Project Report

Traffic Sign Classification

Name: Zeyu Tang

Dataset Exploration

Data summary

The given dataset “train.p”, “valid.p” and “test.p” are used as training set, validation set and test set respectively.

By loading the files above and running simple codes, we can get:

Number of training examples = 34799

Number of testing examples = 12630

Number of validation examples = 4410

Image data shape = (32, 32, 3)

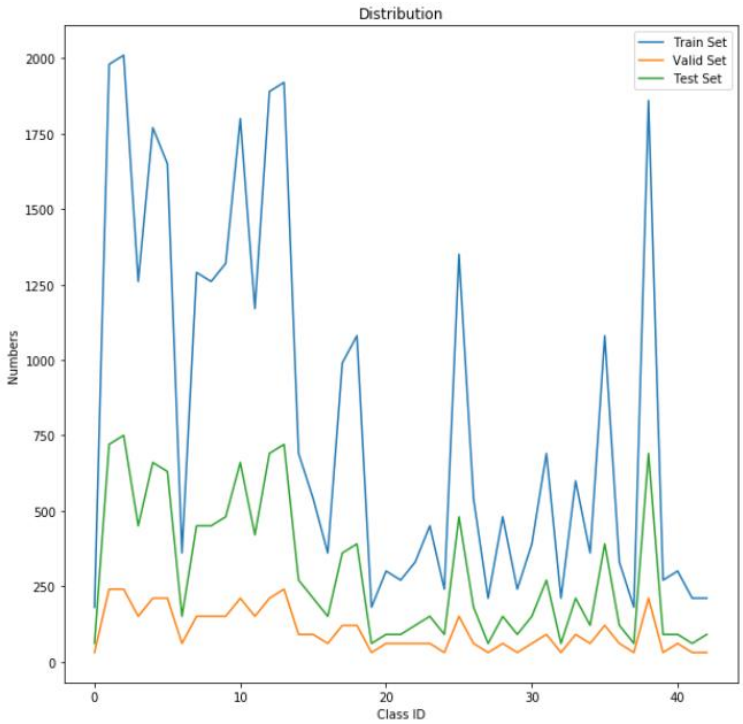
Number of classes = 43

Here we can see that images in dataset are 32*32 images with depth as 3, and have 43 classes. And the size of validation set is relatively small, meaning that the validation accuracy may fluctuate a lot even though the network model do not change.

Exploratory Visualization

For visualization and verifying the dataset, I randomly plot three images from training set, validation set and test set respectively. Here are results.

By referring to “signnames.csv”, we can see that all labels match corresponding images. By countering numbers of images in different classes, I draw a figure to visualize the distribution in different data sets. We can see that the overall tendencies of distribution in all three sets look the same, while the absolute differences on numbers between classes are larger in the data set with more images.

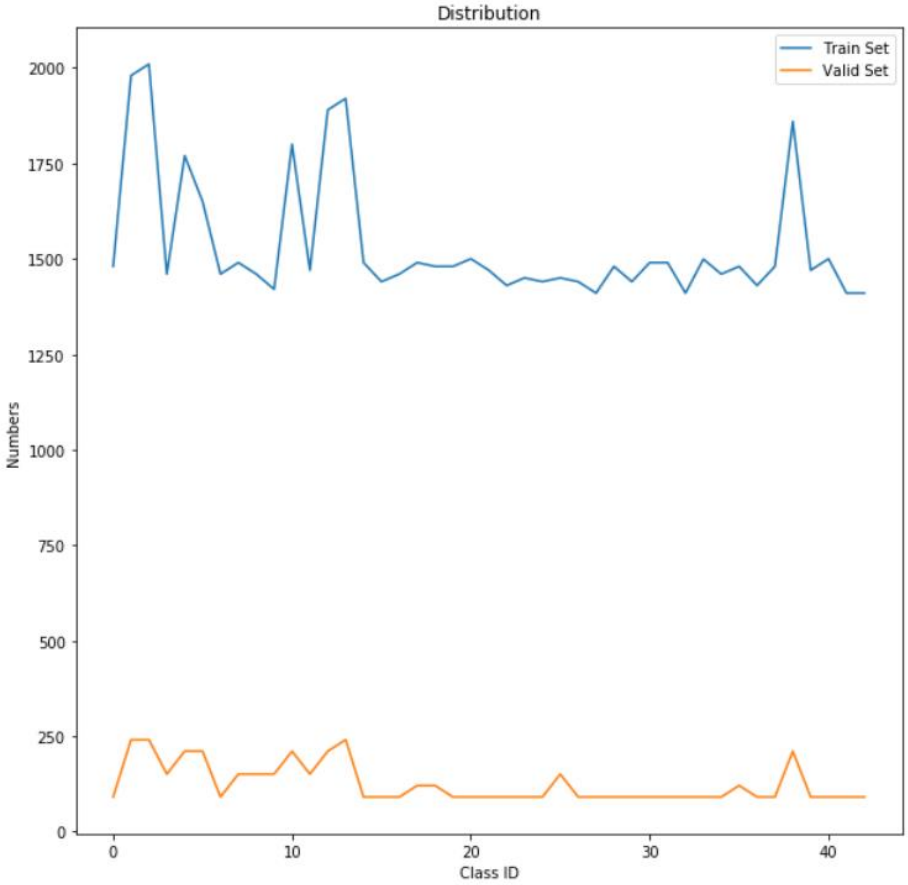


To reduce the effect that the unbalanced distribution to training model, in each of training and validation set, I choose to randomly some images from those classes that have images less than a minimum number, and apply histogram equalization to these images. Then repeat adding these images to corresponding sets, until the numbers images in the class get very close to the minimum number.
Here are chosen parameters during the adding process.

	Training set	Validation set
Min number of images in a class	1000	100
Number of randomly chosen images in a class	100	30
Final numbers of examples	65999	5370
Augmentation method	Histogram Equalization for each channel	

After several trials, I find that applying flipping and rotation to images lead to even lower accuracy. One possible reason might be in each class, all positions of signs in images do not have much differences, and adding relatively small amount of flipped or rotated increase the uncertainty in dataset while do not provide enough examples for model to learn such uncertainty.
Therefore, I choose to use histogram equalization.
Other parameters are chosen to make the distribution much evener while avoiding adding too much same images in one class.

New distribution figure:


Distribution
Train Set
Valid Set
2000
1750
1500
1250
1000
750
500
250
0
Numbers
0
10
20
30
40
Class ID

Design and Test a Model Architecture

Preprocessing

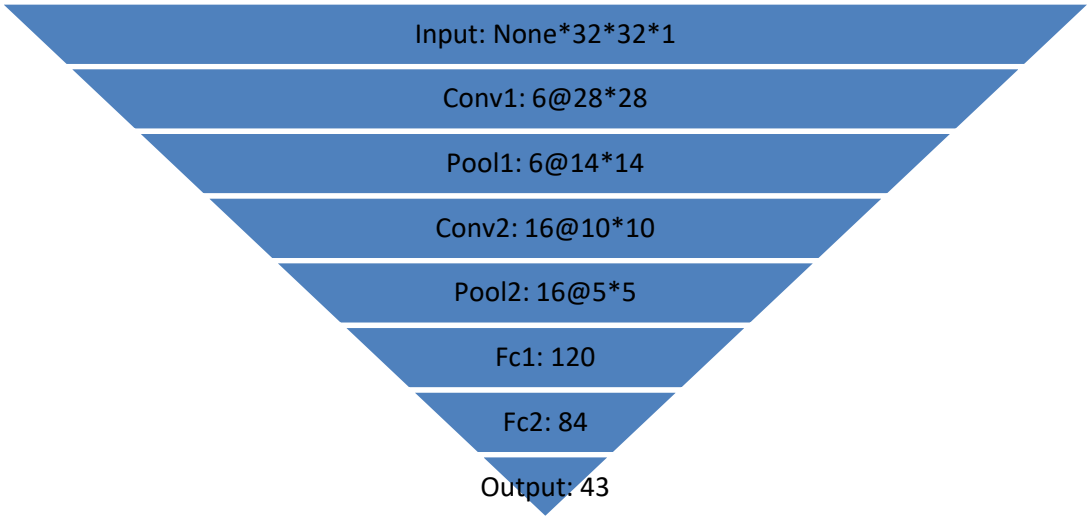
In the final code, I transfer rgb images into grayscale images, then normalized each pixel with formula (pixel - 128)/ 128.

Transferring rgb images into grayscale images reduces the input data for neural network and the number of weight in the first convolutional layer. Normalizing pixels makes pixels' value much smaller and have mean zero and equal variance. Therefore, the two approaches lead to less computation and higher accuracy in training process. Besides, I try to apply Histogram Equalization to all of dataset, but the training result changes little. And the method is already used in generating additional images to balance distributions of classes. Therefore, here I choose not to include the method.

Model Architecture

Here I use the typical LeNet.

The model contains two convolutional layers, two pooling layers, two fully connected layers and one output layer. The structure's detail are showed below.



LeNet Structure

Model Training

Here I calculate the mean of cross entropy as the loss. Then I use the minimize function of AdamOptimizer to update the network by backpropagation and minimize the loss.

After many trials, I choose parameters as below, which have great balance between training speed and accuracy.

Batch size: 128

Number of epochs: 30

Learning rate: 0.0008

keep_prob (drop out): 0.5

What is more, with 30 epochs and learning rate as 0.0008, training accuracy and validation accuracy could always keep almost stable at the last several epochs, meaning that the optimizer almost find the best solution. Lower learning rate and more epochs always mean better accuracy, but the values above should be enough for getting sufficient accuracy.

Solution Approach

As mentioned above, I transfer rgb images to grayscale images and normalize pixels' values in the pre-processing, and choose learning rate as 0.0008 and epochs as 30. What is more, the input data is shuffled before fed into training process.

Then after multiple trials, I choose to use drop out regularization on both of fully connected layers and keep_prob as 0.5, which provides the best performance on the training result.

Here are some trials' results.

Approach: drop out	Training accuracy	Validation accuracy
No drop out.	0.996	0.910
Drop out on fc1. keep_prob=0.75	0.999	0.936
Drop out on fc2. keep_prob=0.75	0.999	0.943
Drop out on fc1, fc2 keep_prob=0.75	0.999	0.953
Drop out on fc1, fc2 keep_prob=0.5	0.995	0.956

Trials results about dropping out

(Since the accuracy in the last several epochs keeps almost stable but still fluctuating, I use the mean of accuracy in the last 3 epochs as the final accuracy)

Here we can see that all results of trials are overfitting, training accuracies are all pretty close to 1, and drop out on more layers and lower keep_prob lead to less overfitting. Therefore, I choose the last approach of dropping out.

With applying the chosen drop out approach, here are some other trials' results.

Approach: Balance distribution of dataset		Training accuracy	Validation accuracy
Min number of images in a class in training set:	Process to additional images		
1000	Histogram Equalization	0.995	0.946
1500	Histogram Equalization	0.998	0.958
	Rotation 45 degree	0.995	0.922
	Flipping	0.995	0.918
	Do nothing	0.998	0.950

Trials results about balancing distribution of dataset

Here we can see that larger min number of images in a class in training set seems to lead to high accuracy. However, since I add lots of additional images that might be the same into dataset, the number with further larger value may lead to higher overfitting. Therefore, I choose to use 1500 as the number's value.

What is more, we can see that significant changes to the signs positions in additional images lead to relatively low accuracy. One possible reason might be in each class, all positions of signs in images do not have much differences, and adding relatively small amount of flipped or rotated increase the uncertainty in dataset while do not provide enough examples for model to learn such uncertainty.

Therefore, I choose to use histogram equalization to process additional images.

Finally, I run the model with test set. The test accuracy is 0.935

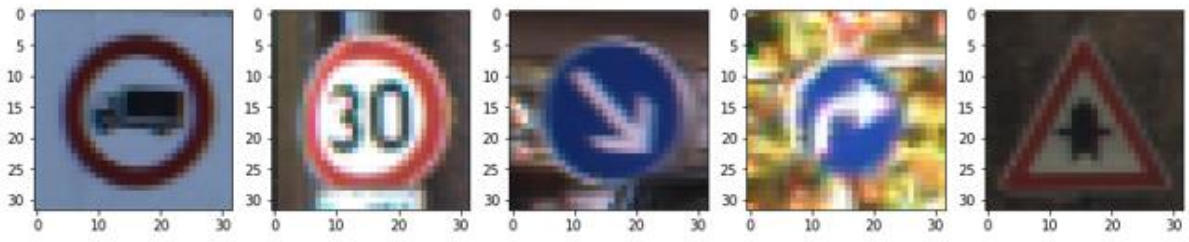
```
INFO:tensorflow:Restoring parameters from .\lenet
Test Accuracy = 0.935
```

Test a Model on New Images

Acquiring New Images

Since sign images are randomly sorted and named in German Traffic signs testing dataset, I choose the first 5 five images for testing, which's names are from "00000.ppm" to "00004.ppm".

I import the five images and resize them into 32*32 with linear interpolation. Results are showed below. The corresponding class ids are: [16, 1, 38, 33, 11].



We can see that the five images are randomly chosen and vary in color, shape and background. They should be representative enough for testing the model.

Performance on New Images

After restore the same session as above and the latest checkpoint, I feed the list of five images into network, and use argmax function to find the corresponding class id to the max scores in logits of each image.

The results are [16 1 38 33 11], which are exactly the same as labels.

Therefore, the accuracy for the prediction is 100%, which is higher than the testing accuracy. The possible reason why the accuracy is extremely high is the small number of new images for testing.

