Traffic Sign Recognition Report

Student: Saad Awan

Section 1. Data Set Summary & Exploration

1. Provide a basic summary of the data set. In the code, the analysis should be done using python, numpy and/or pandas methods rather than hardcoding results manually.

I used the basic python functionality to obtain the sizes the input datasets. Then I used the panda library to find the number of unique number of classes (Signs)

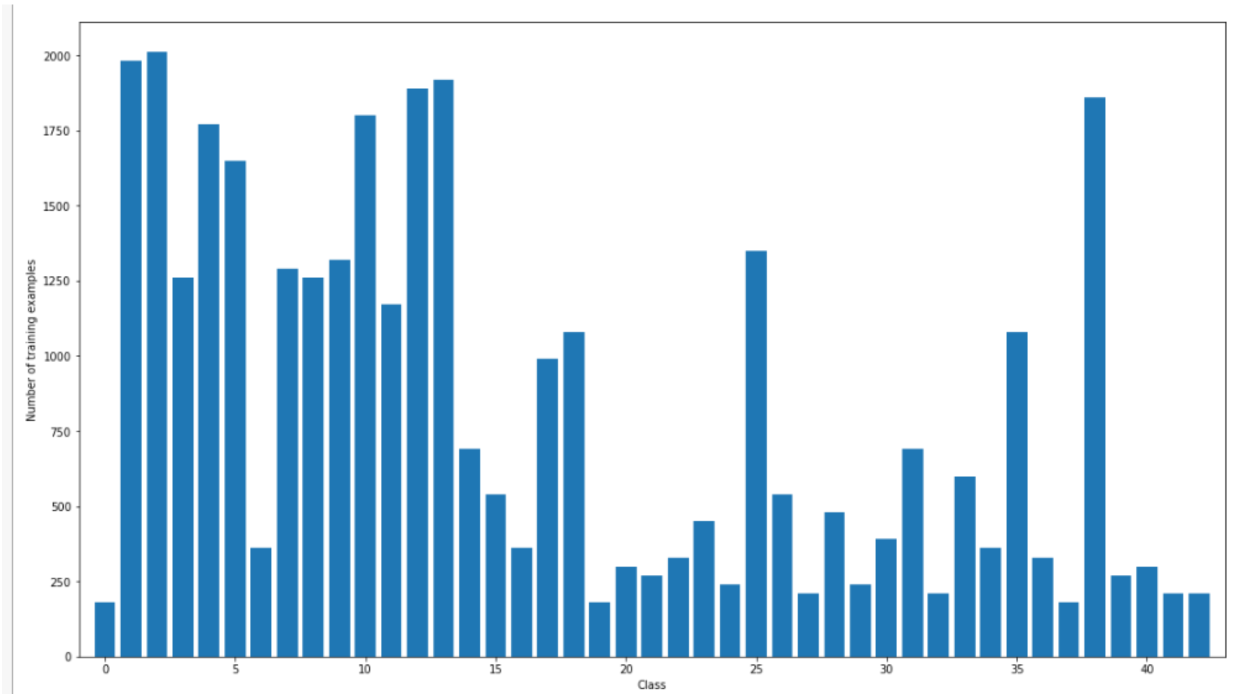
```
for indexer in range(0,label_signnames_csv.as_matrix().shape[0]):

    y_train_panda['Name'] =
np.where(y_train_panda['Sign']==(label_signnames_csv.as_matrix()[indexer,0]),label_signnames_csv.as_matrix()[indexer,1],y_t
rain_panda['Name'])

y_train_panda.insert(0, 'TrainIndex', range(0,len(y_train_panda)))

#signnames = label_signnames_csv.values[:,1] ## Just use the labels

# TODO: Number of training examples

n_train = y_train.shape[0]

# TODO: Number of validation examples

n_validation = y_valid.shape[0]

# TODO: Number of testing examples.

n_test = y_test.shape[0]

# TODO: What's the shape of an traffic sign image?

image_shape = X_train[0].shape

# TODO: How many unique classes/labels there are in the dataset.

n_classes = y_train_panda.groupby('Sign').count().shape[0]

#print(y_train_panda.groupby(['Sign','Name']).sum())


print("Number of training examples =", n_train) #34799

print("Number of testing examples =", n_test) #12630

print("Image data shape =", image_shape) #(32, 32, 3)

print("Number of classes =", n_classes) #43
```

Section 2. Include an exploratory visualization of the dataset.

I also explored the dataset by providing examples of each sign in the dataset (5 per) and the number of each sign in the dataset. Please refer to the HTML preview of the Jupyter Notebook, as the images could not be copied into this report.

