Project # 2: Traffic Sign Classifier Project

Ximing Chen

July 26, 2018

In this project, we aim to classify the traffic signs. The goals and steps of this project are the following: (i) make a pipeline that classifies the traffic signs in a RGB image, and (ii) reflect on the work in a written report.

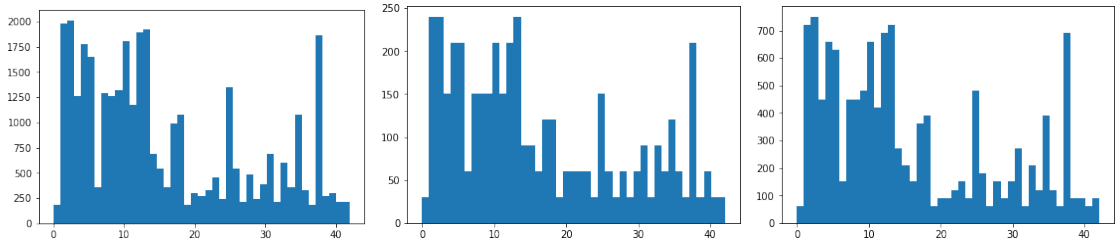
1 Pipeline

Our pipeline consisted of steps, described in details as follows:

1.1 Data Preparation and Visualization

We load image from the German Traffic Sign Benchmark data set. In particular, we separate the data into 3 sets, training, validation and test sets. Each contains 34799, 4419 and 12630 RGB images, respectively.

Moreover, we visualize the distribution of classes labels by plotting the histograms:



(a) (b) (c)

Figure 1: (a), (b), and (c) shows the histogram plot of training, validation and testing labels, respectively. Note that we use 43 (i.e., the total number of c) bins.

As shown in the above figure, we have significantly more entries corresponding to first 20 classes of images in all three data sets. According to the description provided by 'signnames.csv', most of these classes correspond to: (i) speed limits and control (e.g., speed-limit, No-passing, yield), (ii) direction control (e.g., No-passing, no-entry). This matches with common knowledge. Since the distribution of the image labels are roughly similar in these three

data sets, we may expect that classifiers that fits training data well may perform similarly in training and testing data.

1.2 Data Pre-processing

We performed two pre-processing methods. In the first method, we normalize each of the RGB by considering $\hat{X} = \frac{X-128}{128}$, where $X \in \{R, G, B\}$ are the RGB channels of image. In the second method, we transform the RGB image to YUV images and perform normalization therein. More specifically, for the Y channel, we sustract each value by 128 followed by dividing them by 128. For other channels, we simply divide them by 128.

1.3 Neural Network Architecture

We then defined two neural network based classifiers. The first one is a LeNet with appropriate input. The second one is a ConvNet consisting of 3 stages: (i) a convolutional layer with 108 (or 38) filters, (ii) a subsampling (max pooling) layer which down samples the image by 2, (iii) two fully connected layers with the first one has 100 hidden units whereas the second one has 43 units. All activation functions in the ConvNet are tanh functions.

More specifically, the architecture of our network are provided as follows:

