Data Set Summary & Exploration

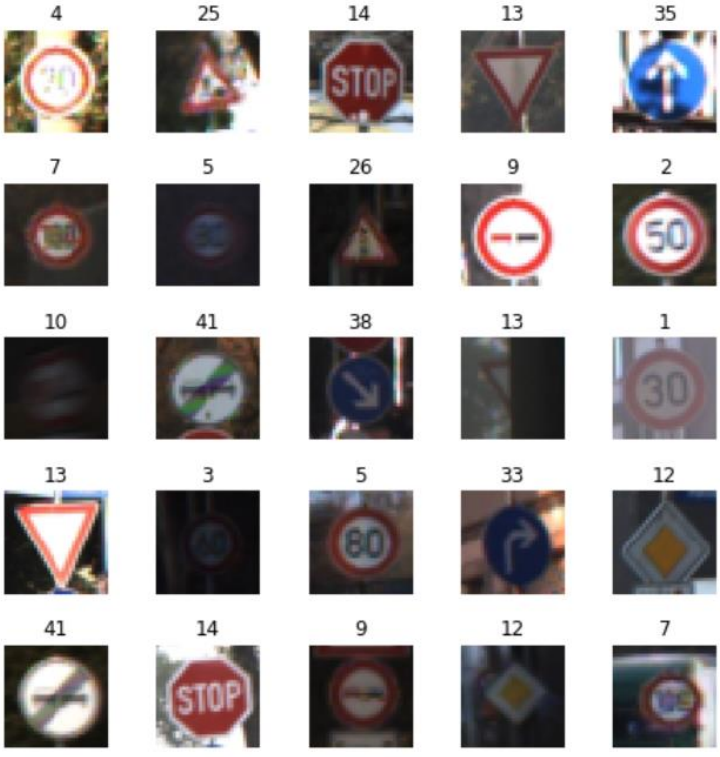
1. Provide a basic summary of the data set and identify where in your code the summary was done. In the code, the analysis should be done using python, numpy and/or pandas methods rather than hardcoding results manually.

The code for this step is contained in the second code cell of the IPython notebook. And the key results are summarized as follows:

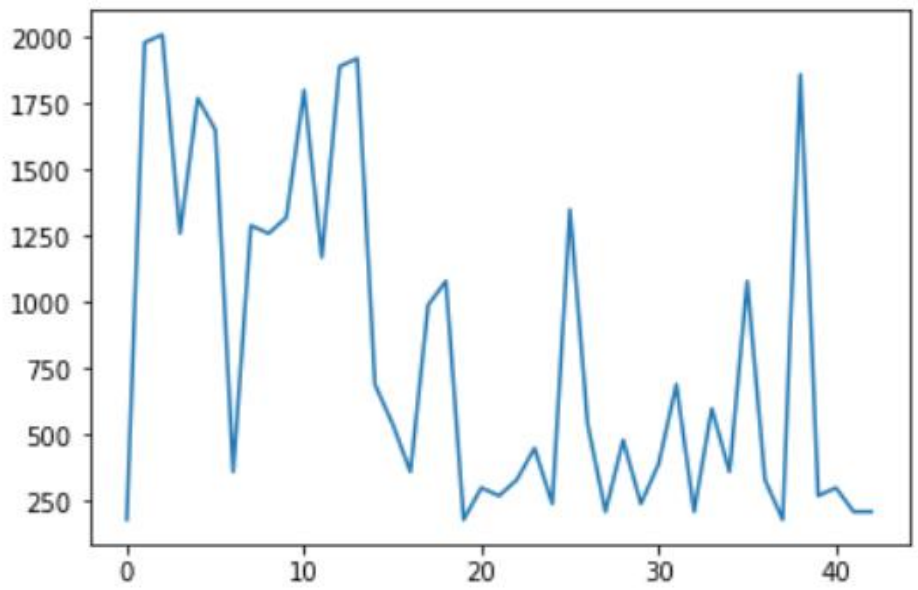
```
Number of training examples = 34799
Number of validation examples = 4410
Number of testing examples = 12630
Image data shape = (32, 32, 3)
Number of classes = 43
```

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

Some sample visualizations of the various categories are done as an exploration in the 3rd code cell, with the following sample output – this is just to get a feel for the data.



I also looked at the distribution of number of images per label:



X-Axis: The Label Categories, Y-Axis: Frequency of that category in the training set

There are a few labels with very low representation – this will be addressed through data augmentation later on.