Model Certainty - Softmax Probabilities

After restore the same session as above and the latest checkpoint, I feed the list of five images into network and use tf.nn.softmax function transfer all scores in logits into probabilities. Then I use tf.nn.top_k function to get the top five probabilities of each images. The results are showed below.

```
INFO:tensorflow:Restoring parameters from .\lenet
[[1.0000000e+00 2.3332956e-09 1.0518225e-10 7.6868484e-11 2.0878995e-11]
 [1.0000000e+00 4.3010007e-10 7.5320739e-12 8.2214276e-13 2.4440275e-13]
 [1.0000000e+00 3.8197785e-27 2.2661552e-30 1.9431640e-31 4.2280501e-32]
 [9.9999881e-01 6.6417812e-07 4.9962301e-07 1.0390826e-08 2.2568952e-12]
 [9.9982494e-01 1.7504100e-04 4.3536563e-08 1.7205741e-08 5.7965105e-11]]
```

Probabilities in each row are the top five probabilities of one image.

We can see that the largest probabilities of each image is very close to 1 and the others and very tiny, meaning that the model are very certain with its prediction.

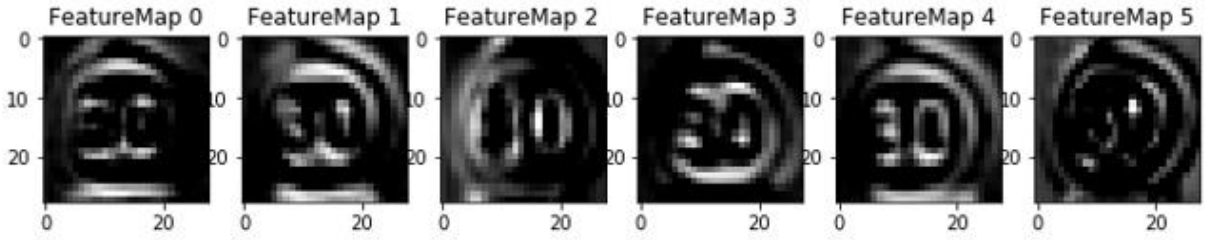
Visualize the Neural Network's State with Test Images

Here I restore the latest check point, import the image of 30 km/h sign, and use the given outputFeatureMap function to get feature maps in both of convolutional layers.

The testing image:

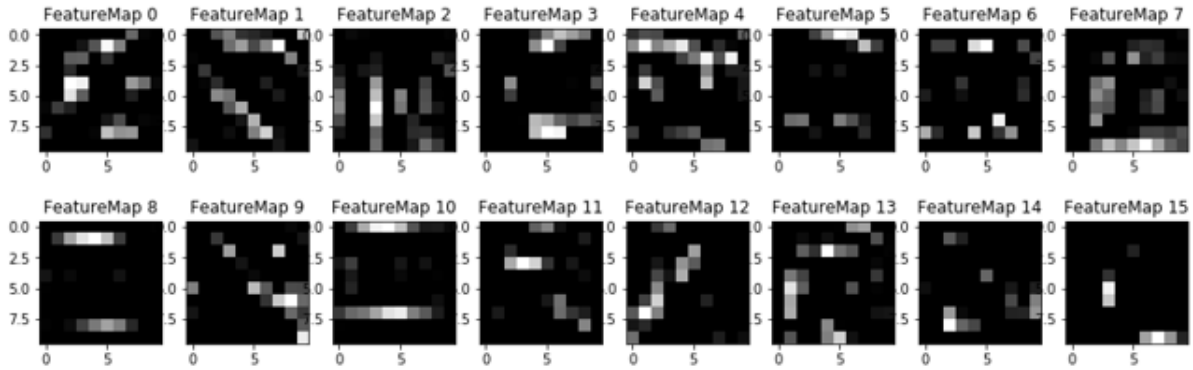
The feature maps of the conv1:

```
INFO:tensorflow:Restoring parameters from .\lenet
conv1
```



The feature maps of the conv2:

```
INFO:tensorflow:Restoring parameters from .\lenet
conv2
```



We can see that the first convolutional layer could capture the shape of the sign, whether the sign itself or the contrast of the sign.

Conclusion and some discussion

To conclude, lower learning rates and more epochs always lead to better training results, but the tendency is not always significant. When the learning rate is already pretty low, keeping reduce the learning rate just increases the validation accuracy a little, whereas much more epochs and computing effort are needed. Therefore, it is important to balance the computing effort and the validation accuracy when choosing the learning rate.

Transferring rgb images to grayscale images and simply normalizing pixels' values provide some improvement to the network training result, raising the validation accuracy from about 0.89 to 0.910. Besides, since training results are always overfitting, drop-out method shows its powerful magic to training results, raising the validation accuracy from 0.91 to 0.956.

Balancing the distribution of training data is a practicable way to reduce overfitting and increasing accuracy. The method rises the validation accuracy from 0.956 to 0.958. When choosing methods for augmentation to these additional images, the relative amounts of additional images and dataset, as well as the similarity of the whole dataset, are needed to be considered.

After implement all methods mentioned above, although the training accuracy gets closer to valid accuracy, training results are still overfitting. I will keep improving it when I learn more methods in the following study.