Traffic Sign Classifier

by Masabumi Chano

CarND Term1 – P2
3/26/2017

1. Write-up

1.1 Submission Files

- The Traffic_Sign_Classifier.ipynb notebook file with all questions answered and all code cells executed and displaying output.
- An HTML or PDF export of the project notebook with the name report.html or report.pdf.
- Any additional datasets or images used for the project that are not from the German Traffic Sign Dataset. Please do not include the project data set provided in the traffic-sign-data.zip file.
- Your writeup report as a markdown or pdf file

2. Dataset Summary and Exploration

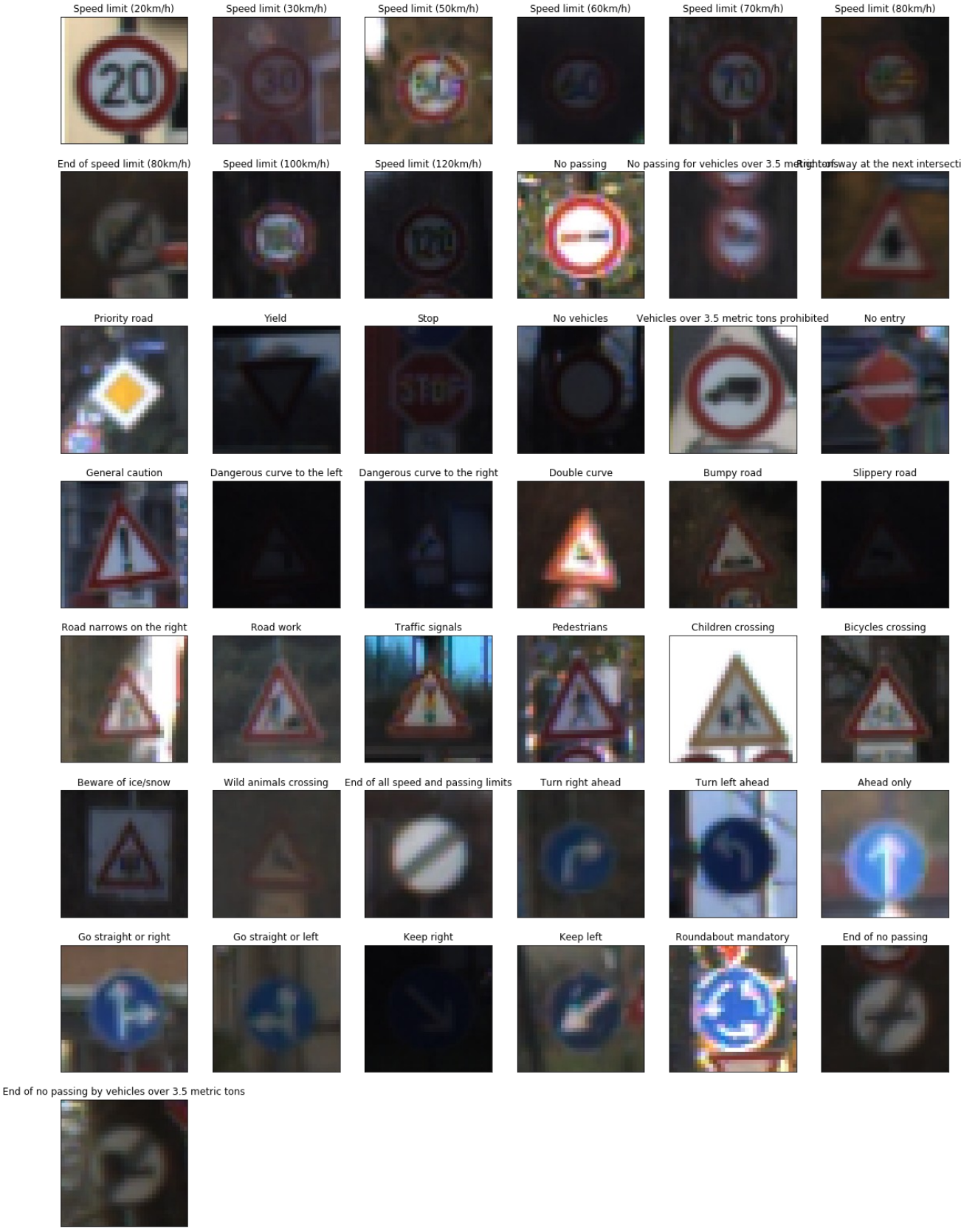
2.1 Dataset Summary

A summary of the original dataset is provided below. The calculation used to obtain this summary can be observed in Cell 3 of the Python notebook. The function `len` was used to calculate number of images in the dataset and `numpy`, in particular `numpy.unique()`, was used to determine the number of classes in the dataset.

