Traffic-Sign-Classifier-Project

Write Up

17th August 2017

The goals / steps

1. Load the data set
2. Explore, summarize and visualize the data set
3. Design, train and test a model architecture
4. Use the model to make predictions on new images and analyze the softmax probabilities of the new images

1. Load the data set

The dataset was presented as pickled dictionary where the images are 4 Dimensional arrays divided into 3 subsets: Training, Validation and Test set..
The labels are a 1 Dimensional numpy array.
Later we will see that I had to use the provided *signnames.csv* file to associate each label number to the traffic sign description.

Loading was trivial:

```
training_file = 'traffic-signs-data/train.p'
validation_file= 'traffic-signs-data/valid.p'
testing_file = 'traffic-signs-data/test.p'

with open(training_file, mode='rb') as f:
    train = pickle.load(f)
with open(validation_file, mode='rb') as f:
    valid = pickle.load(f)
with open(testing_file, mode='rb') as f:
    test = pickle.load(f)
```

2. Explore, summarize and visualize the dataset

It is always important to look into the dataset (whenever possible), in order to get an idea of the problem and spot possible issues.

Summary:

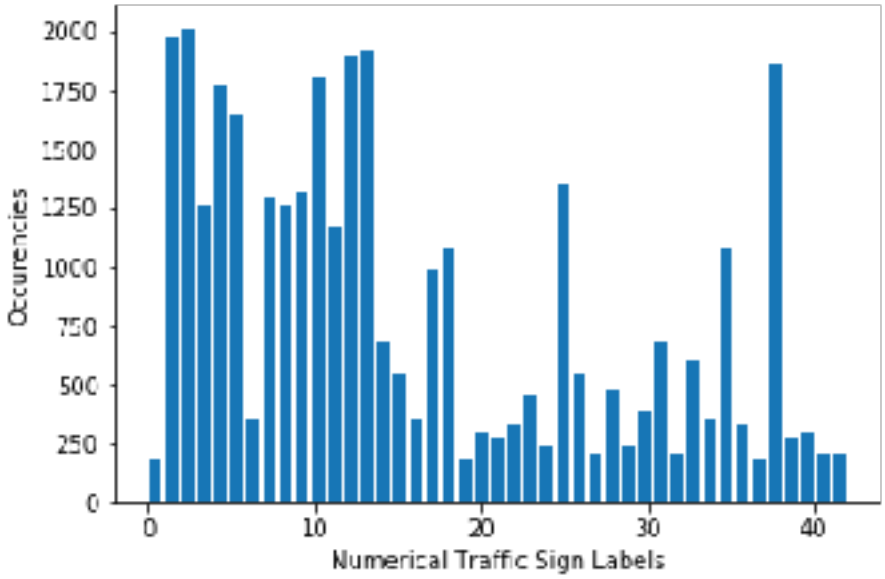
The amount of training data, the shape of the images, etc are the first important features to look at.

By using numpy methods, it's possible to check these values in one line of code.

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

Histogram:

An other useful insight is to plot the histogram of the total occurrences for each label across the training set.



