

Traffic Sign Recognition

Writeup

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.

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. In the code, the analysis should be done using python, numpy and/or pandas methods rather than hardcoding results manually.

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

The size of training set is 34,799.

The size of the validation set is 4,410.

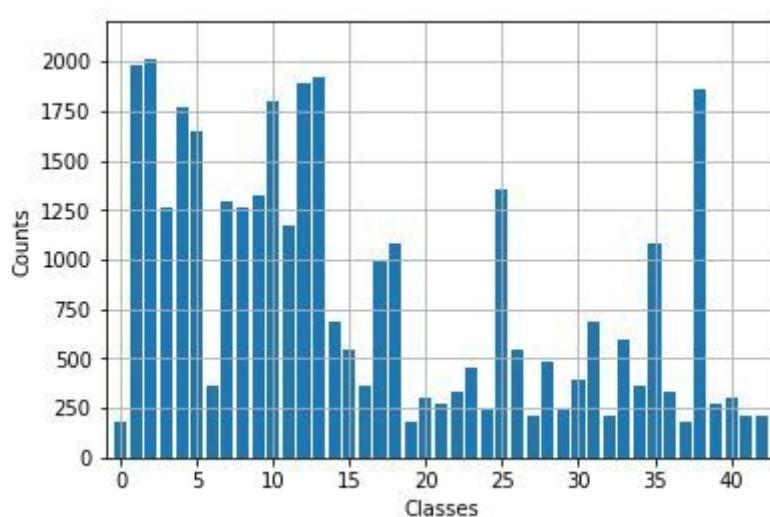
The size of test set is 12,630.

The shape of a traffic sign image is 32x32

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.



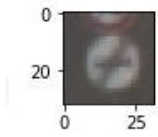
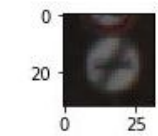
As you can see, there are big differences among the classes. The smallest has less than 200 samples and the biggest has more 2,000 samples. For this reason, in the next steps, we'll try to minimize this difference, as suggested during the class.

Design and Test a Model Architecture

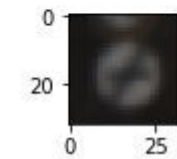
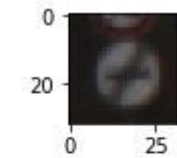
1. Describe how you preprocessed the image data. What techniques were chosen and why did you choose these techniques? Consider including images showing the output of each preprocessing technique. Pre-processing refers to techniques such as converting to grayscale, normalization, etc. (OPTIONAL: As described 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, and provide example images of the additional data. Then describe the characteristics of the augmented training set like number of images in the set, number of images for each class, etc.)

As a first step, I decided to augment the data for the classes that had less than one thousand samples. As I said before, my first goal was to minimize differences classes.

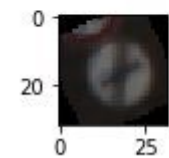
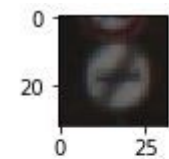
The first data augmentation was to change the brightness of the images. I applied it to all the classes with less than one thousand samples. I took all their images and I created a brother-image brighter than the original. I didn't find any library that worked well for me, so I decided to, simply, increment the RGB values of every image by 50. I think it's not the most efficient way to brighten an image. It took around 4 minutes using AWS to create 10,109 new images. Anyway, it did what it was supposed to do, so I decided that it was ok for me, even if it was not the most efficient way.



The second data augmentation was to blur images. I applied it to all the classes with less than five hundreds samples. In this case, i found a library that worked well for me and I used “Scipy.ndimage.gaussian_filter”.



The third data augmentation was to rotate images by 25 degrees. I applied it to all the classes with less than five hundreds samples. Also in this case I used the Scipy library and, in particular, I used “Scipy.ndimage.rotate”.