Traffic_Sign_Classifi
er.html

I utilized the panda dataframe created in the first section to obtain a count of each sign in the dataset. This is seen in the barchart below.



We can see that the dataset was heavily imbalanced with some images overly represented in the dataset and some image with less than 200 samples. This will create a problem for the recognizer as it will not be sufficiently trained to recognize these images.

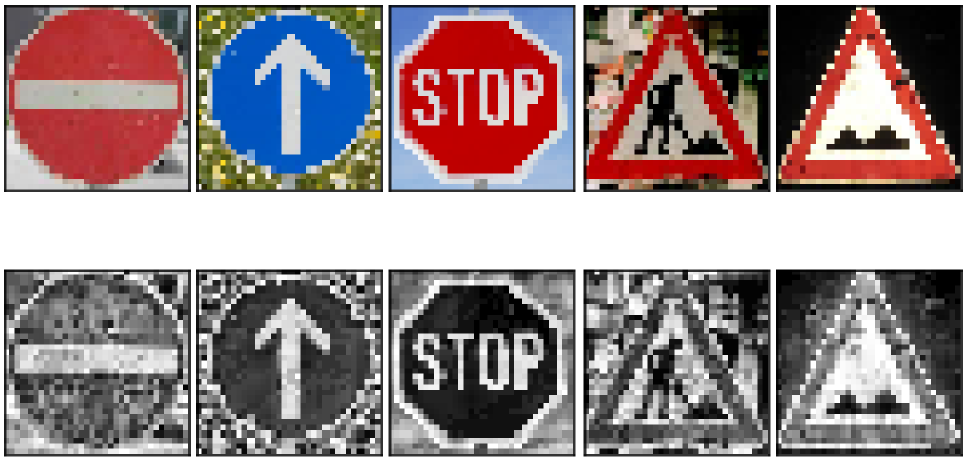
In addition, while visualizing the dataset, it was seen that several images (in the sample) were overexposed or had high contrast. These should be resolved in the preprocessing phase.

Section 3: Design and Test a Model Architecture

1. Describe how you preprocessed the image data. What techniques were chosen and why did you choose these techniques? Consider including images showing the output of each preprocessing technique. Pre-processing refers to techniques such as converting to grayscale, normalization, etc.

As a first step, I decided to convert the images to grayscale because as was discussed in the paper "Traffic Sign Recognition with Multi-Scale Convolutional Networks by Pierre Sermanet and Yann LeCun." Their exploration of a traffic sign recognizer indicated that with the usage of a grayscale input they had seen an increase in the effectiveness of the classifier. The grayscale image allowed the image to be normalized using the CLAHE histogram function available in the skimage library. This function equalized the contrast in the image and balance out exposure.

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



In addition, we also shuffle the dataset to ensure that the system does not rely on the input order (shuffle was also used at runtime).

I had attempted to utilize the skimage library to distort/skew/flip the images in order to expand the dataset was unable to do so before project deadline.

2. Describe what your final model architecture looks like including model type, layers, layer sizes, connectivity, etc.) Consider including a diagram and/or table describing the final model.

My final model consisted of the following layers:

Layer	Description
Input	32x32x1 Grayscale image
First Stage: Convolution 5x5	1x1 stride, same padding, outputs 32x32x50
First Stage: RELU	Activation Layer
First Stage: Max pooling	2x2 stride, outputs 16x16x50 [Dropout – 40%]
Second Stage: Convolution 5x5	1x1 stride, same padding, outputs 16x16x100
Second Stage: RELU	Activation Layer
Second Stage: Max pooling	2x2 stride, outputs 8x8x100 [Dropout – 40%]
Third Stage: Convolution 5x5	1x1 stride, same padding, outputs 8x8x150
Third Stage: RELU	Activation Layer
Combination Max Pooling Layer –First Stage	1x1 sample, 2x2 stride, outputs 8x8x50
Flatten Layer	Combine Outputs of First/Second/Third Layer
Fully Connected Layer 1 & Relu Activation	Input : 19200, Output: 1024
Fully Connected Layer 2 & Relu Activation	Input : 1024, Output: 300
Fully Connected Layer 3 & Relu Activation	Input : 300, Output: 43
Softmax	Output

3. Describe how you trained your model. The discussion can include the type of optimizer, the batch size, number of epochs and any hyperparameters such as learning rate.

To train the model, I used the AdamOptimizer similar to the one used in the LeNet lab. This was used to blackbox the forward / backward propagation using optimized strategies. The hyper parameters were determined using trial and error, baselined from the observations made from the LeNet lab.