Number of Training Examples	34799
Number of Validation Examples	4410
Number of Testing Examples	12630
Image Data Shape	(32, 32, 3)
Number of Classes	43

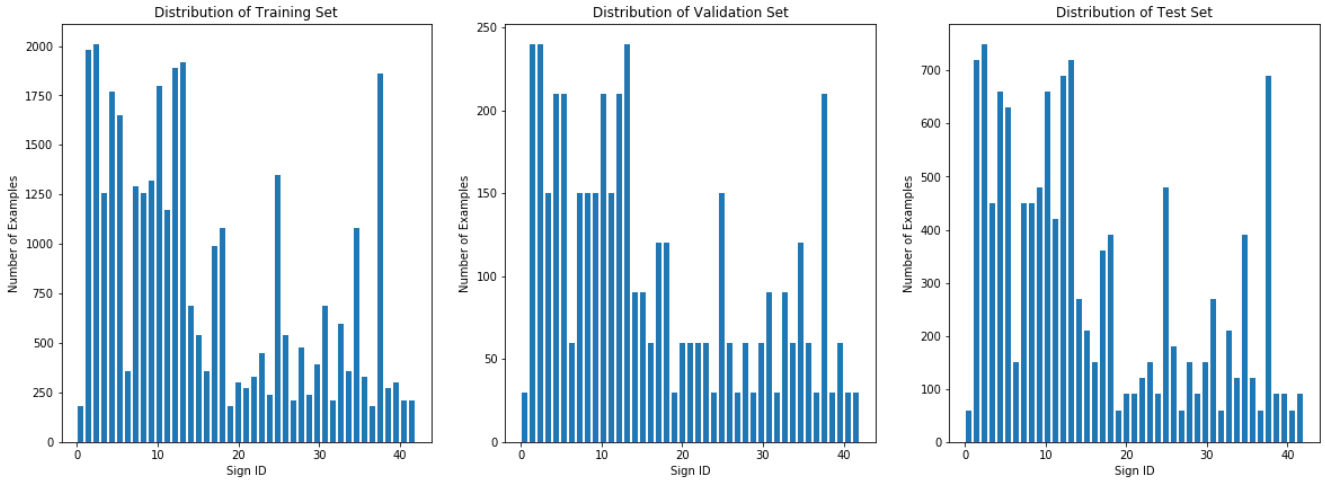
2.2 Exploratory Visualization

First, I printed out one image per class to visualize the dataset. This is performed in Cell 4. The class or the name of the traffic sign, was used as the title of the image plot.



Sample images (1 per class)

Next, I plotted histograms to show the distribution of data within the training, validation, and test dataset. This can be observed in Cell 5. For convenience, it is also plotted below. At first glance, it is evident not all classes are well represented and will pose issues in training.



3. Design and Test a Model Architecture

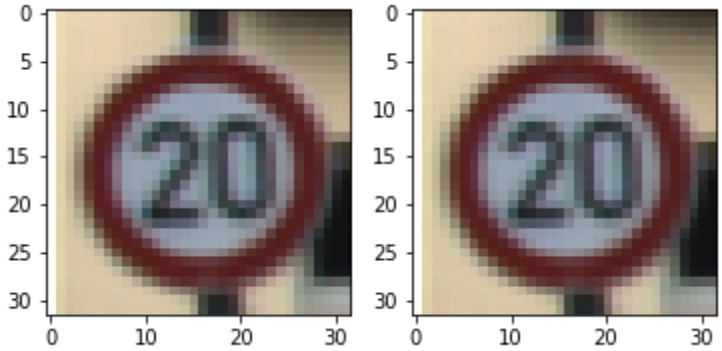
3.1 Preprocessing

In order to aid in feature extraction, preprocessing was performed. A simple global normalization was performed using the following equation.