The fourth data augmentation was to flip upside down images. I applied it to the first one hundred images of every class. In this case, I used the “np.flipud” function



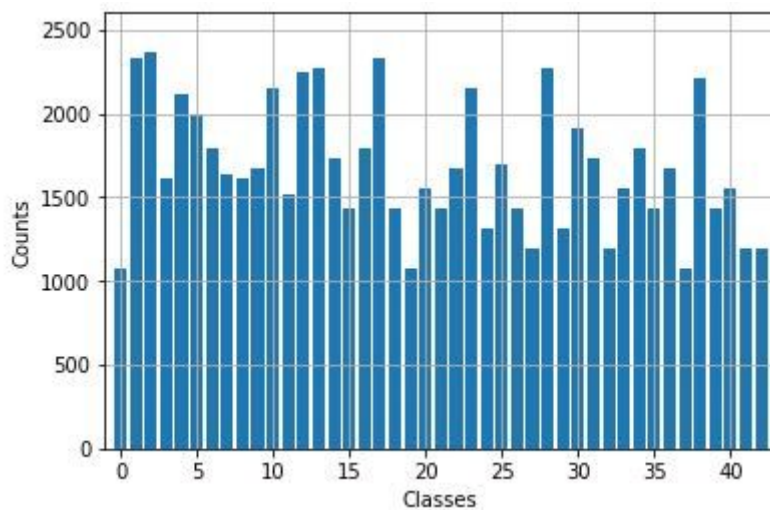
The fifth data augmentation was to randomly rotate images between -45° and $+45^\circ$. I applied it to the first one hundred images of every class. Again, I used the “`Scipy.ndimage.rotate`” function.



The sixth data augmentation was to blur images (again). I applied it to the first one hundred fifty images of every class. And, again, I used “`Scipy.ndimage.gaussian_filter`”.



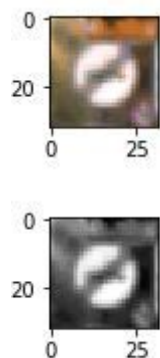
After all these augmentations, I concatenated all the new images to the old ones and I obtained a training set of 72,078 images (more than double that the original one). Here you can see the distribution between classes, after all the augmentations.



I obtained a more uniform dataset. Now, the smallest class is over one thousand samples and the biggest one is less the two thousands five hundreds samples.

As a second step, I decided to convert the images to grayscale, as suggested during class. To grayscale images, I simply summed the RGB values and divide it by 3.

Here is an example of a traffic sign image before and after grayscaling.



As a last step, I normalized the image data between -1 and +1. I divided the grayscale values by 128 and subtracted 1. After that I added one dimension to my images (using “np.expand_dims”), because during grayscaling I lost one dimension. So after that images were 32,32,1 and could be used in the LeNet architecture.

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.

I started implementing the LeNet model, but watching watching other students, I saw that was better to implement a modified version of this model. In particular, they used a model adapted from Sermanet/LeCunn traffic sign classification journal article.

My final model consisted of the following layers:

Layer ONE
Input 32x32x1 Grayscale and normalized image
Convolution 5x5 stride, valid padding, outputs 28x28x6
ReLU activation
Max pooling 2x2 stride, outputs 14x14x6

Layer TWO
Input 14x14x6
Convolution 5x5 stride, valid padding, outputs 10x10x16
ReLU activation
Max pooling 2x2 stride, outputs 5x5x16

Layer THREE
Input 5x5x16
Convolutional 1x1 stride, valid, padding, outputs 1x1x400
ReLU activation

Flatten Layer THREE output 1x1x400 to 400.
Flatten Layer TWO output 5x5x16 to 400.
Concatenate the two flatten and obtain 800.

Insert a Dropout with probability 0.5

Layer FOUR
Input 800.
Fully connected (800,43)
Output 43

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.

To train the model, I used a batch size of 32. I started using a large batch size of 1024, but my model was not accurate enough. Reading on the web, I found out that a smaller batch size of 32 or 64 was, probably, the best solution. It's still not very clear, why it works better, but, in my case, it works better.

I set the epochs to 100. At the beginning, I started with 30 epochs, but reading other students examples, I saw that it was better to increase this number. In fact, I train the model with 100 epochs and with 50 epochs and obtained a 94.6% accuracy in the first case, and 94,2% accuracy in the second case.

I didn't change mu and sigma. They are mu equals 0 and sigma equals 0.1.

I tried different learning rate and I found that 0.001 worked well for me.

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 99.9% (100 epochs)

validation set accuracy of 95.6% (50 epochs) and 95.0%(100 epochs)

test set accuracy of 94.2% (50 epochs), 94.6% (100 epochs) and 93.8 (100 epochs)

If an iterative approach was chosen:

What was the first architecture that was tried and why was it chosen?

