Traffic Sign Recognition - project code

By: Dror Meirovich

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.

Data Set Summary & Exploration

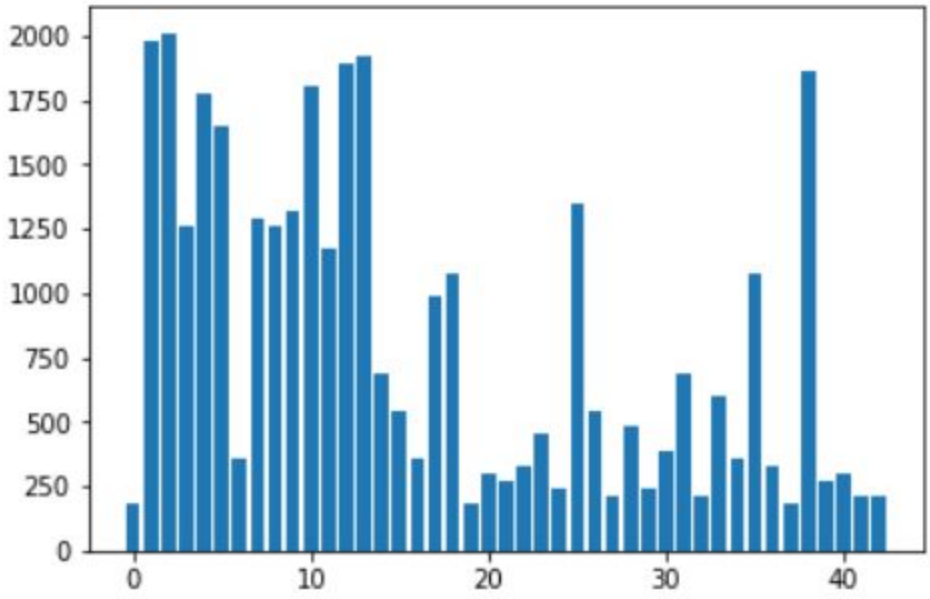
1. The given dataset is German traffic signs images, it consists of tens of thousands examples of 32x32 pixels each, in RGB format (input values range: 0 to 255). The source code for loading and displaying the data is in the first 3 cells in the notebook, under section "Step 0: Load the Data".

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

2. Visualization of the dataset:

Here is the image which its label is denoted as "0" in the training set:

The next bar chart shows the number of images for each label in the training set.



Design and Test a Model Architecture

1. Preprocessing the data:

First I decided to keep the number of image channels and not normalize it into gray scale because I didn't want to lose important data about the image. However, I decided to normalize the entire color values of each image to be at range: -1.0 to +1.0 because I wanted each image to have identical brightness, and same color range. The symmetrical value range around zero is for the sake of mathematical robustness when multiplied by positive and negative weights of the calculation net.

The big difference in the number of images per each label can cause convergence problems: The calculation net can "ignore" rare labels which means it will fail to learn and adjust the weights to identify them, while more common labels would be easier to identify. On the other hand, if the validation set is not balanced, the rare labels will have less effect on the final score of the net.

I decided to balance the training data and the validation data in the following way: First I over-sampled the rare labels in such way they will have the same number of occurrences as the most common label. While this solve the overall weight of each label in the total net score, it still had some convergence problem because it actually learned from fewer examples which were repeated again and again by the over-sampling process.

To solve this issue I added some small random noise to the over sampled images so the calculation net didn't adapt to the too few examples, and learned to overcome some noise in the input.

As mentioned before, this process of over-sampling + random noise was applied only on the training set and validation set, and only later the image was normalized to the -1,+1 range. As for the test set, only normalization was applied.

The source code for the preprocess phase is in the functions:

NormalizeImages()

PreprocessData4Train()

3. The model architecture:

I used the Le-Net architecture as taken from Le-Net-Lab exercise. I change it to receive 3-channel images and output 43 labels. The source code is in the function: LeNet()

My final model consisted of the following layers:

Layer	Description
Input	32x32x3 RGB image
Convolution 5x5	1x1 stride, VALID padding, outputs 28x28x6
RELU	
Max pooling	2x2 stride, outputs 14x14x6
Convolution 5x5	1x1 stride, VALID padding, outputs 10x10x16
RELU	
Max pooling	2x2 stride, outputs 5x5x16
Flattening	Input = 5x5x16. Output = 400
Fully connected	Input 400, outputs 120
RELU	
Fully Connected.	Input = 120. Output = 84
RELU	
Fully Connected	Input = 84. Output = 43 = number of classes

4. The model is trained inside the code cells below the title: “Train, Validate and Test the Model”

I used the preprocessed images for train set and validation set. The train method was stochastic gradient descent with 25 EPOCHS and 128 images as BATCH_SIZE. The learning rate starts at 0.01 and gets “cooler” when advancing each EPOCH until it reaches 0.0001, all these three are parameters for the train process. The motivation for cooling the learning rate is to start in big steps towards the optimum, and afterwards to notify the program that it approaches the optimum location, and therefore to make smaller changes to the model.
I also changed the learning rate of the last batch in each epoch to take into account the fact that it is actually smaller than the defined batch size, it could be even as small as 1 and that case may ruin some of the result at the end of each epoch.
All these are written in the function: Trainer()

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

- validation set accuracy of 93.12%
- test set accuracy of 92.72%

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.

The images can be found in the sub-directory: DownloadedFromGoogle/*.jpg
The labels were found after comparing the images and existing images in the training set.
Here are five German traffic signs that I found on the web:

Label: 1

Label: 3

Label: 27

Label: 17

Label: 14

I cropped and resized these images to fit into 32x32 pixel

2. The model succeed to predict 4 out of 5 images. It failed only for label 3 which was identified by the model as label 23.
When the model gave predictions it had 100% certainty even if it was wrong.
It was done in the function: PreprocessAndTest()

I learned a lot in this project!
Thank you!

Dror