Design and Test a Model Architecture

1. Describe how, and identify where in your code, you preprocessed the image data. What tecniques 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 step-2 section of the notebook.

As a first step, I decided to convert the images to grayscale because it significantly reduces the amount of training data fed to the neural network. As a human, we can recognize the meaning of (most, if not all) traffic signs even when shown a grayscale picture. Though color helps to amplify/highlight some signs, for an initial exercise I figure training speed is more critical than the extra accuracy gained. The code is in convert_to_gray function.

I also chose to normalize the image data based on the advice from the course. All images are put through a (x-127.5)/127.5 operation for this purpose. The code is in the

normalize function. This is needed because CNNs learn by adding gradient error vectors x learning rate computed during the backpropagation step. If the input training vectors are not scaled, the range of distribution of feature values can be different for each feature and the learning rate would cause corrections in each dimension that would differ from each other (Thanks to this article for explaining this)

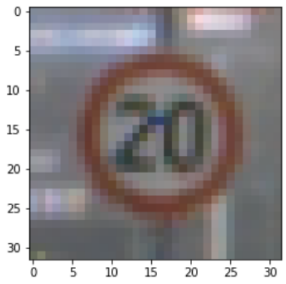
2. Describe how, and identify where in your code, you set up training, validation and testing data. How much data was in each set? Explain what techniques were used to split the data into these sets. (OPTIONAL: As described in the "Stand Out Suggestions" part of the rubric, if you generated additional 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 training, validation and test sets were provided separately for the project. As the 12th code cell shows, they have the respective dimensions as well

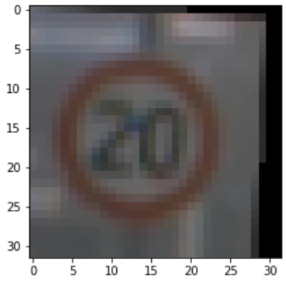
```
Train (40859, 32, 32, 1)
Validation (4410, 32, 32, 1)
Test (12630, 32, 32, 1)
```

The 9th code cell has the code to add more images to augment the database. This is because the training dataset is not balanced, and some categories have too few images. For any category that had less than MINLIMIT images, the code adds some "randomized" images to boost the number of images in that category. The randomization changes include a shift, rotate and exposure change to the picture.

Before



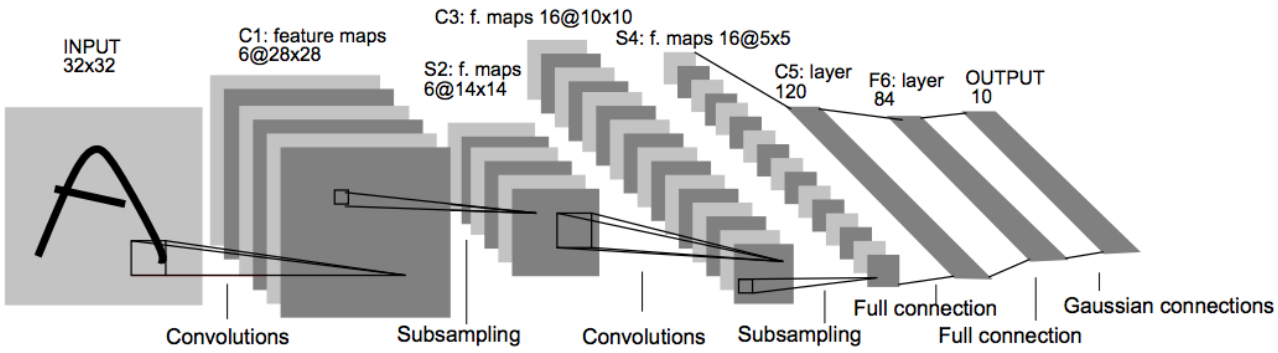
After



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 seventh cell of the ipython notebook.

I used the original LeNet Model as my inputs are exactly in the same format and in grayscale, and that seems to have worked well for the traffic sign case.



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 20th cell of the ipython notebook.

To train the model, I used an AdamOptimizer. I experimented with different types of BATCH SIZE, EPOCH and learning rate. With multiple iterations, I ended up at a batch size of 200, EPOCH = 400 and learning rate = 0.0009 which resulted in a validation accuracy of 93.6%

```
EPOCH 398
Validation Accuracy = 0.937

EPOCH 399
Validation Accuracy = 0.937

EPOCH 400
Validation Accuracy = 0.936
```

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.

