Traffic Sign Recognition

Writeup Template

You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

Build a Traffic Sign Recognition Project

The goals / steps of this project are the following:

- Load the data set (see below for links to the project data set)
- Explore, summarize and visualize the data set
- Design, train and test a model architecture
- Use the model to make predictions on new images
- Analyze the softmax probabilities of the new images
- Summarize the results with a written report

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Writeup / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. You can use this template as a guide for writing the report. The submission includes the project code.

You're reading it! and here is a link to my project code

Data Set Summary & Exploration

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

The code for this step is contained in the second code cell of the IPython notebook.

I used the Numpy library to calculate summary statistics of the traffic signs data set:

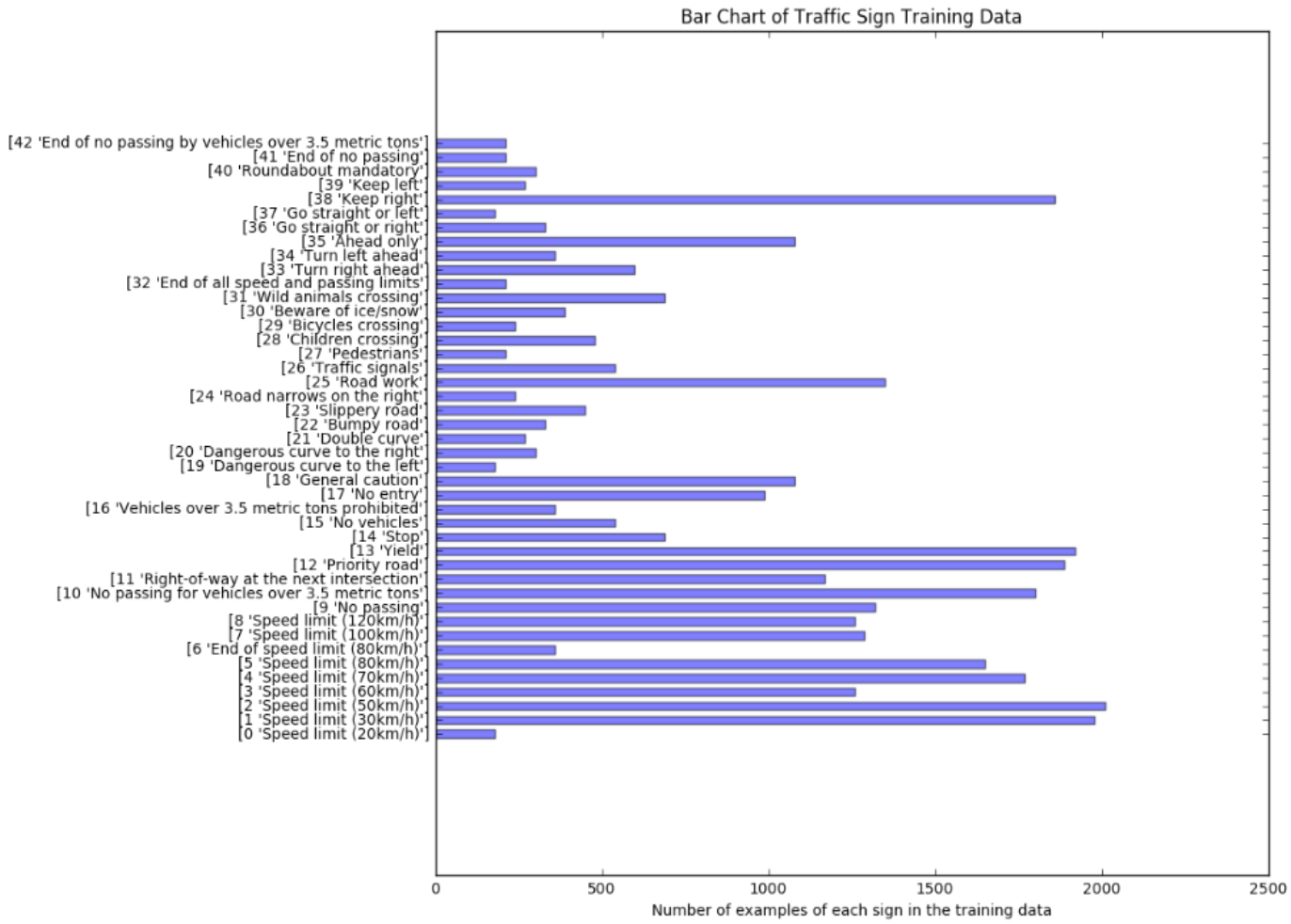
- The size of training set is 34,799 examples
- The size of test set is 12,630 examples
- The shape of a traffic sign image is (32, 32, 3)
- The number of unique classes/labels in the data set is 43

2. Include an exploratory visualization of the dataset and identify where the code is in your code file.

The code for this step is contained in the third code cell of the IPython notebook.

I created a horizontal barchart showing the count for each type of sign in the training data set.