The final parameters are shown below:

Batch size = 256
Number of Epoch = 16
Dropout per Layer = 60%
Learning rate = 0.004

4. Describe the approach taken for finding a solution and getting the validation set accuracy to be at least 0.93. Include in the discussion the results on the training, validation and test sets and where in the code these were calculated. Your approach may have been an iterative process, in which case, outline the steps you took to get to the final solution and why you chose those steps. Perhaps your solution involved an already well known implementation or architecture. In this case, discuss why you think the architecture is suitable for the current problem.

Due to the intensive model architecture and the pre-processed grayscaled/normalized input, the system was able to pass the Validation/Test checks in the first pass. However, the model was unable to properly categorize the web captured images and was also outputting low confidence values for all inputs. This triggered the manipulation of the model architecture to adjust the model to have a higher confidence *threshold. I had initially tried to use a 4x4 input to the first fully connected layer, but later removed the pooling of the last layer to provide a larger visualization for the fully connected layers.*

My final model results were:

Training set accuracy of 95.4%

Validation set accuracy of 95.4%

Test set accuracy of 94.8%

If an iterative approach was chosen:

What was the first architecture that was tried and why was it chosen?

The first approach chosen was as below:

Layer	Description
Input	32x32x1 Grayscale image
First Stage: Convolution 5x5	1x1 stride, same padding, outputs 32x32x50
First Stage: RELU	Activation Layer
First Stage: Max pooling	2x2 stride, outputs 16x16x16 [Dropout – 20%]
Second Stage: Convolution 5x5	1x1 stride, same padding, outputs 16x16x100
Second Stage: RELU	Activation Layer
Second Stage: Max pooling	2x2 stride, outputs 8x8x32 [Dropout – 40%]
Third Stage: Convolution 5x5	1x1 stride, same padding, outputs 8x8x64
Third Stage: RELU	Activation Layer
Third Stage: Max pooling	2x2 stride, outputs 4x4x64 [Dropout – 40%]
Combination Max Pooling Layer –First Stage	9x9 stride, outputs 4x4x16
Combination Max Pooling Layer –Second Stage	1x1 stride, outputs 4x4x32
Flatten Layer	Combine Outputs of First/Second/Third Layer
Fully Connected Layer 1 & Relu Activation	Input : 1792, Output: 860
Fully Connected Layer 2 & Relu Activation	Input : 860, Output: 320
Fully Connected Layer 3 & Relu Activation	Input : 320, Output: 43
Softmax – Output	

* What were some problems with the initial architecture?

As previously stated, the model was unable to properly categorize the web captured images and was also outputting low confidence values for all inputs. This triggered the manipulation of the model architecture to adjust the model to have a higher confidence threshold.

* How was the architecture adjusted and why was it adjusted? Typical adjustments could include choosing a different model architecture, adding or taking away layers (pooling, dropout, convolution, etc), using an activation function or changing the activation function. One common justification for adjusting an architecture would be due to overfitting or underfitting. A high accuracy on the training set but low accuracy on the validation set indicates over fitting; a low accuracy on both sets indicates under fitting.

I had initially tried to use a 4x4 input to the first fully connected layer and scaled down the outputs of the first and second stage to a 4x4 as well. I removed the pooling of the last layer in the final model to provide a larger visualization for the fully connected layers. By providing a larger 8x8, I believe the model was able to better understand the input images.

* Which parameters were tuned? How were they adjusted and why?

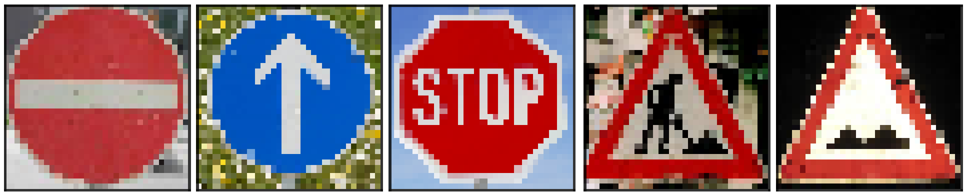
The architecture was able to meet the design requirements from the initial model. The learning rate was increased by a small margin (0.0001 to 0.0004) but did not impact the outcome much.

* What are some of the important design choices and why were they chosen? For example, why might a convolution layer work well with this problem? How might a dropout layer help with creating a successful model?

