Traffic Sign Recognition - project code

Second submission 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. 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 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 basic analysis of the data is in the first 3 cells in the notebook, under notebook section “Step 0: Load the Data”.

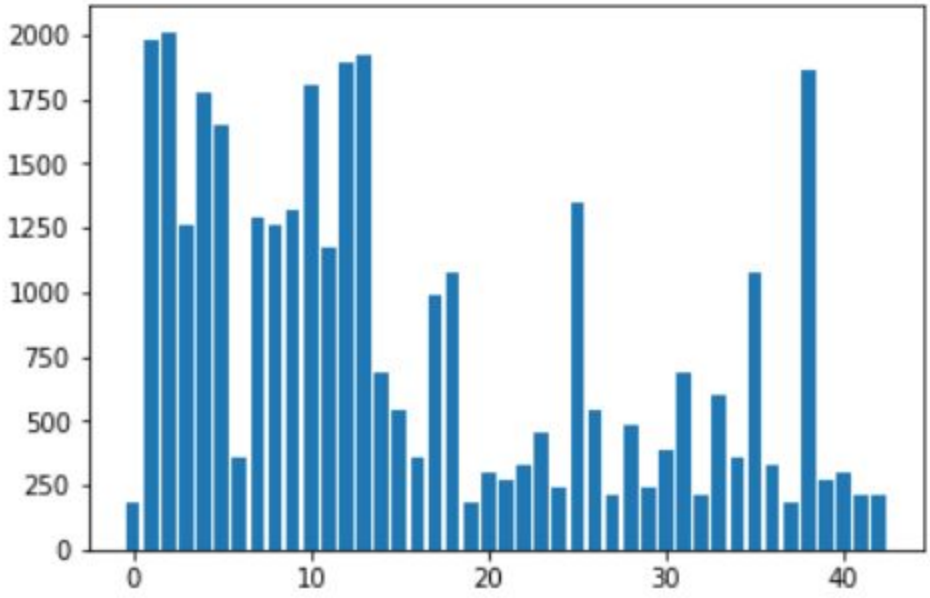
Number of training examples = 34799
Number of validation examples = 4410
Number of testing examples = 12630
Image data shape = (32, 32, 3)
Number of classes = 43

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

The relevant code for this section is splitted between the previous notebook section and “Step 1: Dataset Summary & Exploration“ (another notebook section)

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.



One can see the huge difference in the number of examples for each label in the training set, this can lead to overfitting of some labels while underfitting others. Moreover, the validation chart is similar but not identical, which means the accuracy rate that is calculated on the validation set may not be exact. These issues will be addressed in the pre-process section.

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 pre-process phase has two distinguished levels, and the relevant source code is located in two cells under the notebook section: “Pre-process the Data Set“
The lower level is consists of normalizing each image into a new range of values and is performed on all the data.
The higher level is consists of running all over the examples, counting them and create a new training/validation set with balanced number of examples.

Low level pre-process: The relevant source code is in the function NormalizeImage()
First I decided that the Neural Network will work the 3 channel image because there is an important information about the color differences inside each image (It was made originally for humans) and I cannot rely on the grayscale alone to be identified. However, I did calculate the grayscale of each image as a measurement of luminosity. I calculated the average and standard deviation of the luminosity, and truncated the image according in such way that its luminosity will not exceed +-2.5 standard deviation from the average. This way I aim to remove “luminosity spikes” before I normalize the entire image value range to -1.0,+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 luminosity can be added or reduced because of the mathematical linear relation between the luminosity and the RGB values. It doesn’t matter exactly what the RGB weights are (the are several common ways) as long as the entire formula is linear.
I also added an option to add random noise (small ratio from the standard deviation) to the image because I need it later in the “over sampling phase”

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

NormalizeImages()

NormalizeImage()

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)

High level pre-process
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.
The third issue is that the imbalance in the training set is different than in the validation set, i.e. different images have different proportion of validation number to training number.

I decided to address these balancing issues as follows:
1. Unit the training set and validation set into one united set (their labels as well)

2. Splitting (in a random manner) the united set into training and validation set with the predefined given proportion between the original validation to training set (around: 0.1267 taken from original samples sizes)
Now I have two distinguished sets with the same ratio between each label.
3. Decide what should be the number of examples of each image. First I thought it would be a good idea to take the maximum number, because I didn't want to loose any image information, but this would give me a lot of noise for images that have poor number of samples. So my final decision was to the the median number of examples calculated from the vector containing the number of examples for each image. Images with lower number of examples will be over-sampled with noise, images with higher number of examples will be sub sampled which means random number of images will be selected to fit the desired global number of examples per label.
4. Using the previous steps as-is can lead the program to ignore many examples of the more common images in the training set, so my final trick is to process step 2 and 3 for each training epoch in this way, every time different images are taken into account in the training phase, and different images for the validation calculation. It would be more epochs for the program to converge but it will take smaller sets (median number of images with few noise comparing to the maximum number of images + a lot of noise)

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:

PreprocessData4Train()
UniteImagesData()
Split()
SplitImagesData()

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 model architecture:

I used the original Le-Net architecture as taken from Le-Net-Lab exercise. The only change wasI modify it to receive 3-channel images and output 43 labels. I didn't have time to check different architecture or to check what is the influence of dropout layers or other type of activation layer.

My approach to this project was to spend most of the time on finding the best normalization of the data and decide how to test it.

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. 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 model is trained inside the code cells below the title: “Train, Validate and Test the Model”,
The relevant source code:
evaluate()
TrainOneEpoch()
TrainerWithPreprocess()

As mentioned before, I decided to unite the entire training data and validation data only once, and split it and over sampled it for every epoch. I think I could write it more efficiently especially the normalization phase to be excluded from the inner loop.

The train method was AdamOptimizer() which converges faster than gradient descent (according to my own tests) The learning rate was the given default number 0.001, and the epoch BATCH_SIZE was the given default: 128 examples. My previous submission of this project included a changing learning rate that gets “cooler” in each epoch and a predefined epoch number for the entire training phase. I decided to remove that approach and using a the desired accuracy ratio of 0.93 to terminate the training process. I also decided to limit the number of training epoch to 100 just in case the entire process does not converges.
I don’t feel entirely confident with this approach because the training process can reach the threshold of 0.93 just by a coincidence, so I added another parameter: avoid_coincidence=3 which means you must pass the desired accuracy test 3 times in a row so it would not considered as “coincidence” of convergence.
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.

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 approach to converge the training process was discussed in the section before.
My final model results were:

- validation set accuracy of 93.11% (three times in a row above the desired threshold 0.93)
- test set accuracy of 90.86%

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 original images can be found in the sub-directory: DownloadedFromGoogle/*.jpg
Here are five German traffic signs that I found on the web:

Label: 1

Label: 3

Label: 27

Label: 17

Label: 14

I choose the most “nice”images I could find because I wanted an easy and simple test for my predictive model. Nevertheless, it was somewhat harder than I thought as will be discussed later.
I manually cropped and resized these images to fit into 32x32 pixel and saved them in the subdirectory: 32x32fromGoogle/*.jpg

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 succeed to predict 3 out of 5 images. It failed for label 3 and 27 which was identified by the model as label 1 and 11 respectively.

I choose a clear and sharp images, still the predictions were average. I guess it was because of different zoom level and different angles of the image. I would have solve it by changing to over sampling method to deal with rotated images and different zoom scale instead of random noise as I wrote.

Another strange phenomena that I don't know its reason is that 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