Here is an exploratory visualization of the data set. It is a bar chart showing the data



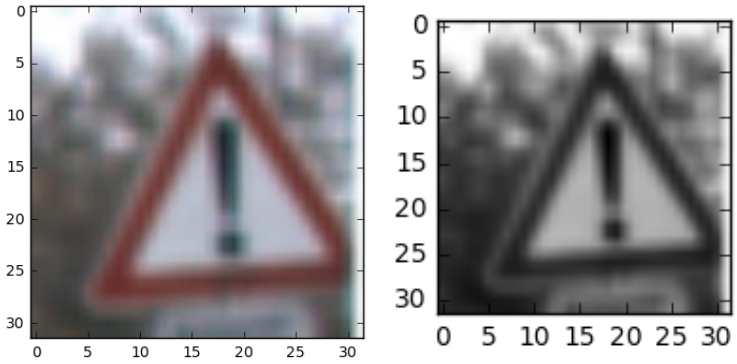
Design and Test a Model Architecture

1. Describe how, and identify where in your code, 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.

The code for this step is contained in the fifth code cell of the IPython notebook.

As a first step, I decided to convert the images to grayscale based on the information provided in the paper “Traffic Sign Recognition with Multi-Scale Convolutional Networks”. Models were tested with both greyscale and RGB images. The grayscale images seemed to preform slightly better but only by 1 to 2 %.

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

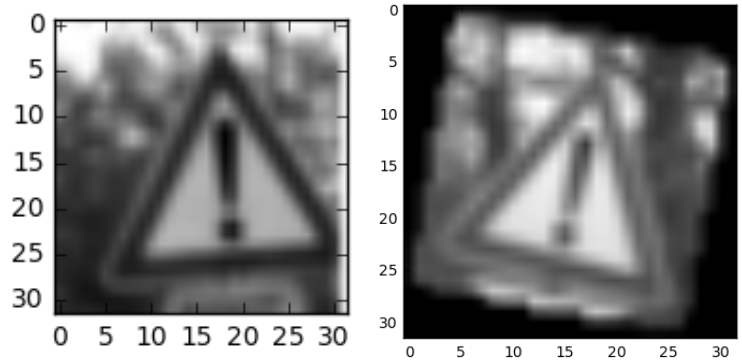


As a last step, I normalized the image data between 0.1 and 0.9. Normalizing the data showed a 3% improvement over nonnormalized data.

2. Describe how, and identify where in your code, you set up training, validation and testing data. How much data was in each set? Explain what techniques were used to split the data into these sets. (OPTIONAL: As described in the "Stand Out Suggestions" part of the rubric, if you generated additional data for training, describe why you decided to generate additional data, how you generated the data, identify where in your code, and provide example images of the additional data)

The pickled data appeared to already be split into a training set, a validation set and a testing set. The training set had 34,799 images. The validation set had 12,630 images and the test set had 4,410 images. I looked into taking the training set and taking each image and randomly rotating the image (-15 to 15) degrees and adding this data to the original training data set. After completing this process the network seemed to preform worse than when no manipulation to the training data was performed. The decision was made to not include these images in the final project.

Here is an example of an original image and an augmented image that was not used:



The difference between the original data set and the augmented data set is the following ...

The images are now in grayscale and normalized between 0.1 and 0.9

3. Describe, and identify where in your code, 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.

The code for my final model is located in the sixth cell of the ipython notebook.

My final model consisted of the following layers:

Layer	Description
Input	32x32x1 grayscale image
Convolution 5x5	1x1 stride, Valid padding, outputs 28x28x50
RELU	
Max pooling	2x2 stride, outputs 14x14x50
Convolution 2x2	1x1 stride, Valid padding, outputs 10x10x16
RELU	
Max pooling	2x2 stride, outputs 5x5x16
Flatten	Outputs 400
Fully connected #1	Outputs 120

Layer	Description
Relu	
Fully connected #2	Outputs 84
relu	
Dropout layer	0.5
Fully connected #3	Outputs 43

4. Describe how, and identify where in your code, 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.

The code for training the model is located in the 7th cell of the ipython notebook.

To train the model, I used the train. AdamOptimizer. The batch size was 128, the learning rate was 0.001 and the Epochs was 25

5. Describe the approach taken for finding a solution. 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.

The code for calculating the accuracy of the model is located in the ninth cell of the Ipython notebook.

My final model results were:

- training set accuracy of 0.986
- validation set accuracy of 0.931

- test set accuracy of 0.908

If an iterative approach was chosen:

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

  The first architecture used was Lenet 5 with RGB images and no normalization and 10 Epochs. Validation accuracy was 0.874%.

