Project 3: Traffic Sign Classification - Jack Wetherell

1 Introduction

The goals of this project were the following:

- Load a provided dataset of labelled German traffic signs.
- Explore, summarise and visualise the dataset.
- Design, train and test a deep-learning model architecture in order to classify the traffic signs.
- Use the model to make predictions on new images, and analyse the softmax probabilities of the new images.

The project notebook (`Traffic_Sign_Classifier.ipynb`) contains the source code of this implementation, as well as the results. In addition there are also html and pdf copies of this notebook available. In the next sections I will cover how each of the rubric points have been addressed in this work.

2 Data Set Summary and Exploration

I used the python standard library module `pickle` to load the provided dataset into memory. This was pre-split into a training, validation and test set. I used the `numpy` library to calculate summary statistics of the traffic signs data set:

- Number of training examples = 34799
- Number of validation examples = 4410
- Number of testing examples = 12630
- Image data shape = (32, 32) each with 3 color channels
- Number of classes = 43

I used `pandas` to import the csv if class labels and used this to creat a dictionary object to convert between class index (int) and class label (string). Next I visualised 25 random training examples, plotting the image along with the class label in each case (Figure 1a). Next, I plotted the distributions of each class in the train, validation and test sets respectively (Figure 1b) . In order to ensure the train, validation, test split was done correctly, and without any bias, I ensured that the relative count of each of the classes fell on a straight line, to ensure no occurrence bias is present in the splitting (Figure 1c).

Figure 1: Data set summary and exploration. Panel a shows 25 random images and respective class labels from the training set. Panel b shows the distributions of the 43 classes within each of the train (blue), validation (yellow) and test (green) sets. Panel 3 shows the plots of the counts of each of the 43 classes for the test, train and validation sets relative to each other, showing they are correctly proportioned.

3 Design and Test a Model Architecture

During my model tuning I used several preprocessing techniques.

1. I normalised the images such that the values ranged from 0 to 1 rather than the original 0 to 255. This is because this well aligns the inputs with the sale of the randomly initialised normally distributed weights of the CNN.

2. I ensured the data was centred around a mean of zero, insuring that it would result in a statistically well behaved training process. I used `numpy` to perform this shift of the mean to zero

3. I also shuffled the images, to ensure the order they are presented in training does not result in any systematic bias. This was performed using `sklearn`.

I also attempted gray scaling and different color spaces. I used 'textttopencv to perform these, but I found no significant improvement the the model performance in the limit of large number of EPOCHS, it only had noticeable effects early in the training. Therefore no color modification took place in the final training.

The final model architecture was a modified LeNet model. The architecture consists of the following layers:

1. Input: shape = (32, 32, 3)

2. Convolution layer: output shape = (28, 28, 6), activation = relu

3. Pooling layer: output shape = (14, 14, 6)

4. Convolution layer: output shape = (10, 10, 16), activation = relu

5. Pooling layer: output shape = (5, 5, 16)

6. Flatten layer. output shape = (400)

7. Fully connected layer: output shape = (150), activation = relu

8. Fully connected layer: output shape = (100), activation = relu

9. Fully connected layer: output shape = (43)

When I first training this modified LeNet model I achieved an accuracy of 91.%. I noticed that the train performance quickly approached 100% and so showed clear signs of over fitting, in order to fix this I made several modifications:

1. Added drop-out with a keep probability of 75% to each of the fully connected layers.

2. Lowered the learning rate to 0.0009 and increased the number of epochs to 50.

The final hyper-parameters of the model are: `EPOCHS = 50, BATCH_SIZE = 150, LEARNING_RATE = 0.0009, DROP = 0.75`. The final results of the model was:

- Training Accuracy = 100.0%
- Validation Accuracy = 97.0%
- Test Accuracy = 95.2%.

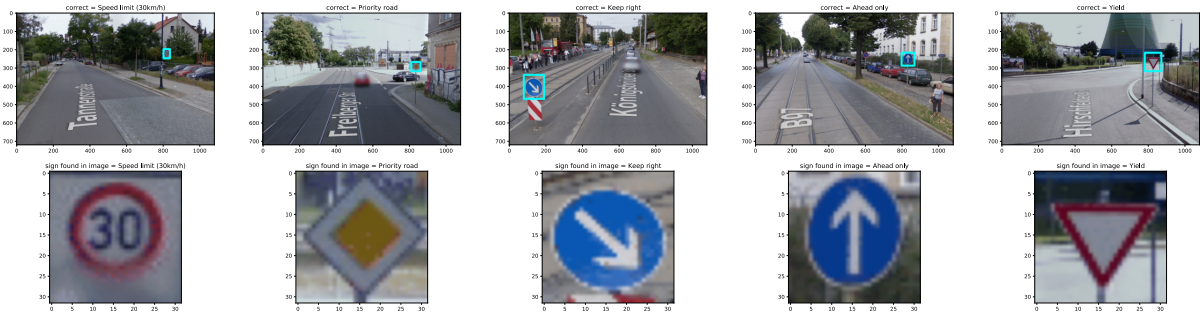

Figure 2: Results of applying the model to five signs found 'driving' around Dresden in Google Street View. All five of the signs are correctly classified.

4 Testing the model on new images

In order to test the model on some new images, I 'drove' around Dresden using Google Street view, and took some screen-shots containing street signs. I used `opencv` to pre-process the images, extracting the part of the image containing the street sign. I used the perspective transform technique from the lane line finding project. The results of this are shown in the figure 2. As can be seen it correctly categorised all five of the images: The softmax values for each of these cases are 1.0, showing that the model has a high confidence in these predictions. The calculations of these, with the details, is shown in cell 13 of the project notebook.