$X_{normalized}$ `= (X - np.min(X)) / (np.max(X) - np.min(X))`

The dataset is in 3 channel RGB represented by 8-bits per pixel per channel. Therefore, the maximum value per pixel per channel is less than or equal to 255 or 2^8-1, and the minimum value is greater than equal to 0. The data is now normalized to values between [0,1] rather than [0,255].

Because Matplotlib automatically scales plotted images, the before and after preprocessing images look identical. However, if we examine the max and min values, we can see that they are normalized properly. (Note: individual images were not scaled during the preprocessing to fill the full range)



`Left: Before with (min,max) of (16,254)`

```
Right: After with (min,max) of (0.063,0.996)
```

During testing, grayscale was also used as a preprocessing technique. However, unlike the findings in the Sermanet paper, performance was worse with grayscaling than without. Therefore, it is not used.

3.2 Data Augmentation

As explained in 2.2, not all classes are well represented in the dataset. To overcome this, we augment data by creating new data based on existing data. Three operations are performed to obtain new augmented data.

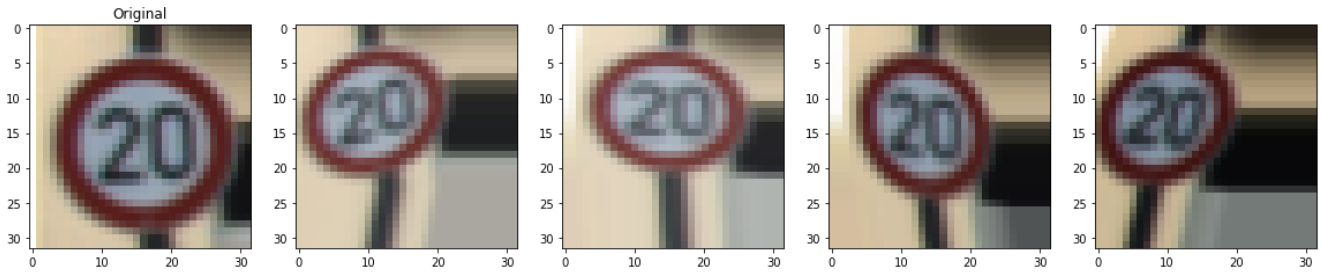
The Skimage library was used to perform these operations. First, rotate is applied to rotate an existing image by a random angle between -15 and 15 degrees.

Next adjust_gamma is used to change the gamma or exposure level. This makes the images visually look darker (lower gamme) or brighter (high gamma). Gamma was kept in a modest range so the images were not too dark or bright based on trial and error.

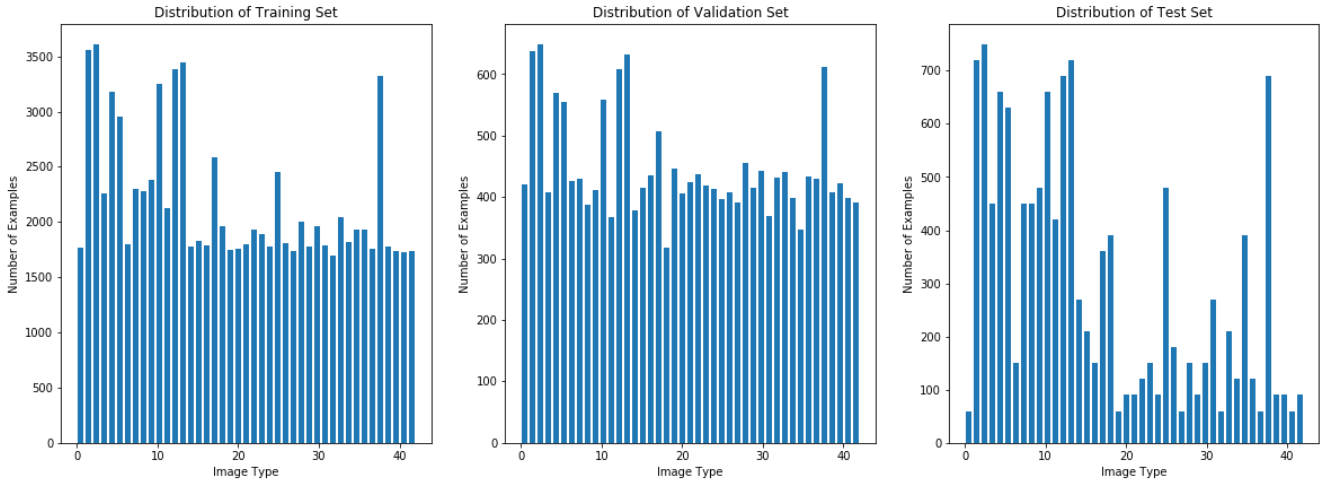
Finally, warp is applied to skew the image. I used the AffineTransform to scale, shear, and translate the image all at once.