- What were some problems with the initial architecture?
- Based on the success of other projects it appeared a validation accuracy in the mid to high 90's should be achievable
- 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 over fitting or under fitting. 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.
- Several iterations were tried with various results:
  - The first adjustment made after normalizing the data was to add a drop out layer at 0.5 to the final fully connected layer and raising the epochs to 25. This took the validation accuracy from 0.904 to 0.917 on the RGB data and from 0.87 to 0.914 on the grayscale data
  - A second adjustment was made to add 3 dropout layers at 0.5 for each fully connected layer and raising the epochs to 50. This reduced the overall performance taking the validation accuracy from 0.904 to 0.842 on the RGB data and from 0.87 to 0.80 on the grayscale data
  - The final adjustment was raising the hidden layers from 6 to 50 on the first layer of the network and adding a drop out layer at 0.5 to the final fully connected layer and raising the epochs to 25. This took the validation accuracy from 0.904 to 0.927 on the RGB data and from 0.87 to 0.931 on the grayscale data
  -
  -
- Which parameters were tuned? How were they adjusted and why?
- I increased the number of hidden layers on the first layer from 6 to 50

- 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?
- A dropout layer was added to reduce the probability of over fitting the data.

If a well known architecture was chosen:

- What architecture was chosen?
- The final model was based off the Lenet 5 architecture
- Why did you believe it would be relevant to the traffic sign application?
- This model was used as the starting point because convolutional neural networks have shown great success in image recognition problems. The convolutional layers allow for local correlation of image pixels. In pattern and image recognition applications, the best possible correct detection rates have been achieved using CNNs
-
- How does the final model's accuracy on the training, validation and test set provide evidence that the model is working well?
- A training accuracy of 0.986 was achieved
- A validation accuracy of 0.931 was achieved
- A testing accuracy of 0.908 was achieved
- A testing accuracy of almost 91% shows that our model is working well. We were also able to detect 4 out of the 5 images randomly picked from the web.

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:

The first image, dangerous curve to the left, might be difficult to classify because there are very few examples of this sign in the training data, under 400 out 34,799 examples. Also, 15 of the 43 signs are triangle shaped signs shaped like this one, which may make it harder to classify.

The 2nd image, a stop sign, has a unique shape which should help it be classified more easily. Converting to grayscale may cause issues with the solid color of the sign though.

The 3rd image, a 30kmh sign, might be difficult to classify because there are 8 different speed limit signs that are all similar. All 8 signs include a 0. Also 25 out of 43 signs are in the shape of a circle.

The 4th image is a 60KM/h sign. This sign will be difficult to classify for the same reasons listed for the 30Kmh sign.

The 5th image, a yield sign, might be difficult to classify because it has few distinguishing markings. It is a triangle with a red border and a white center.

2. Discuss the model's predictions on these new traffic signs and compare the results to predicting on the test set. Identify where in your code predictions were made. 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 (OPTIONAL: Discuss the results in more detail as described in the "Stand Out Suggestions" part of the rubric).

The code for making predictions on my final model is located in the eleventh cell of the Ipython notebook.

Here are the results of the prediction:

Image	Prediction
Curve to the left sign	Curve to the left sign
Stop	Stop
30 km/h	Yield
60 km/h	60 km/h
Yield	Yield

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 0.908

3. Describe how certain the model is when predicting on each of the five new images by looking at the softmax probabilities for each prediction and identify where in your code softmax probabilities were outputted. Provide the top 5 softmax probabilities for each image along with the sign type of each probability. (OPTIONAL: as described in the "Stand Out Suggestions" part of the rubric, visualizations can also be provided such as bar charts)

The code for making predictions on my final model is located in the 11th cell of the Ipython notebook.

For the first image, dangerous curve to the left sign, the model is relatively sure that this is a Dangerous Curve to the left sign (probability of 0.513), and the image does contain a Dangerous Curve to the left sign. The top five soft max probabilities were

Probability	Prediction
.513	Dangerous Curve to the left sign
.438	Slippery Road
.284	Bicycles crossing
.0298	Wild animals crossing
-8.17	Road work

For the second image, a stop sign, the model is relatively sure that this is a stop sign (probability of 0.199), and the image does contain a stop sign. The top five soft max probabilities were

Probability	Prediction
.199	Stop sign
.163	Keep Right
.124	30kmh
.0785	50kmh
.0626	Yield

For the third image, 30KM/h sign, the model is relatively sure that this is a 60kmh sign (probability of 0.348), and the image does not contain a 60kmh sign. The top five soft max probabilities were

Probability	Prediction
.348	60kmh
.192	Yield
.136	Ahead Only
.0784	50kmh
-.111	Go straight

For the fourth image, a 60KM/H sign, the model is relatively sure that this is a 60kmhsign (probability of 0.396), and the image does contain a 60kmh sign. The top five soft max probabilities were

Probability	Prediction
.396	60kmh
.348	50kmh
.626	80kmh
.0184	30kmh
.008	Right-of-way

For the fifth image, a yield sign, the model is relatively sure that this is a yield sign (probability of 0.782), and the image does contain a yield sign. The top five soft max probabilities were

Probability	Prediction
.782	Yield
.336	60kmh
.122	50 kmh
.115	Ahead Only
-.023	Road Work