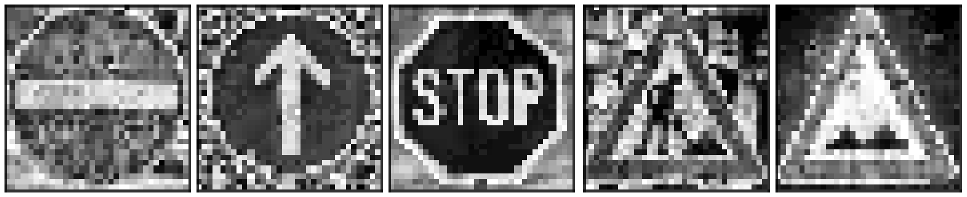
Section 3: Test a Model on New Images

1. Choose five German traffic signs found on the web and provide them in the report. For each image, discuss what quality or qualities might be difficult to classify.

Here are five German traffic signs that I found on the web:



They were preprocessed to grayscale and normalized to the images below

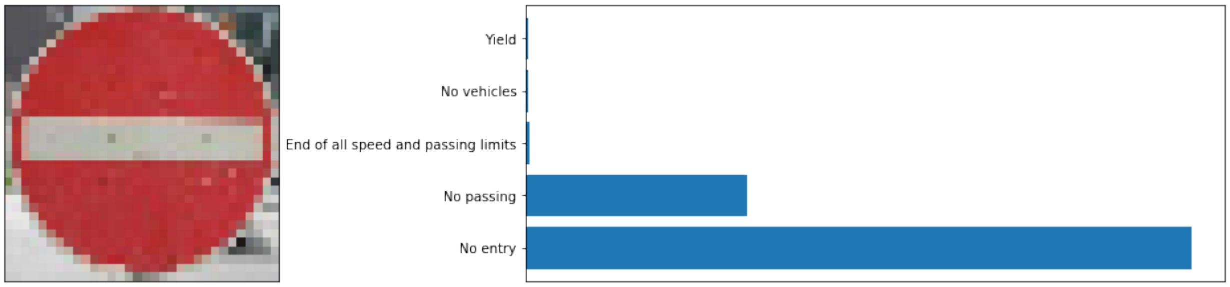


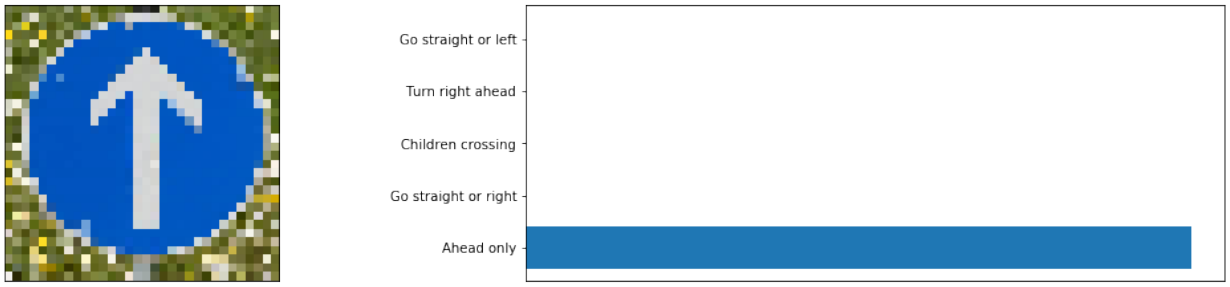
:

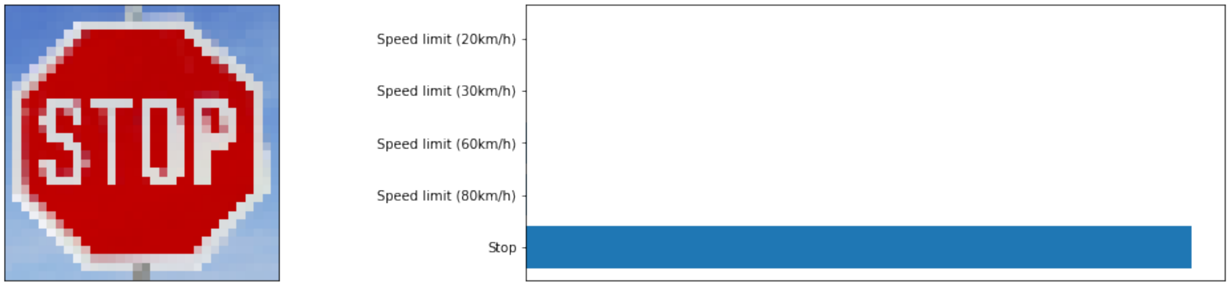
The model should easily be able to classify these images as most of them are clear without noise. The roadwork sign has a lot of background objects so it would be interesting see how the NN handles this input.

