Traffic Sign Classifier

Seymur Dadashov

August 19, 2020

Abstract

The purpose of the project is implement a simple German traffic sign classifier using a Convolutional Neural Network (CNN) with the TensorFlow framework.

1 Data Set Summary Exploration

1.1 Basic Summary of Data Set

The following output is a simple analysis performed using python to read the incoming examples from the data set provided.

```
Number of training examples = 34799
Number of validation examples = 4410
Number of testing examples = 12630
Image data shape = (32, 32, 3)
Number of classes = 43
```

Figure 1: Information on data set examples including size, shape and class

1.2 Exploratory Visualization

The following images sample the data set to observe the type of images that are to be expected to be fed into the CNN (Figure 2) and the distribution of the number of images that belong to a specific class (Figure 3).

2 Design and Testing of Model Architecture

2.1 Data Preprocessing

The initial conversion of the RGB image to gray scale is done by taking the weighted sum of individual gamma intensities of the RGB color-space. The standard I used for the conversion was based on `Rec.601 luma` and has a nonlinear luma component (Y') that is computed as follows:

$$Y' = 0.299R' + 0.587G' + 0.114B' \tag{1}$$

The image data was then normalized to have a mean zero and equal variance using the following equation:

$$x_{norm} = \frac{x - max_x/2}{max_x/2} \tag{2}$$

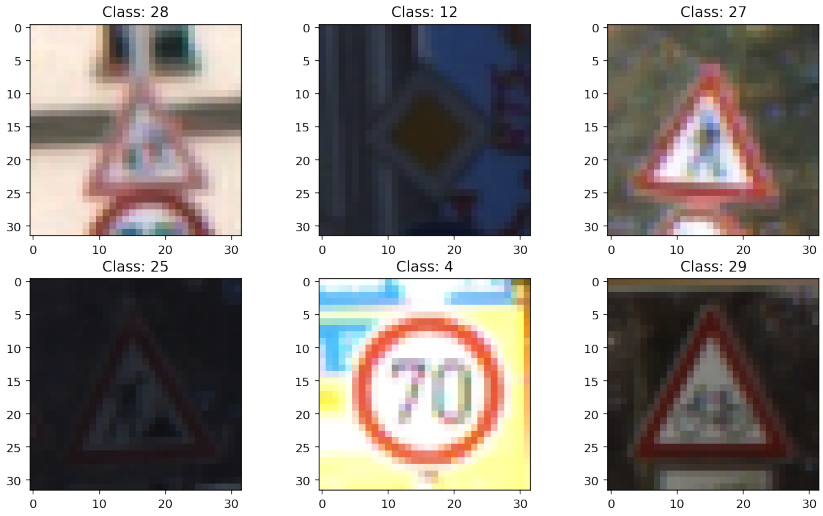

Figure 2: Subset of examples within the provided data set

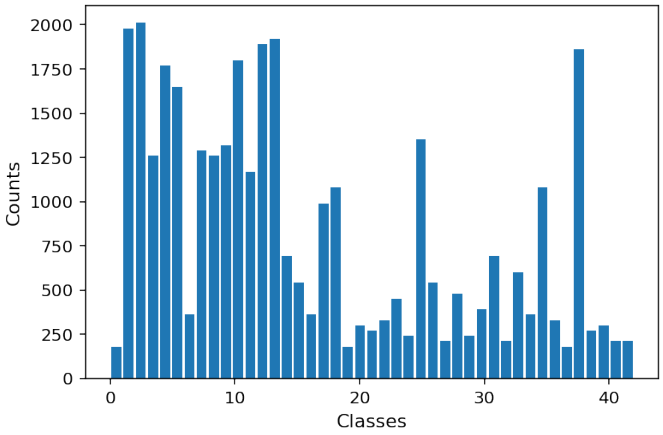

Figure 3: Distribution of images into classes

2.2 Model Architecture

The model architecture that I made was based on the `LeNet` Architecture, but slightly more built up to improve the accuracy of the model to meet the project requirement of above 93%. The model architecture is as follows and consists of two primary layers separated by a max pooling before begin flattened and going through two fully connected layers and finally providing logit scores. From previous experience a network like this that doesn't create too many output features within the CNN layers in the middle for a simple task performs well as the complexity of the input images is not very intense and thus the highest number of output features is seen to be 128 features not including the fully-connected layers. Once again from experience, decreasing the output features over several layers before reaching the desired number of classes tends to work the best from my experience for such simple classification problems.