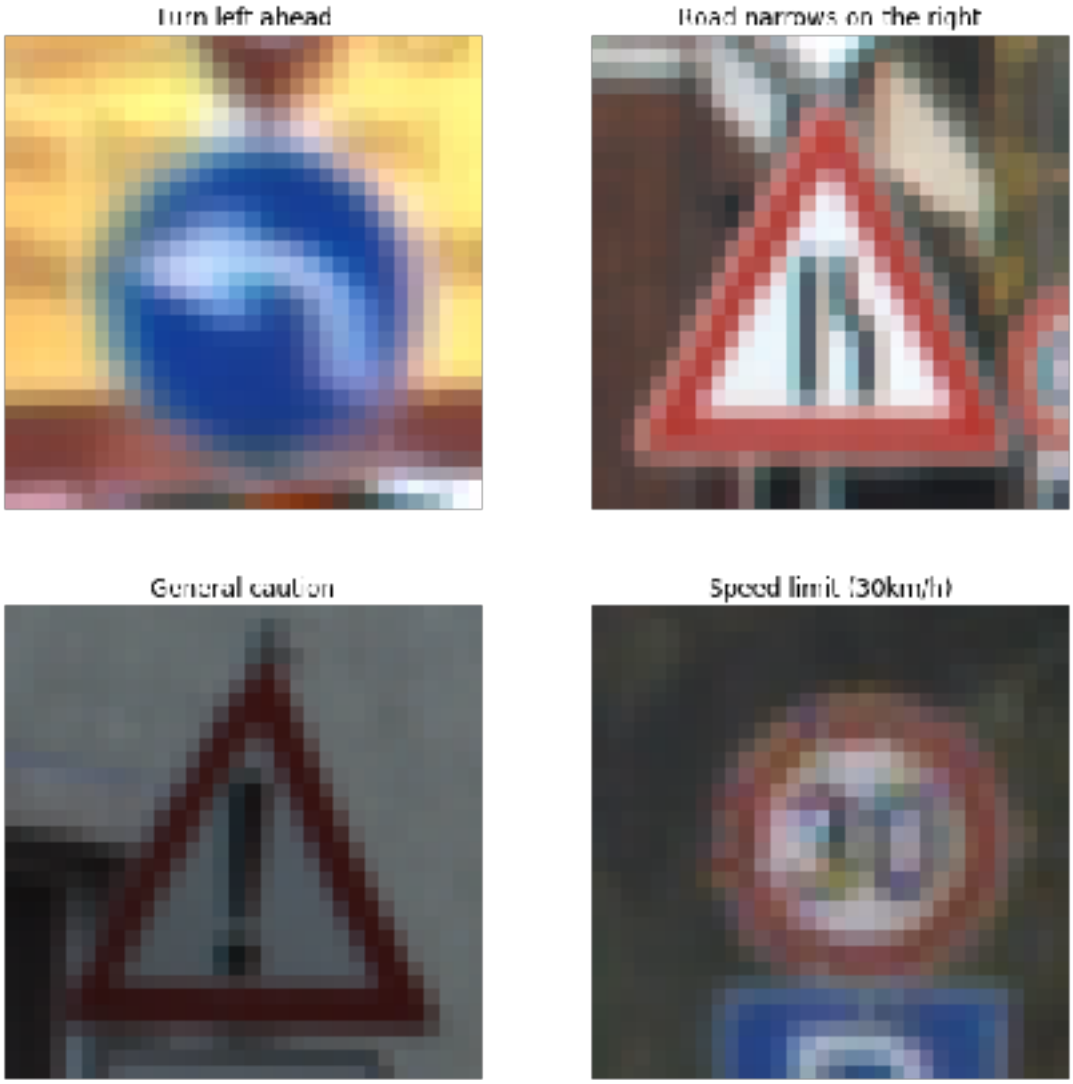
The histogram above shows that not all class are represented equally.
It is something to keep in mind during training, validation and testing in case I will notice some difficulties in getting to a certain level of accuracy

Visualization

Last step is to simply have a look at the images I am going to train my Neural Network on.

I used *Pandas* to load the *signnames.csv* and defined a function that returns the traffic sign description, given the numerical label.

Numpy provides useful methods for selecting random numbers.
I used one of these to choose 4 random signs.
Pyplot is the tool I chose for displaying the 4 randomly selected images, together with their description.



3. Design, train and test a model architecture

1. LeNet5 - No Preprocessing

Before performing any changes to the LeNet 5 architecture, before applying regularization and before performing any preprocessing, it is important to get an idea of where we are at with the minimum effort.

NB: The only “preprocessing” that is always necessary, is to shuffle the data.