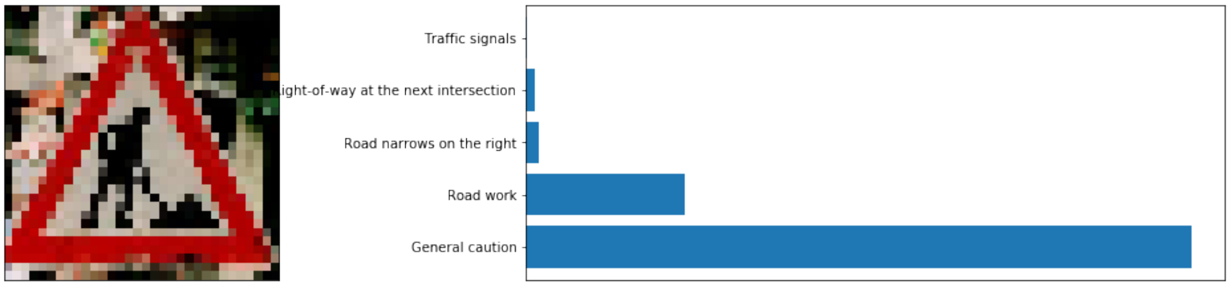
2. Discuss the model's predictions on these new traffic signs and compare the results to predicting on the test set. At a minimum, discuss what the predictions were, the accuracy on these new predictions, and compare the accuracy to the accuracy on the test set

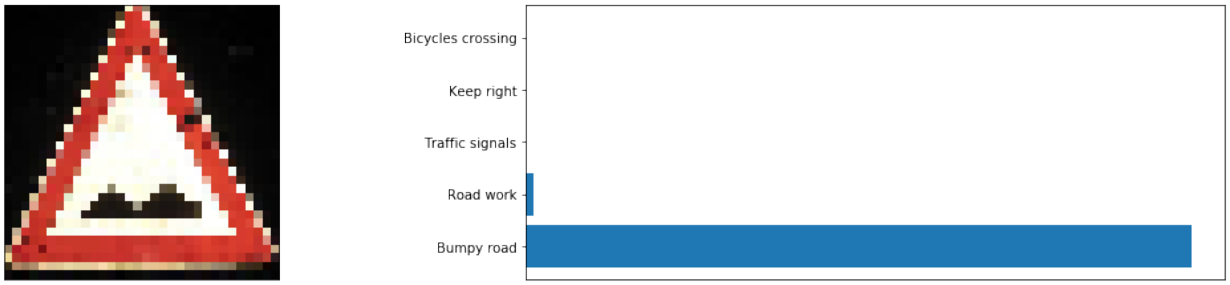
Here are the results of the prediction:











The model was able to correctly guess 4 of the 5 traffic signs, which gives an accuracy of 80%. This compares favorably to the accuracy on the test set of 94.8%. While the system did classify most of the test set correctly, the exploration of the softmax probabilities bring insight in the confidence of the neural net. When this was ran initially, the neural net was unable to classify these images correctly or had less than 10% confidence scores on the images.

As previously mentioned, these signs should have been easy to classify for the web images. The system was had only a 70% confidence on the no-entry symbol, but had a 25% confidence that it could also be a No passing sign. These signs look similar after a normalized grayscale, so this would be understandable.

However, it seems that the road work image proved difficult for the net to classify due to the extra noise in the system. The neural net had a high confidence that the roadwork image was a general crossing sign instead of road work which was rated second. I believe with a successful implementation of the skew/flip algorithm for preprocessing the system would have been more robust against the noise.

The code for making predictions on my final model is below:

```
with tf.Session() as sess:
    saver.restore(sess, './lenet')
    p = sess.run(tf.nn.softmax(logits), feed_dict = {x: X_new_pre,keep_prob_1 :1,keep_prob_2 : 1,keep_prob_3 : 1})
    top_5 = sess.run(tf.nn.top_k(p,k=5))
```

Final Output Values from the Softmax calculation

```
TopKV2(values=array([
       [0.738, 0.245, 0.004, 0.003, 0.002],
       [1.000, 0.000, 0.000, 0.000, 0.000],
       [0.997, 0.002, 0.001, 0.000, 0.000],
       [0.783, 0.187, 0.014, 0.010, 0.002],
       [0.988, 0.012, 0.000, 0.000, 0.000]], dtype=float32), indices=array
([     [17,  9, 32, 15, 13],
       [35, 36, 28, 33, 37],
       [14,  5,  3,  1,  0],
       [18, 25, 24, 11, 26],
       [22, 25, 26, 38, 29]]))
```