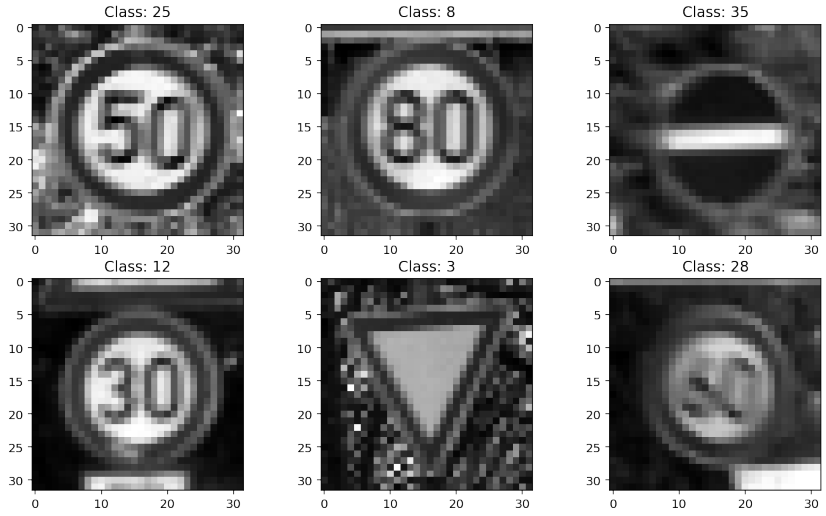

Figure 4: Gray-scale conversion results

Layer	Description
Input	32x32x1 Grayscale Image
Convolution 3x3	1x1 stride, valid padding, 30x30x32
ReLU	
Convolution 3x3	1x1 stride, valid padding, 28x28x64
ReLU	
Max Pooling 3x3	2x2 stride, 13x13x64
Convolution 3x3	1x1 stride, valid padding, 11x11x128
ReLU	
Convolution 3x3	1x1 stride, valid padding, 9x9x128
ReLU	
Max Pooling 2x2	2x2 stride, 4x4x128
Flatten	
Fully-Connected	512 output features, dropout
ReLU	
Fully-Connected	256 output features, dropout
ReLU	
Output	43 logit scores
Softmax	Probability transform from logit scores

2.3 Training the Model

Once again some of my previous experience allowed me to narrow down these parameters quickly as for classification problems with images involving very obvious geometry, I have found a set of approximate parameters that work together with my own CNN. The chosen number of epochs is 20 for the sake of redundancy, but can be reduced to about 15 for good results. The `batch_size` can be either 64 or 128, but for this problem 128 seemed to provide slightly better results. Typically I would begin with 64 and decrease or increase the hyper-parameter accordingly after observing the behaviour and run-time of the model. The learning rate is usually set to 0.001 from my experience as a good starting point, but with GPU access being provided, I decided I can take smaller steps to achieve a confident result without having to wait for too long, thus I went with 0.0002.

2.4 Approach to Successful Solution

The approach taken for achieving a result about 0.93 revolved around setting up the correct CNN architecture primarily and testing a few variations of the way the model should be trained. Figure 5 shows the how model quickly broke over the required threshold in only 4 epochs and reached a peak of 0.977 and began to see no further improvement after that. Depending on the random variable initialization, results above 0.98 were observed as well. Figures 6 and 7 show the Training Loss and Training Accuracy as a function of epoch cycles.

```
Training ...

Epoch 1, val acc 0.742 Training loss 1.215, val loss 1.577
Epoch 2, val acc 0.908 Training loss 0.324, val loss 0.522
Epoch 3, val acc 0.919 Training loss 0.266, val loss 0.286
Epoch 4, val acc 0.940 Training loss 0.087, val loss 0.115
Epoch 5, val acc 0.944 Training loss 0.106, val loss 0.182
Epoch 6, val acc 0.952 Training loss 0.048, val loss 0.090
Epoch 7, val acc 0.956 Training loss 0.208, val loss 0.092
Epoch 8, val acc 0.956 Training loss 0.011, val loss 0.059
Epoch 9, val acc 0.960 Training loss 0.015, val loss 0.100
Epoch 10, val acc 0.958 Training loss 0.013, val loss 0.014
Epoch 11, val acc 0.963 Training loss 0.007, val loss 0.067
Epoch 12, val acc 0.964 Training loss 0.005, val loss 0.065
Epoch 13, val acc 0.973 Training loss 0.001, val loss 0.025
Epoch 14, val acc 0.971 Training loss 0.023, val loss 0.060
Epoch 15, val acc 0.968 Training loss 0.001, val loss 0.055
Epoch 16, val acc 0.977 Training loss 0.091, val loss 0.063
Epoch 17, val acc 0.970 Training loss 0.002, val loss 0.033
Epoch 18, val acc 0.972 Training loss 0.002, val loss 0.174
Epoch 19, val acc 0.976 Training loss 0.033, val loss 0.003
Epoch 20, val acc 0.975 Training loss 0.005, val loss 0.080
Model saved with accuracy 0.977
```

