Term1-P2: Traffic Sign Recognition
Sebastian Lena

Overview

Project #2 is about using deep learning, specifically convolutional neural networks, to classify images of German traffic signs. This is an important problem, for being able to classify an image of a traffic sign according to its type will allow a self-driving car to make important decisions.

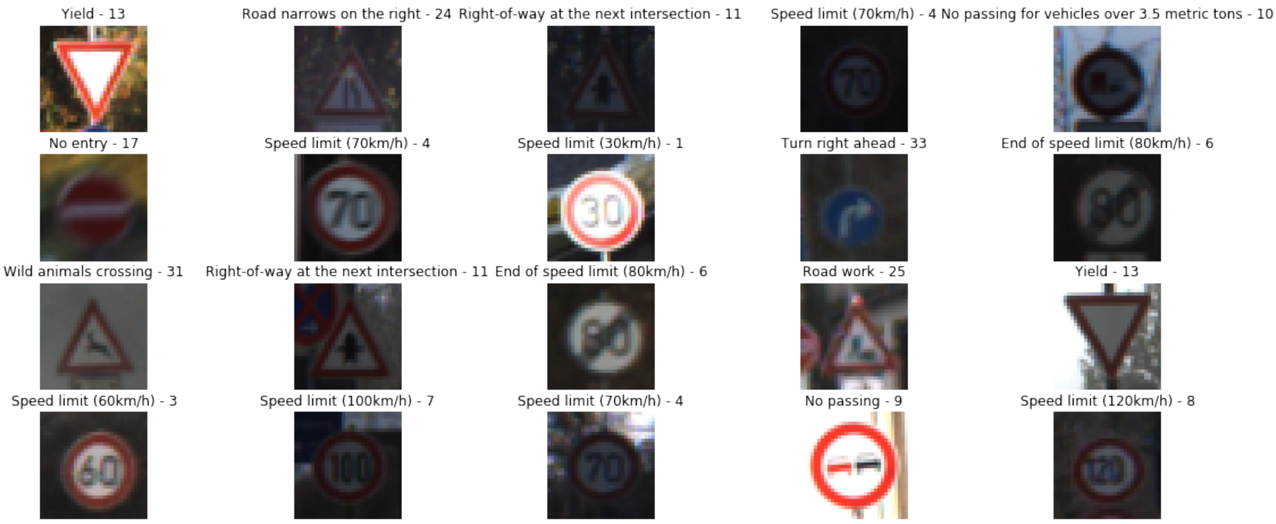
My code for this project is publicly available and can be found here:

https://github.com/B0WS3R/CarND-Term1-P2-Traffic-Sign-Classifier

Exploring the Dataset

- There 34,799 training images and 12,630 testing images
- The images are indeed 32x32 pixels
- There are 43 different classes of traffic signs labeled 0-42

Exploratory Visualization



Pre-processing the data

I did a minimal pre-processing using only the raw images from the dataset.
Because I felt like the color information was a valuable and important information I kept the images in RGB, I didn't converted them to gray scale.

I implement the image pre-processing with 2 steps:

1. Shuffling: Suffled the data to make sure about mixing (avoid groupings)
2. Normalizing: Normalized the data to [0, 1], easy diving the np.array by 255

Dividing the Data to Train, Validate, and Test

The data came pre divided into training, validation, and test sets. All I had to do was pre-process each of these sets accordingly

The Model architecture

I implemented the same architecture from the LeNet Lab with a few small changes
- I used RGB images
- Added a third convolutional layer

I used convolutions, activations (relu), maxpooling and linear combinations
The images are 32x32x3 and are 43 traffic signs categories.

The model look like:

Input -> RGB images of shape 32x32x3
Convolution with Activation -> Transform from 32x32x3 to 28x28x18 using a 5x5 filter.
Convolution with Activation -> Transform from 28x28x18 to 24x24x48 using a 5x5 filter.
Maxpool -> Condense the space to 12x12x48
Convolution with Activation -> Transform 12x12x48 into an 8x8x96 using a 4x4 filter
Maxpool -> Condense the space to 4x4x96, bringing the total volume from 6,144 down to 1,536.
Flatten -> Flatten the space down to one dimension of size 1,536 in preparation for the fully-connected layers.
Fully-Connected with Activation -> input(4x4x32x3) to (120x3) Calculate linear combinations of the features bringing the space down to size 360.
Fully-Connected with Activation -> input(120x3) to (84x3) Calculate linear combinations of the features bringing the space down to size 252.
Output -> input(252) to (43) Calculate linear combinations of the 252 features to produce estimates for each of the 43 classes of traffic signs.

Training the model

One-hot enconded the labels
I used softmax cross entropy to determine loss and the Adam optimizer
Hyperparameters:

- Learning rate = 0.001
- Batch size = 128
- Number of epochs = 10
- MU = 0
- SIGMA = 0.1

After each epoch the model was tested against the validation set. I set the model to save when a new validation record was reached. The final result:
From training were a validation accuracy of **97.6%**, and a training accuracy of **95.2%**. I think it happened a little overfitting in the model.

I started with LeNet, from there was a nice process of trial and error, tweaking with the hyperparameter but I could not get the
expected results. Later, and thanks to advices from the people from the forum (slack) I modified the architecture and tried with a third convolutional layer, getting better results.

Testing the Network on My Own Images

The choosen images were:

- Speed limit (60 km/h) - class id = 3
- Yield - class id = 13
- Stop - class id = 14
- No entropy - class id = 17
- Go straight or right - class id = 36

Nothing in particular with the selected images.

Result of predictions

I don't know how but the model appears to have predicted the new signs perfectly, with 100% of accuracy.

Softmax probabilities to visualize the certainty of its predictions

The model is somehow 100% certain of 5 out of 5 of the signs that I gave it. (I can't believe that)