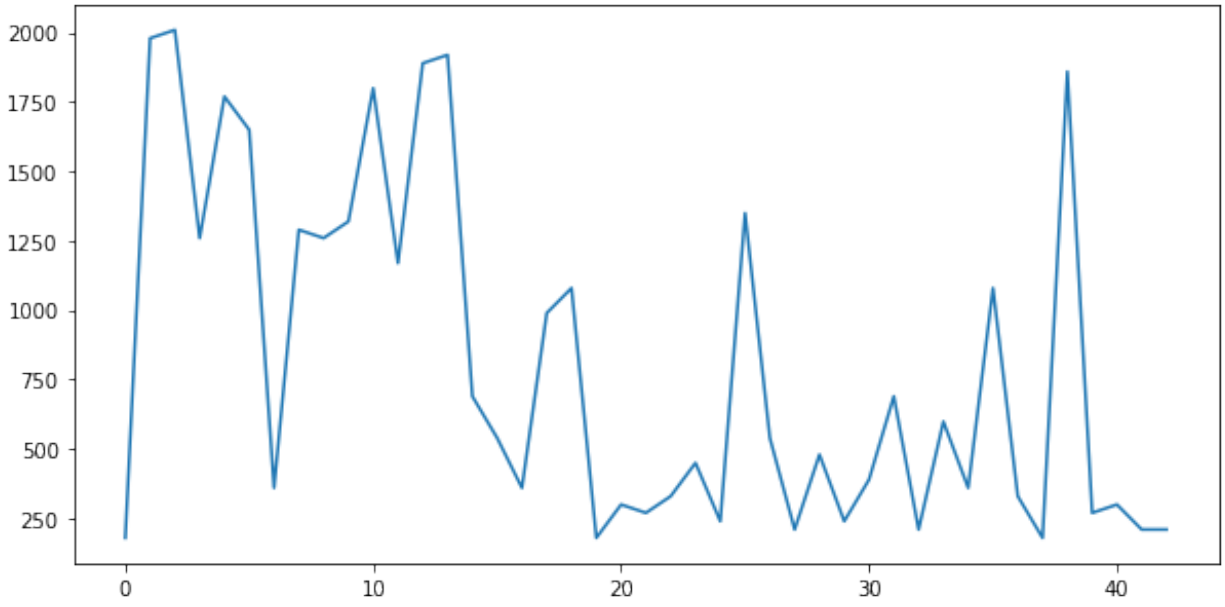
This Project is to implement a traffic sign classifier by utilizing deep learning

The dataset

I downloaded Germany traffic sign dataset from suggested website, it consists of 3 types of data: ~35k training data, 4410 validation data and ~12k test data. Each data includes a 32*32 color traffic sign image and its corresponding label; there are 43 labels in totals. I plotted the label distribution for all the training data, it looks like this:



Data preprocess

- Training data shuffle: the purpose is to reduce variance and make sure the model remains general and overfit less
- Normalization: it's to let training data have zero mean and equal variance. In the project, the color image has 3 channels: R, G and B. each channels ranges from 0 to 255; I take a formula update channel value to (pixel – 128.0) / 128.

Model Architecture:

The model is a typical LeNet architecture with some modifications.

- Layer1: a convolution layer, which takes input as the training data, applies a 1*1 stride filter to generate a 28*28 layer with depth as 6; then a ReLu activation function applied, and lastly followed by a max pool dropout to generate a 14*14 layer with depth as 6.
- Layer2: also a convolution layer, which takes input from Layer 1, applies a 1*1 stride filter to generate a 10*10 layer with depth as 16; then a ReLu activation function applied, and lastly followed by a max pool dropout to generate a 5*5 layer with depth as 6.

- Layer3: also a convolution layer, which takes input from Layer 2, applies a 1*1 stride filter to generate a 3*3 layer with depth as 32; then a ReLu activation function applied.

- Layer4: a full connection layer, which takes input from Layer3, flattens it as a 400 dimension layer, applied matrix multiplication and addition, generated a 120 dimension layer with ReLu applied.
- Layer5: a full connection layer, which takes input from Layer4, flattens it as a 120 dimension layer, applied matrix multiplication and addition, generated a 84 dimension layer with ReLu applied.
- Layer6: a full connection layer, which takes input from Layer5, flattens it as a 84 dimension layer, applied matrix multiplication and addition, generated a 43 dimension layer with ReLu applied, the output from this layer is our logits output of the entire training module.

Data Training and Data Validation

- Training parameter: epoch as 50, learning rate as 128 and batch size as 128
- Training operation: it's an Adam optimizer taking a cross entropy mean as loss function. The cross entropy takes the logits calculated from above LeNet architecture.
- Training process: it runs 50 epoch, in each epoch
    - It shuffles the training data set.
    - It divides the training data set into batches with 128 as batch size, the optimizer runs each batch data.
    - After we finish train the data set, we calculate the accuracy from each epoch.
- The recognition accuracy was printed out for each epoch, it finally comes to 93.4% at epoch 39, then climb very slow to 93.5% in the last epoch.
- In the end, we save the session data to file so we can use later.

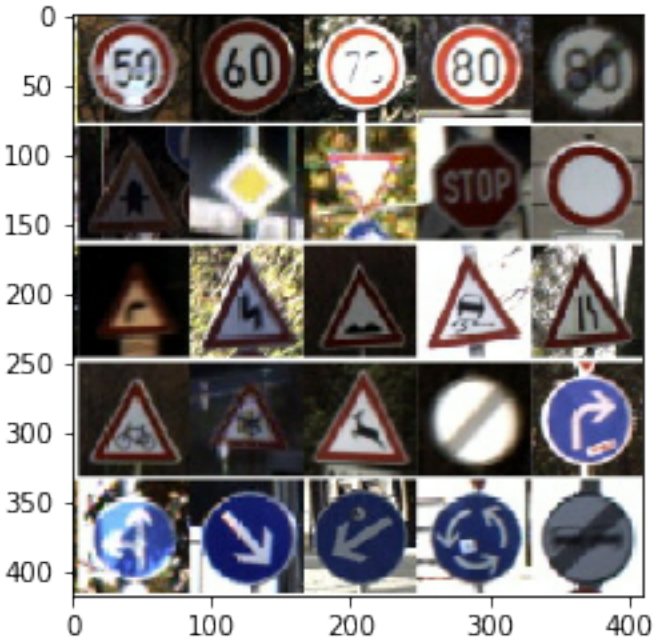
Data test

We load the session data from file (generated from training process), run the same validation operation as in validation process, we came up with a 85.5% accuracy.

Test on new image

- I downloaded a Germany traffic sign picture from website, it includes 25 small traffic sign pictures within.
- I cropped the original picture to get 25 small pictures, I then resized each small picture to 32*32; in the end I got 25 small picture with shape as 32*32*3.
- I used same way (pixel – 128.0)/128 to normalize each picture, loaded the session data from file, verified accuracy, surprisedly I got 100%. I had to check each picture and file signnames.csv to get the expected class though.

- I also analyzed the top 5 probability for those 25 pictures by utilizing tensor flow top_k activation function on soft max result on logits, I found out that most of top 1 is 100%, and the rest are 99%.
- The accuracy of the model upon those new images is 100%, better than the accuracy on test images which is 91.9%. I think the new images are brighter, test images seem blurred. This should be one of the reason of high accuracy on new images.
- Plotted new images like this:



An image from test dataset: