Project: Build a Traffic Sign Recognition Classifier

Juan Cheng **2017.08.11**

The goals / steps of this project are the following:

- Load the 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

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.

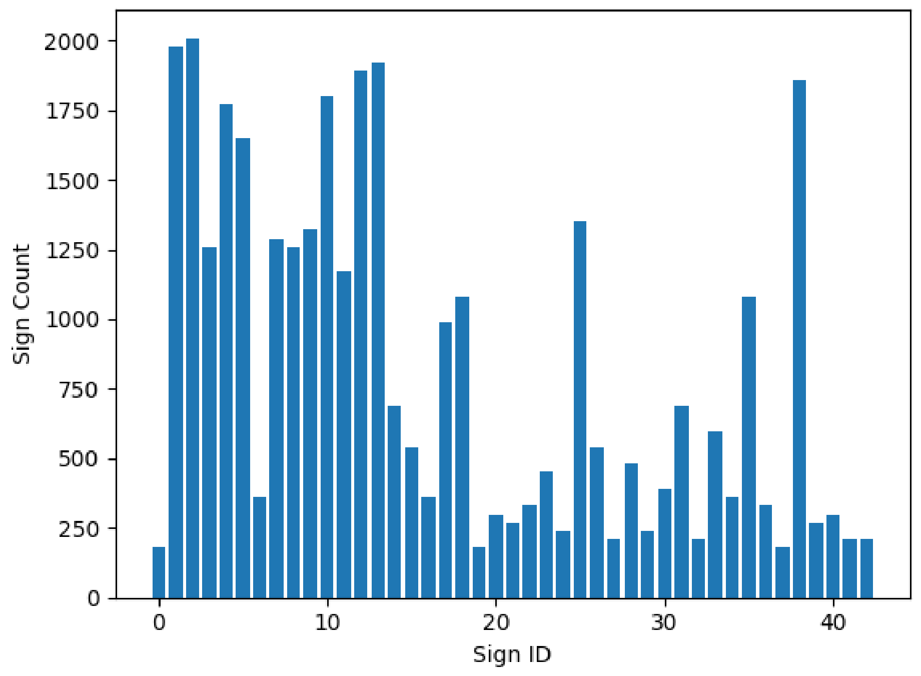
I used the numpy library to calculate summary statistics of the traffic signs data set:

- The size of training set is 34799
- The size of the validation set is 4410
- The size of test set is 12630
- 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.

Here is an exploratory visualization of the data set. It is a bar chart showing how the data is distributed among 43 different traffic signs.

It shows that the data set is not evenly distributed, which may cause the training model bias towards those signs have higher data sets and less favorable for signs with lower data counts.



Design and Test a Model Architecture

Question 1

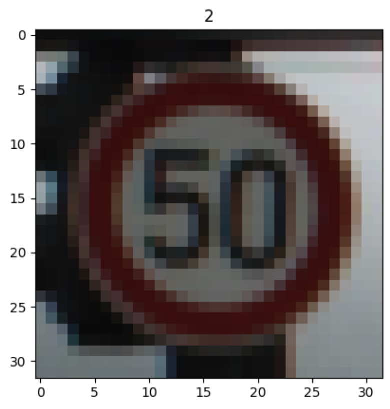
Describe how you preprocessed the data. What techniques were chosen and *why did you choose these techniques?*

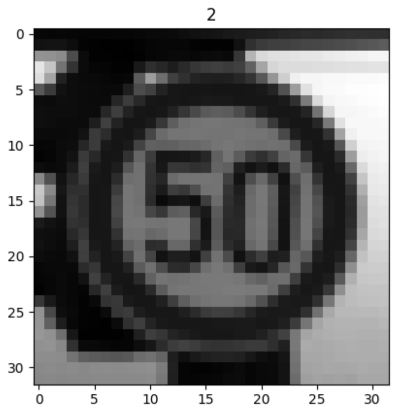
Answer:

My dataset preprocessing consisted of:

1. Converting to grayscale – This worked well as described in Sermanet and LeCun's traffic sign classification paper. It helps to reduce training time too by reducing the input depth of the first convolution stage.

Here is an example of a traffic sign image before and after gray-scaling.

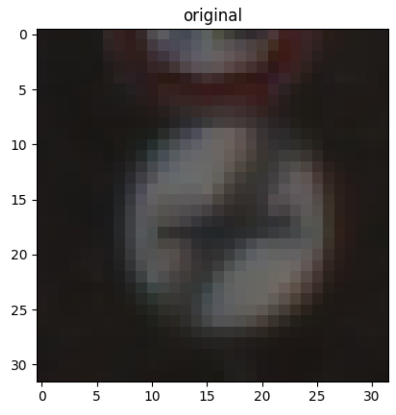


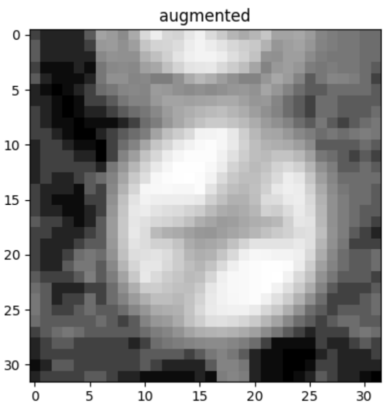


2. Histogram equalization – This was used to reduce variation in the contrast between different images.

3. Normalizing the data to the range (-1,1) - This was done using the line of code X_train_normalized = (X_train - 128)/128. The resulting dataset mean wasn't exactly zero, but it was reduced from about 82 (after preprocess step 1) to 0.026 (after preprocess step3). I chose to use this because it was suggested in the lessons and it worked well.

Here is an example of an original image and an augmented image:





Question 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	32x32x3 RGB image
Preprocessed Input	32x32x1 Gray image
Convolution 1	Input = 32x32x1 Output = 28x28x16 1x1 stride, valid padding
RELU	
2x2 Maxpooling	Input = 28x28x16 Output = 14x14x16
Convolution 2	Input = 14x14x16 Output = 10x10x32 1x1 stride, valid padding
RELU	
2x2 Maxpooling	Input = 10x10x32 Output = 5x5x32
Flatten	Input = 5x5x32 Output = 800
Fully Connected 1	Input = 800 Output = 400
RELU	
Fully Connected 2	Input = 400 Output = 200
RELU	
Fully Connected 3	Input = 200 Output = 43

Question 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.

Here are the hyper-parameters I used for training my model:

- epochs: 50
- batch size: 128
- learning rate: 0.0009

- mu: 0
- sigma: 0.1

Question 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.

My final model results were:

- training set accuracy of 1.000
- validation set accuracy of 0.966
- test set accuracy of 0.946

1) I first chose LeNet model as a starting point. Below is the configuration:

Layer	**Description (Config 1)**
Input	32x32x3 RGB image
Preprocessed Input	32x32x1 Gray image
Convolution 1	Input = 32x32x1 Output = 28x28x6 1x1 stride, valid padding
RELU	
2x2 Maxpooling	Input = 28x28x6 Output = 14x14x6
Convolution 2	Input = 14x14x6 Output = 10x10x16 1x1 stride, valid padding
RELU	
2x2 Maxpooling	Input = 10x10x16 Output = 5x5x16
Flatten	Input = 5x5x16 Output = 400
Fully Connected 1	Input = 400 Output = 120
RELU	
Fully Connected 2	Input = 120 Output = 84
RELU	
Fully Connected 3	Input = 84 Output = 43

The Validation Accuracy was **0.936**

2) I then tried to increase the depth of convolution and fully connected layers as follows, the Validation Accuracy went up to **0.955**

Layer	Description (Config 2)
Input	32x32x3 RGB image
Preprocessed Input	32x32x1 Gray image
Convolution 1	Input = 32x32x1 Output = 28x28x16 1x1 stride, valid padding
RELU	
2x2 Maxpooling	Input = 28x28x16 Output = 14x14x16
Convolution 2	Input = 14x14x16 Output = 10x10x32 1x1 stride, valid padding
RELU	
2x2 Maxpooling	Input = 10x10x32 Output = 5x5x32
Flatten	Input = 5x5x32 Output = 800
Fully Connected 1	Input = 800 Output = 400
RELU	
Fully Connected 2	Input = 400 Output = 200
RELU	
Fully Connected 3	Input = 200 Output = 43

3) I started trying a 2-stage ConvNet architecture as suggested in Sermanet and Lecun’s paper. Below is the configuration:

