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.

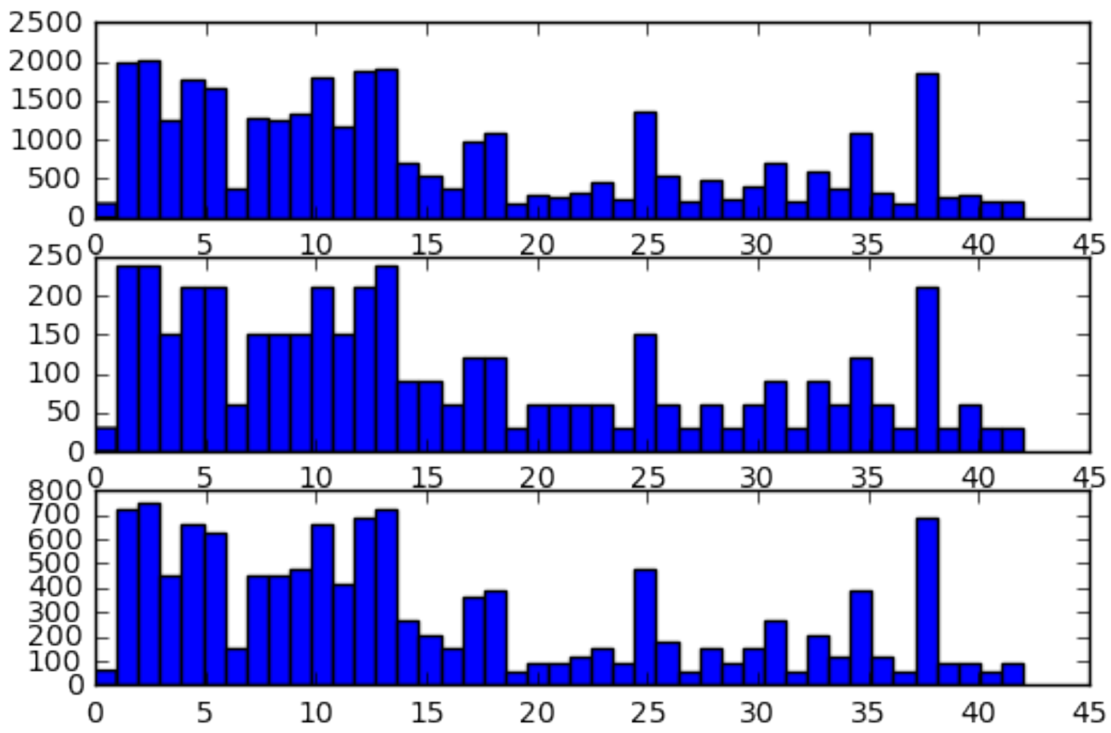
Simple dataset summaries are done in cell [2]. I used basic Python and numpy to get datset sizes and label counts.

- The size of training set is 34799
- 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 and identify where the code is in your code file.

Cell [3] contains code to map class ids to string names and a function that samples images for each label from dataset. Cell [4] contains a function for plotting samples. In cell [5] I perform sampling and plot 10 samples for each category. An important insight is that many images are very dark, presumably taken at night in low illumination. Even human has to squint hard to be able to recognize these images.

In cell [6] I plot distribution of examples for each plot, also reproduced below.



Histogram plots show distribution of labels in training, validation and test sets. One can see that they are heavily imbalanced, with some labels having only 10% as many examples as labels with large count of samples.

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.

I did three simple preprocessing steps. Images equalization to reduce some of illumination problem, rebalancing training dataset to address low number of samples for certain labels and scaling images to 0-1 float range.

Cell [7] contains code for equalizing image channels. This is done with a simple scheme that remaps image values so that they are spread from 0 to 255 values. It works reasonably well on many dark images, but doesn't achieve much for dark images that have some small number of white pixels. Cell [10] contains plots of equalized images, showing that indeed many dark images are now brighter and show greater colour variation.

In cell [8] I rebalance training dataset so that each label has as many samples as label with largest number of samples. Rebalancing is done simply copying existing labels - a very crude scheme that could be improved upon with augmentation. Still, this simple approach improved my validation accuracy by ~2%.

Cell [9] plots distribution of rebalanced dataset to show that indeed each label is represented by an equal number of samples.

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)

Data was already split into training, test and validation sets in provided code (as per cell [1]). The only significant change I made was rebalancing training dataset in cell [8], as noted above. One could get much more fancy with augmented datasets, but for this project I chose to stick to simple, quick approaches.

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.

Cell [11] sets up placeholders for images and labels, as well as for keep probability and learning rate, since I'm using dropout layer and variable learning rate.

Prediction model, as well as loss and accuracy ops, are located in cell [12]. My model is a simple fully convolutional network inspired by VGG net.