When applicable, mode is set to ‘edge’ to keep the edges consistent with the image. The following are a few example of augmented images from the same original image. The left most image is the original image.

The actual steps can be found under, “Pre-process the Data Set” section of the notebook.



In the end a total of 74,217 augmented images were created. The newly created images were then distributed to the training dataset and the validation dataset. 80% of the new images were added to the training dataset. The remaining 20% was added to the validation dataset. The test dataset remains unchanged. The resulting histogram of the datasets can be seen below. All classes are now well-represented in the training and validation datasets.



3.3 Model Architecture

The model used for this projects consists of the following layers:

Layer	Description
Input	32x32x3 RGB image
Conv3x3	1x1 stride, valid padding, outputs 30x30x32
RELU	
MaxPool1	2x2 filter, 2x2 stride, outputs 15x15x32
Conv3x3	1x1 stride, valid padding, outputs 13x13x64
RELU	
MaxPool2	3x3 filter, 2x2 stride, outputs 6x6x64
CONCAT	Flatten and Concatenate MaxPool1 and MaxPool2
FC	outputs 1x1x1024
RELU	
DROP	Keep_Prob = 0.5
FC	outputs 1x1x512
RELU	
Logits/FC	outputs 1x1x43

```
|:------------:|:---------------------------------------------:|
```

The project model takes inspiration from Pierre Sermanet and Yann LeCun's 2-strage ConvNet architecture described in IJCNN 2011 (Traffic Sign Recognition with MultiScale Convolutional Networks). I expanded on it by adding a second Fully Connected layer and adding dropout between the first and second Fully Connected Layer. The model was arrived at through numerous trial-and-error. Some of the design choices are highlighted later in this section.

3.4 Describe Model Training

The following hyper parameters were used to train the network.

```
Rate: 0.001, Epochs: 20, Batch: 128, Dropout: 0.5
```

For the optimizer, the AdamOptimizer was used for its relatively straight forward parameter tuning. This optimizer also takes into account some of the advantages of other methods such as the AdaGrad or RMSProp. Additionally, per the AdamOptimizer paper by Kingma and Ba, (https://arxiv.org/pdf/1412.6980.pdf), experimentally it was found to outperform other methods in multi-layer neural networks.

3.5 Describe approach taken to find the solution

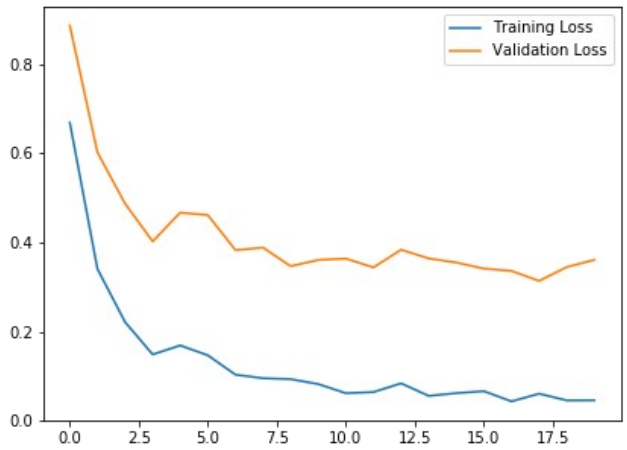
The approach was highly iterative. Using the IJCNN 2011 model as a start, I mostly experimented with filter sizes, number of filters, and size of the fully connected layer. I started with big filter sizes of 5x5 in the first convolution layer based on past use in the LeNet model. However, I felt the filter was too big and not capturing features as well as when using a smaller filter. Furthermore, it reduced the image space a bit too early in the process. In the end, I felt 3x3 was a good start.

