies most images, including images that haven't been seen before. The performance could be better improved by making the network deeper, using the VGG-16 and Inception architectures with pre-trained weights or with adequate time, running for more EPOCHs on a GPU, and fine-tuning chosen hyper-parameters.Furthermore, using Keras or a higher-level deep learning software will make it easier for building the model, for implementing monitors like Early Stopping, and mainly for tuning hyper-parameters much more easily. Scikit-Learn Cross-Validation's GridSearch or RandomSearch was looked into (but later abandoned) for choosing optimum hyper-parameters.