It was a mixed approach; I started out with a well-known architecture. My feeling is that if I were to start from scratch, this step would have taken a lot of effort in trying to experiment with different combinations and figure out which architecture would give a good result. Once the architecture was defined, the rest of the trial and error approach was relatively simple as it just involved rerunning the code with different parameters. I fantasized about a GridSearch type of approach to scan the hyperparameters of the CNN, but then decided to just experiment with a handful of values instead as the computational runs were really expensive in time.

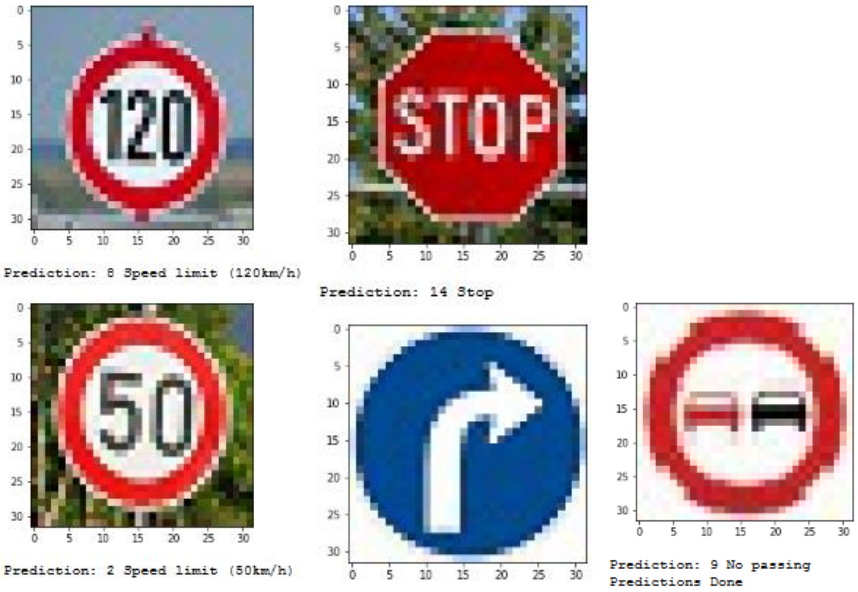
The code for calculating the accuracy of the model is located in the 18th cell of the Ipython notebook. My final model results were:

- validation set accuracy of 93.6%
- test set accuracy of 92.3%

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.

I found the following 5 images from the web. They are shown with the model predictions



At a high level, I don't particularly see any factors that might make it hard to classify these specific set of pictures; they seem to be relatively easy test cases.

I have picked images that look particularly bright and clear. Some of the pictures in the training set were very hazy/cloudy, and compared to that the features seem very clear in my set. In general, if the pictures used for training are particularly clearer and brighter than what was used to train the CNN, possible misclassification may happen. But I don't see that could be a case here though.

Another possible reason for misclassification is the shape of the signs. If the turn right arrow is in a square box vs circular box its possible that there is reason to confuse the CNN. However I haven't studied whether some of these signs can appear in different bounding box shape or if they are mandated to be in the same shape.

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 model was able to correctly guess all 5 of the traffic signs, which gives an accuracy of 100%. This is much better than the 92.3% accuracy obtained on the test set. I guess I just got lucky with the choices above – and I don't see these results as reflective of my model being perfect ☺

My ideal test would be to download a completely different dataset that is large, and then test the model on that (for example, I saw that there is a Belgian traffic sign database that was available). However I did not have time to do it yet as the image sizes were not matching and some significant preprocessing was needed to get it to match the example that I have.

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)

From the softmax probabilities, we can see that the model was pretty sure of the predict ions in each case. None of the secondary or terriary possibilities have any significant pro bability values.

```
TopKV2
(values=array([
       [  1.00000000e+00,   0.00000000e+00,   0.00000000e+00],
       [  1.00000000e+00,   2.76173604e-34,   0.00000000e+00],
       [  1.00000000e+00,   0.00000000e+00,   0.00000000e+00],
       [  1.00000000e+00,   0.00000000e+00,   0.00000000e+00],
       [  1.00000000e+00,   0.00000000e+00,   0.00000000e+00]]

indices=array([
       [ 8,  0,  1],
       [ 2,  1,  0],
       [14,  0,  1],
       [33,  0,  1],
       [ 9,  0,  1]]))
```