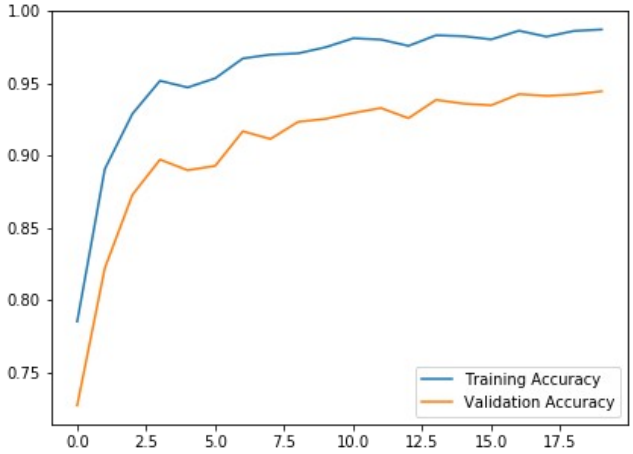
Similarly, I used a smaller number of filters in the beginning to keep computation speed bearable. For example, just 4 in the first convolution layer and 8 in the second convolution layer. However, accuracy was not very high, again probably due to not being able to capture features with limited filters, so I gradually increased the number of filters in the early layers until I reached 32 in the first layer and 64 in the second. Increasing the number of filters enables the network to fine tune each filter to different features (and not have to "share" filters for features).

Finally, I added a dropout layer to prevent overfitting of the training data. The random dropouts "force" the network to learn multiple independent interpretations of the data. This helps generalize the network. In my testing, I didn't find drastic differences between a keep_probability of 0.5 to 0.75. I believe having a modest dropout was sufficient to provide good overfitting protection for this particular model.

In the end, my model had the following settings and results.

```
Rate: 0.001, Epochs: 20, Batch: 128, Dropout: 0.5
```






```
Epoch, Tloss, Tacc, Vloss, Vacc

1, 0.669, 0.785, 0.887, 0.727
2, 0.341, 0.891, 0.602, 0.822
3, 0.221, 0.929, 0.487, 0.873
4, 0.149, 0.952, 0.402, 0.897
5, 0.169, 0.947, 0.467, 0.890
6, 0.147, 0.953, 0.461, 0.893
7, 0.103, 0.967, 0.383, 0.917
8, 0.095, 0.970, 0.388, 0.911
9, 0.093, 0.971, 0.346, 0.923
10, 0.082, 0.975, 0.361, 0.925
11, 0.062, 0.981, 0.363, 0.929
12, 0.064, 0.980, 0.344, 0.933
13, 0.084, 0.976, 0.383, 0.926
14, 0.056, 0.983, 0.364, 0.938
15, 0.062, 0.982, 0.355, 0.936
16, 0.066, 0.980, 0.341, 0.935
17, 0.043, 0.986, 0.336, 0.942
18, 0.060, 0.982, 0.313, 0.941
19, 0.045, 0.986, 0.345, 0.942
20, 0.046, 0.987, 0.361, 0.944

Test Accuracy = 0.901
```

Test Accuracy was 90.1% and certainly has room for improvement.

Some other test scenarios, after having narrowed down the model, are noted in the following table. These show some of the fine tuning performed after the general range of hyper parameters were determined through trial and error.

FC2 Size	Rate	Epochs	Batch	Dropout	Final VAcc	Test Acc
512	0.001	20	128	0.5	0.944	0.901
1024	0.0003	25	256	0.5	0.929	0.859
1024	0.001	25	256	0.5	0.937	0.903
1024	0.002	25	256	0.5	0.933	0.903

1024	0.001	10	128	0.5	0.925	0.886
1024	0.001	20	128	0.5	0.948	0.912
1024	0.002	20	128	0.5	0.926	0.892
1024	0.001	40	128	0.5	0.950	0.910
1024	0.0008	40	128	0.75	0.952	0.909

Here I'd like to point out that after observing the size of FC2 or the second fully-connected layer didn't make a big difference when moving down from 1024 to 512, I opted to use 512.