I tried the LeNet architecture. Then, I used a modified version as suggested by other students. The modified version worked better and so I decide to use that version

What were some problems with the initial architecture?

The accuracy was low. In the low nighties. So, I had to do something to improve it.

How was the architecture adjusted and why was it adjusted? Typical adjustments could include choosing a different model architecture, adding or taking away layers (pooling, dropout, convolution, etc), using an activation function or changing the activation function. One common justification for adjusting an architecture would be due to overfitting or underfitting. A high accuracy on the training set but low accuracy on the validation set indicates over fitting; a low accuracy on both sets indicates under fitting.

My model has been always overfitting, but I obtained an accuracy on the test set over 93%, so I decided I was ok with that. If I will have more time at the end of this period, I will come back to this project and improve it.

To fix overfitting, I could increase the training data and have a more wide selection of data. I could use a simpler model that should reduce overfitting. I could change the hyperparameters mu and sigma. I could introduce another dropout layer or diminish the keep probability of the existing one.

Which parameters were tuned? How were they adjusted and why?

I tried different learning rate and I found that 0.001 was a good choice for my model. I set the dropout rate to 0.5. I tried different dropouts rate, but 0.5 was the best for my model.

What are some of the important design choices and why were they chosen? For example, why might a convolution layer work well with this problem? How might a dropout layer help with creating a successful model?

I didn't investigate this aspect. I found that it worked and, so, I was happy with that.

What architecture was chosen?

Why did you believe it would be relevant to the traffic sign application?

How does the final model's accuracy on the training, validation and test set provide evidence that the model is working well?

The model worked well on the training set, and poorly on the validation set and on the test set. Anyway, it was good enough to get an accuracy higher than 93%.

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 took some photo to Italian traffic sign closer to the Cuneo International Airport. Than I cropped the images to get 5 samples 32x32x3.






Here are the five Italian traffic signs:



The second, the fourth and the fifth image might be difficult to classify because they had low brightness and they are a little bit blurred.

2. Discuss the model's predictions on these new traffic signs and compare the results to predicting on the test set. 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.

Here are the results of the prediction:

IMAGE	LABEL	PREDICTION
	38	38
	2	1
	2	0
	13	13
	31	31

The model was able to correctly guess 3 of the 5 traffic signs, which gives an accuracy of 60%. This compares not favorably to the accuracy on the test set of my project, that it was over 90%.

The model has some problem recognizing 50km/h speed limit. It recognize that is a speed limit, but it does not recognize that is a 50km/h speed limit. I tried two different images and both images had the same problem. The other three have been correctly labeled.

The problem with the prediction are probably connected to the overfitting of my model.

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. Provide the top 5 softmax probabilities for each image along with the sign type of each probability.

The top five softmax probabilities for each image were:


Image	Label	38	0	1	2	3
	38	1.00000000 e+00	0.00000000 e+00	0.00000000 e+00	0.00000000 e+00	0.00000000 e+00


Image	Label	1	21	24	30	2
	2	9.96179581 e-01	3.78374197 e-03	3.53277828 e-05	8.19087120 e-07	5.69540077 e-07


Image	Label	0	1	4	2	38
	2	1.00000000 e+00	2.17445972 e-09	1.32508324 e-22	2.22942189 e-23	1.76571453 e-27



Image	Label	13	18	0	1	2
	13	1.00000000 e+00	2.49414172 e-29	0.00000000 e+00	0.00000000 e+00	0.00000000 e+00

Image	Label	31	23	29	21	19
	31	1.00000000 e+00	9.70950076 e-09	7.52558710 e-11	1.42445988 e-14	8.99289436 e-21

My model is always very certain of his prediction even when is wrong. Overfitting should be the reason.

4.Conclusion

This was a challenging project for me. It took me over one month to conclude it and I need to thank my mentor that guide me through.

My model is overfitting the train set. At the end of the period, If I will have time, I will come back to this a try to solve the overfitting problem.

Anyway, I got a sufficient accuracy on the validation set and the test set and my model worked with random images I took on the street. It had some problem with 50km/h speed limit and, again, if I have time at the end of this period I will investigate further and fix the problem.