1.3.1 LeNet

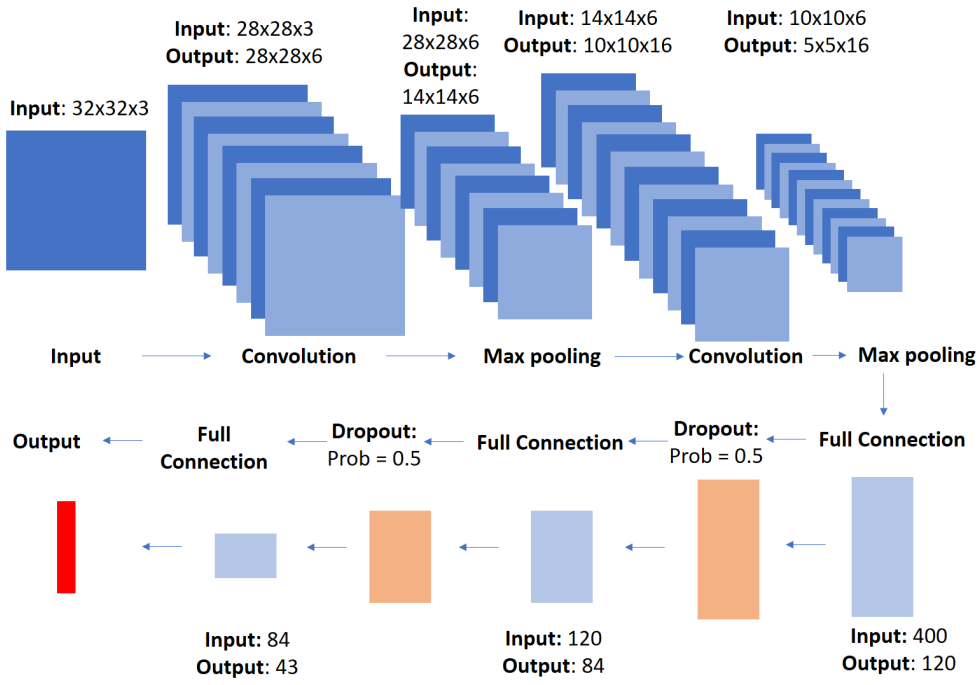


Figure 2: Illustration of architecture of LeNet in this project.

Thus, in the LeNet, we have in total 9 layers. The sequence of layers and their input-output dimensions are demonstrated in Figure 2. In particular, all activation functions here are *rectified linear* functions.

Different from the original LeNet, we implemented two drop-out layers in this project. The reason is that we observe an overfitting phenomenon in the training set. If LeNet is directly applied, the training error goes to 0 in around 100 epochs. As a result, we introduced two drop-out layers to prevent overfitting.

1.3.2 ConvNet

The other neural network is implemented as follows: Different from the LeNet architecture,

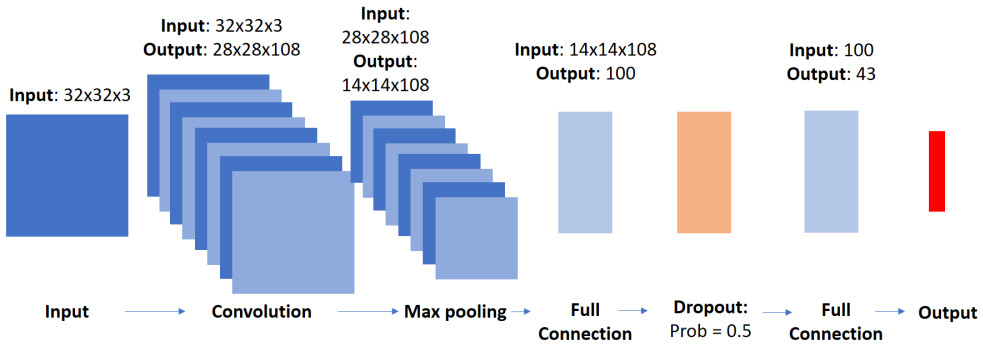


Figure 3: Illustration of architecture of single-convnet in this project.

all activation functions are $\tanh(\cdot)$ functions.

1.4 Setting Hyper-parameters

We set the epochs e, batch-size B and the learning rate η of the supervised learning pipeline and train the neural network and verify the accuracy on validation data set.

In particular, we choose the parameters according to the following general high-level scheme: we set all other parameters fixed and tune one of them, then we adopt the parameter with highest validation accuracy.

We first set epoch $e = 200$ and vary the other parameters, notice that we set epoch to large values to guarantee convergence of error. Thus, we can obtain a set of hyperparameters with relatively good performance. In this project, we adopt the following parameters:

- Epoch: 120
- Batch size: 150
- Learning rate: 0.2
- Optimizer: AdagradOptimizer

With the above parameters, the LeNet architecture and the YUV-normalized training data, we obtain a model that achieve $\geq 93\%$ accuracy.

1.5 Discussion on the parameter tuning phase

Overall, we have tried the following (in general, we have tried more parameter settings, however, we select a few notable changes):

1. Epoch: 50, batch size: 50, learning rate: 0.01 with LeNet (no dropout) and normalized data, GradientDescentOptimizer, this results in 74% accuracy;

2. Epoch: 100, batch size: 128, learning rate: 0.1 with LeNet (no dropout)/ConvNet and normalized data, GradientDescentOptimizer, this results in around 89% accuracy;

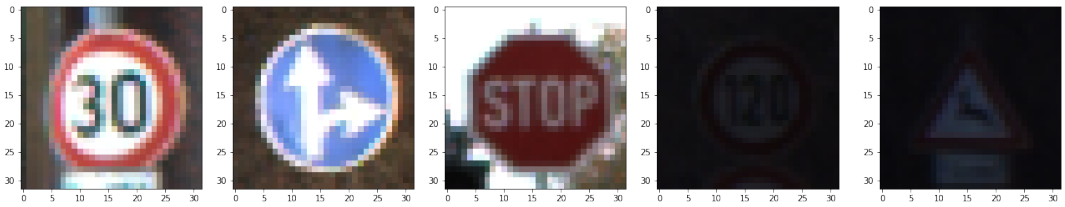
3. Epoch: 100, batch size: 128, learning rate: 0.1 with LeNet and YUV-normalized data and GradientDescentOptimizer, which results in around 92% accuracy (last submission);

4. Epoch: 120, batch size: 150, learning rate: 0.2, with LeNet with dropout and YUV-normalized data, and AdagradOptimizer, results in 93% accuracy (current manuscript method);

The most significant step we adopted is to introduce dropout layers in the LeNet. Such change is motivated by an observation on the training error. Essentially, we observe that the training error is very close to 0 whereas the validation error stuck on around 10% margin. Therefore, it indicates that the learned model overfits the training data.

1.6 Testing

We test the trained network using testing data from German traffic sign data. We notice that the images are of different sizes. To fit into our program, we resize the image to (32, 32, 3) using opencv.resize. The images are shown in the figure below:



(a) (b) (c) (d) (e)

Figure 4: Test images.

The first 3 images are rather easy to classify whereas the last two are difficult due to two reasons. First of all, images are down-sampled to (32, 32, 3), therefore certain information are lost during downsampling process. Secondly, the images are taken under poor lighting conditions. However, our classifier classifies all 5 images successfully.

2 Observations

Throughout this project, we observed the following phenomena.

- Normalization of data set is crucial. Without normalization, our LeNet has only around 70% accuracy on the validation data set, whereas 90% accuracy is achieved when data is normalized.
- The hyper-parameters of the neural network are: epochs, batch size and learning rate. From our experiment, the learning rate controls how fast the accuracy becomes approximately stationary. We observed that when we set the learning rate to be around 0.1 the algorithm converges in around 50 epochs. The batch size is set to be 128.

2.1 Shortcomings

There are a few short comings listed as follows:

- We have tested the hyper parameters through repetitive experiments. However, there is no general guideline on how to choose them and why they result in different accuracy.
- The architecture of neural network governs the performance of the classifier. However, we are unable to create a network with accuracy as high as introduced in Lecun's paper.

2.2 Possible Improvements

- As discussed previously, it is important to consider the architecture of the neural network. However, it is non-intuitive on how to choose the size of filters, and how to choose the activation functions. By visualizing the outputs of each layers, we may be able to analyze what features are valued most during the training process and whether they matches with our intuition. As a result, by leveraging these information, we may be able to redefine the network architecture hence improve the accuracy.