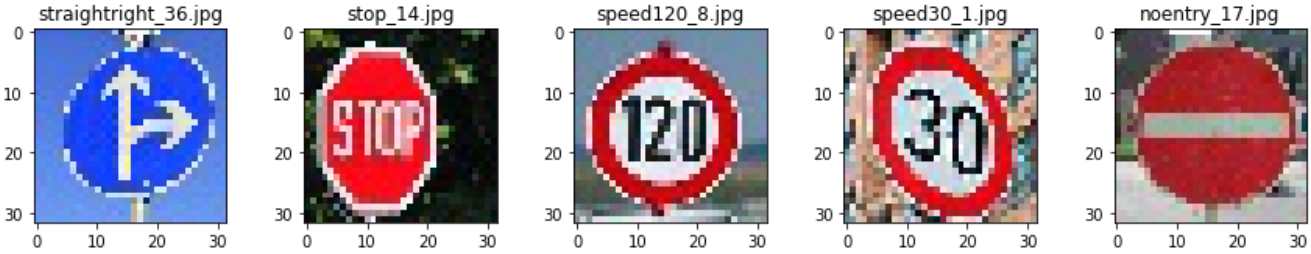
A significant limiting factor in performing more tests for this project has been the GPU instance on AWS (the smaller GPU instance). Running a jupyter notebook on AWS frequently timed when performing long training. Furthermore, the slightest increase in data or increase in complexity of the tensorflow model quickly led to epochs taking more than 5min per epoch. I was forced to spend hours training a model (if I could avoid the timeout by periodically running an empty cell in another jupyter notebook).

I hope to revisit new architectures later in the course after I purchase a GPU.

4. Test Model on New Images

4.1 Choose 5 Images

The following 5 images were downloaded from the web.



The first image might be difficult to classify because the photo was taken at an angle. The sky and the sign are also similar in color (to the human eye, but perhaps not to the computer).

The second image might be difficult to classify because it is skewed heavily possibly due to the angle the photo was taken in.

The third image hopefully is a straight forward one. But it may be difficult because the zero in 120 is not quite clear.

The fourth image might be difficult to classify also because it looks quite skewed and taken at an odd angle.

The last image hopefully is relatively straight forward.

4.2 Model Prediction

When the images were passed to our model, the following was calculated.

```
Web Accuracy = 0.800

Image0 is 36, predicted 35 @ 0.999 probability
Image1 is 14, predicted 14 @ 1.000 probability
Image2 is 8, predicted 8 @ 1.000 probability
Image3 is 1, predicted 1 @ 0.859 probability
Image4 is 17, predicted 17 @ 1.000 probability
```

The model correctly predicted 4/5 images or 80% accuracy. It unfortunately incorrectly predicted the first image as “Ahead Only” when it was actually “Go straight or right”. The model had high confidence in its prediction at 99.9% probability estimate. It should be noted, “Go straight or right” (the correct answer) had the second highest probability (although a very very small number). While the accuracy wasn’t as good as the test set accuracy, the sample size here was very small and therefore not a good indicator of performance.

4.3 Softmax

Here are the results summarized with the top 5 softmax probabilities. Despite an accuracy of 80%, the network was extremely confident in its predictions, almost reaching 1 in all cases. I’m not quite sure why this is the case and hope to investigate. The softmax probabilities are arranged on the y-axis from the 5th highest probability at the top and the highest probability at the bottom.


36
Turn left ahead:34
Road narrows on the right:24
Turn right ahead:33
Go straight or right:36
Ahead only:35
14
STOP
Speed limit (60km/h):3
Speed limit (50km/h):2
Speed limit (30km/h):1
Speed limit (20km/h):0
Stop:14
8
120
Speed limit (80km/h):5
Speed limit (20km/h):0
Speed limit (30km/h):1
Speed limit (70km/h):4
Speed limit (120km/h):8
1
30
No passing:9
Speed limit (120km/h):8
Speed limit (20km/h):0
Speed limit (70km/h):4
Speed limit (30km/h):1
17
Speed limit (60km/h):3
Speed limit (50km/h):2
Speed limit (30km/h):1
Speed limit (20km/h):0
No entry:17