Layer	Description (Config 3)
Input	32x32x3 RGB image
Preprocessed Input	32x32x1 Gray image
Convolution 1	Input = 32x32x1 Output = 28x28x6 1x1 stride, valid padding
RELU	
2x2 Maxpooling	Input = 28x28x6 Output = 14x14x6
Convolution 2	Input = 14x14x6 Output = 10x10x16 1x1 stride, valid padding
RELU	
2x2 Maxpooling	Input = 10x10x16 Output = 5x5x16
Flatten 1	Input = 5x5x16 Output = 400
Convolution 3	Input = 5x5x16 Output = 1x1x400 1x1 stride, valid padding
Flatten 2	Input = 1x1x400 Output = 400
Concat Flatten 1 and 2	Output = 800
Fully Connected	Input = 800 Output = 43

The Validation Accuracy was **0.925**, not as good as the LeNet model in configuration 1.

4) **Based on configuration 3**, I tried **adding dropout to the model** as follows:

Layer	Description (Config 4)
Input	32x32x3 RGB image
Preprocessed Input	32x32x1 Gray image
Convolution 1	Input = 32x32x1 Output = 28x28x6 1x1 stride, valid padding
RELU	
2x2 Maxpooling	Input = 28x28x6 Output = 14x14x6
Convolution 2	Input = 14x14x6 Output = 10x10x16 1x1 stride, valid padding

RELU	
2x2 Maxpooling	Input = 10x10x16 Output = 5x5x16
Flatten 1	Input = 5x5x16 Output = 400
Convolution 3	Input = 5x5x16 Output = 1x1x400 1x1 stride, valid padding
Flatten 2	Input = 1x1x400 Output = 400
Concat Flatten 1 and 2	Output = 800
Dropout	Keep probability = 0.5
Fully Connected	Input = 800 Output = 43

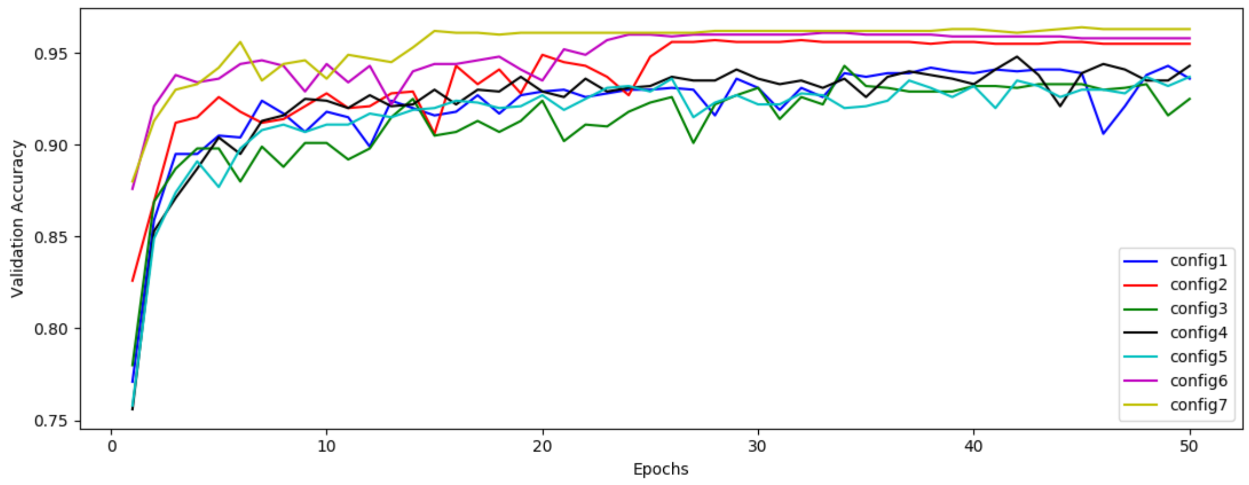
The Validation Accuracy went up to **0.943**

5) **Based on configuration 2**, I also tried **adding dropout to the model**. But it didn't help. As a matter of fact, the Validation Accuracy went down to **0.937** (Config 5)

6) From step 1 and 5, I found that **so far the best configuration was 2**. I also realized that when I was running configuration 1 to 5, I used the same data pre-processing methods: grayscale and normalization. Then I started wondering if data augmentation would make a difference. So for configuration 6, I tried **adding histogram equalization in the data preparation** and **re-ran Configuration 2 model**. It helped to boost Validation Accuracy to around **0.960**

7) I saw validation accuracy of configuration 6 plateaued out after epochs 24 while training accuracy stayed at 1.000 which means it's overfitting. So I **decreased the learning rate to 0.0009**. It helped to reach the peak validation accuracy sooner and the final Validation Accuracy was **0.966** which is slightly better than configuration 6.