Here's a summary:

Layer	Description
Input	32x32x3 RGB image
Convolution 3x3	1x1 stride, valid padding, 32 filters
ELU	
Max pooling	2x2 stride
Dropout	0.5 keep probability at training
Convolution 3x3	1x1 stride, valid padding, 32 filters
ELU	
Max pooling	2x2 stride
Dropout	0.5 keep probability at training
Convolution 3x3	1x1 stride, valid padding, num labels filters
ELU	

My initial network contained more filters, but some experiments showed that decreasing filters count didn't hurt performance at all. I also tried using L1 and L2 losses to fight ~5% overfitting to training set, but to no effect - I couldn't decrease overfitting without hurting overall performance.

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.

Cell [13] contains a simple helper function to evaluate performance on a dateset. Training is done in cell [14]. There are only a few points worth mentioning:
* I used variable training rate, going with 0.001 for first 5 epochs and 0.0002 for last 5 epochs
* I train on rebalanced training set, but estimate training set statistics on original dataset. This is because rebalanced dataset simply repeats many elements, hence its statistics would be skewed

Everything else pretty much follows standard practices. 128 batch size, which I didn't try to optimize at all, and Adam optimizer, which has served me well in many projects so far and is my default choice.

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.

Accuracy on training and validations sets are reported in cell [14] and for test set in cell [15]

My final model results were:
* training set accuracy of 0.9961
* validation set accuracy of 0.9547
* test set accuracy of 0.9450

As one can see model clearly overfits to training data. I tried using L1 and L2 losses to prevent that, but to no avail - small regularization coefficient didn't change output at all and increasing it hurt overall performance (although indeed narrowed the accuracy between training and validation sets somewhat).

I used a slightly modified design of VGG net that uses blocks [3x3 convolution, 3x3 convolution, max pooling]. My network uses only two blocks of [3x3 convolution, max pooling, dropout], since problem is simple enough that introducing more layers doesn't bring much benefit. After two blocks network output is [batch_size, 6, 6, 32] and I map it to classes with a classes count (or 43) filters of size [6, 6]. This is exactly equivalent to a single fully connected layer.

My intial network followed above design with larger number of layers and filters, but I found that a thinner and more shallow network achieved just as good results. I experimented with different learning rates and decided that using 0.001 for first 5 epochs and 0.0002 for last 5 was optimal. I trained for only 10 epochs and it seems that network continued to learn, so possibly small improvements could be gained by training longer.

My network is fully convolutional, which is equivalent to having only a single fully connected layer at the end. Increasing number of fully connected layers to 2 or 3 could likely bring up the accuracy somewhat, but I didn't try this.

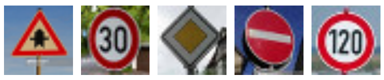
I also expect using a dynamic data augmentation scheme that would randomly perturb images intensities, rotate and translate them as well as scale them (and crop when necesary) would boost the accuracy by an appreciable margin. Again, I didn't try this approach, since I'm not trying to beat benchmarks in this project.

As noted before, my simple intensity equalization scheme works on most, but not all images. A more robust scheme, a different colorspace or, more generally, a representation less sensitive to illumination would likely yield improved results.

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:



They are also plotted in cell [16].

I didn't choose images to be particularly tricky. Still, following could trip up the network:
* First image, or "Right-of-way at the next intersection", is aligned a bit different that training set images. In all training set images center of sign seems to be exactly at the center of image, while in this test image sign is shifted up a bit, so that it occupies mostly upper half of the image.
* Second to last image, or "No entry" sign, has a large out of plane rotation. Training dataset contains some out of plane rotations, but they don't seem to be as strong.

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).

Logits predictions for test images are calculated in cell [17] and accuracy in cell [18]. Prediction probablities and top 5 k results are calculated in cell [19].

Here are the results of the prediction:

Image	Prediction
Right-of-way at the next intersection	No entry
Speed limit (30km/h)	Speed limit (30km/h)
Priority road	Priority road
No entry	No entry
Speed limit (120km/h)	Speed limit (120km/h)

The model was able to correctly guess 4 of the 5 traffic signs, which gives an accuracy of 80%. With a test set of only 5 images it would be wrong to make any statements about real world accuracy at large, but results are at least promising.

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)

Softmax computations are done in cell [19]. Model is 100% sure about all of its predictions, including the the single wrong prediction. Since all but top prediction had true 0% (rather than say 0.0001%) confidence, reporting top predictions 2 to 5, as well as their orders, isn't meanigful - it's just the order in which labels are defined by numerical ids. Therefore in here I only provide top prediction, for top 5 output please refer to cell [19].

For Right-of-way at the next intersection: No entry: 100.0% (WRONG PREDICTION), everything else 0%

For Speed limit (30km/h): Speed limit (30km/h): 100.0%, everything else 0%

For Priority road: Priority road: 100.0%, everything else 0%

For No entry: No entry: 100.0%, everything else 0%

For Speed limit (120km/h): Speed limit (120km/h): 100.0%, everything else 0%