- `convolutional layer: kernel size 5X5X6 - stride 1`
- `max pool layer: kernel size 2x2 - stride 1`
- `ReLU`
- `convolutional layer: kernel size 5X5X16 - stride 1`
- `max pool layer: kernel size 2x2 - stride 1`
- `ReLU`
- `Fully connected layer with 120 units`
- `ReLU`
- `Dropout`
- `Fully connected layer with 84 units`
- `ReLU`
- `Dropout`
- `Fully connected layer with number_of_classes units`

The LeNet 5 architecture was provided during classes.
I rewrote it anyway and made defined some functions (i.e. conv2d, or get_variables) that might turn out to be useful if I will decide to change the Neural Net architecture.

With this architecture, a learning rate of 1.0E-3, a batch size of 256, and dropout set to 0.5, I was able to reach 97.13% accuracy on the training set and 78.00% accuracy on the validation set in 15 epochs.

2. We are overfitting, but…

We are clearly overfitting, but before performing any data augmentation, adjusting the dropout or adding any other sort of regularization, I decided to convert the images to grey scale, since it appears that the information needed for classifying them is not in the colors. (Visualizing them earlier happened to be very useful)

This was of very good help since the validation accuracy jumped up to 88.82%, while the training accuracy remained around 97%.

3. A little improvement.

Because regularisation is already used, (I could try L2 Norm if I am still in trouble), I decided to increase the Neural Net capacity.

I increased the number of kernels in the first convolutional layer to 16, and in the second to 32.
Again, after 15 epochs I obtained a validation accuracy of 93.56%.

4. Final step .

More capacity seems to help, so I added an extra Convolutional layer with kernel size 3X3X32 and an other maxpooling layer.

Because we were already overfitting, I added more regularisation, more specifically L2 norm regularisation to the all weights (not biases) with constant multiplier L2_lambda 1E-2.

I also increased the number of epochs up to 30.

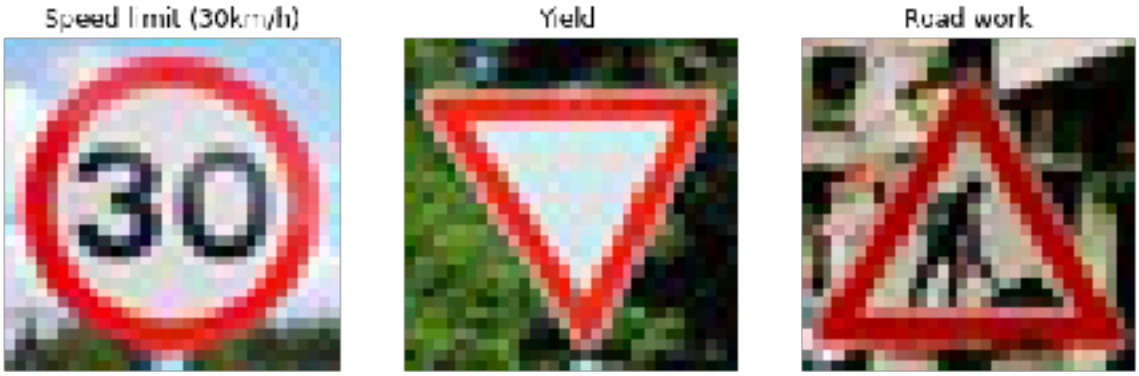
Validation accuracy of 94.49% :-)

I will use these results for testing on new images downloaded from the web.

4. Use the model to make predictions on new images and analyze the softmax probabilities of the new images

I downloaded 6 German Traffic signs from the web using Google Images search.
I cropped them in a square and resized them to 32x32 pixels.

These are the images with relative labels.



The images were transformed to grey scale and ran through the trained model. The total accuracy on these 6 pictures is 83%, so one image was misclassified.

In order to see what went wrong and to understand with which confidence the Neural Net classified each picture, I ran the model so that I would get the softmax values (probabilities) and the top5-k predictions with relative probability.

```
probabilities, test_accuracy, top5_value = sess.run([softmax, accuracy, top5], feed)
```

The results for each image, with the first 5 top predicitons and probabilities are shown in the following picture:

The misclassified image is the first one, the Speed Limit (30km/h).
We see that the Neural Network tends to have high confidence on all the images but this one.
It is promising the fact that the right label is the second choice with confidence 34.37%