Here is the Validation Accuracy Comparison for 7 configuration models:



8) Since configuration 7 model was the best, I selected it as the final model and ran it through the test set. The **Test Accuracy** was **0.946**.

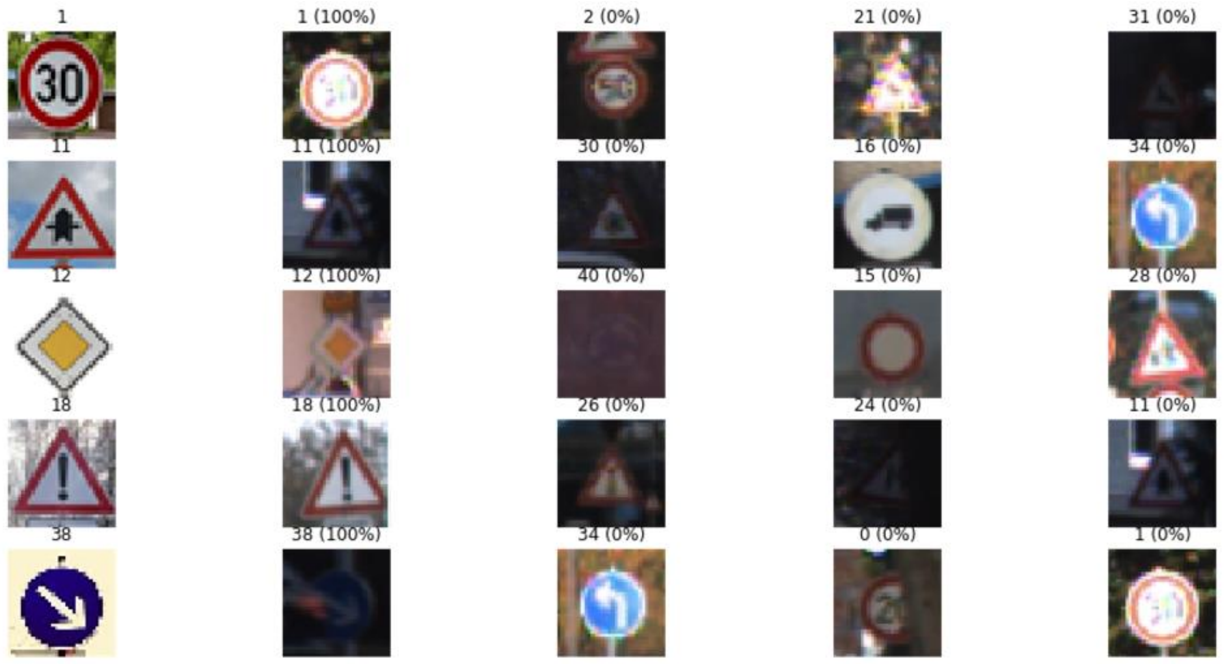
Test a Model on New Images

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

Here are the results of the prediction:

Image	Prediction
30 km/h	30 km/h
Right-of-way at the next intersection	Right-of-way at the next intersection
Priority road	Priority road
General caution	General caution
Keep right	Keep right

The model was able to correctly guess 5 of the 5 traffic signs, which gives an accuracy of 100%. This proves that the model worked pretty well – even better than the validation accuracy and test accuracy. In the real world, it's hard to achieve 100% considering the real data may not be as clear or recognizable as those 5 selected images, but it at least gives a positive outlook that using the chosen model we may obtain a very high accuracy. Below is the plot print out of the top five softmax probabilities for the predictions on the German traffic sign images I found on the web.


1
1 (100%)
2 (0%)
21 (0%)
31 (0%)
11
11 (100%)
30 (0%)
16 (0%)
34 (0%)
12
12 (100%)
40 (0%)
15 (0%)
28 (0%)
18
18 (100%)
26 (0%)
24 (0%)
11 (0%)
38
38 (100%)
34 (0%)
0 (0%)
1 (0%)