Figure 5: Training data tracked by epochs

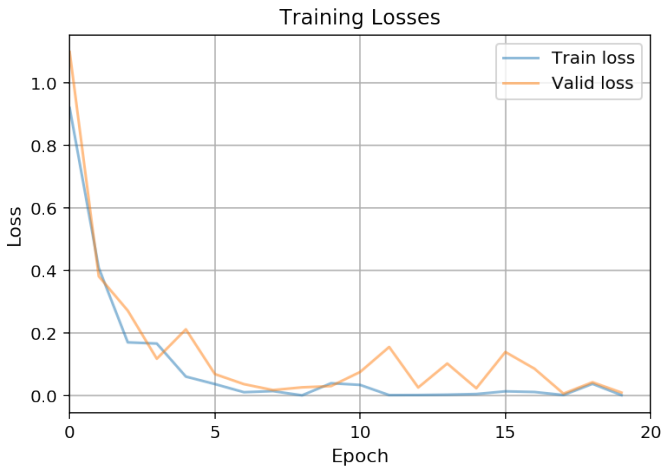

Figure 6: Training loss as a function of epoch cycles

Once again a 'transfer learning' (not true to the meaning of transfer learning as I did not use the same architecture or weights of the original LeNet model, simply used the LeNet architecture as reference) method was used as I had recycled the `LeNet` architecture, but added a few more layers to improve its performance on slightly more complex images as had written digits were a lot simpler in comparison to German Traffic Signs. Therefore, a slightly deeper network with slightly larger output features would be able to capture

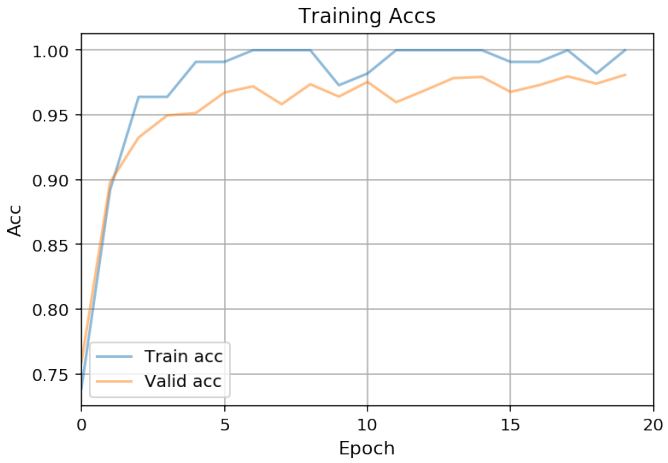

Figure 7: Training accuracy as a function of epoch cycles

these differences between the models. Additionally, an important aspect of the model in my opinion is the dropout layer that allows for the model to not rely on any single node and therefore makes the model overall robust.

3 New Images: Test of Robustness

The following new images were chosen to be classified based on a quick Google search. Figure 8 shows nine new traffic sign images that fall into one of the 43 classes that the model is capable of identifying. The outcome of the classification can be seen in Figures 9-13. Some interesting things can be noted is the misclassification of the image found in Figure 12. The classifier believes with high confidence that the sign is a Speed Limit of 30km/h, but in reality the sign should be 'End of all speed and passing limits'. The result of the misclassification can be attributed to the fact that the sign itself has a diagonal line drawn through the actual sign and thus is alerting alerting some of the features in the model that it bears a close resemblance to the Speed Limit sign rather than its real sign. This tells us that the model is not as robust as we believe it to be when it comes to generalizing to images that are not found in the training, test, or validation set as it has clearly not come across this type of image. Additionally, the image in Figure 13, although classified correctly does appear to be confident in its decision. Although the image is not of poor quality, I believe its lack in confidence is due to the new lighting conditions found in the image. Perhaps that images provided in the data set originally used for the model did not have this particular angle or lighting conditions and thus to the model this image would appear to be similar to another sign. Luckily, in this case it was able to classify it correctly, but this is not something that can be overlooked if we would like to implement this model on a real traffic sign classifier. Therefore, it would be nice to generate more data through artificially modifying the image by rotation, distortion, cropping, adjusting brightness/contrast in order to make our model more robust to such variations. Lastly, the feature map visualisation can be seen in Figure 14 that corresponds to the first convolution layer of our CNN as it discovers certain characteristics about the sign.

Figure 8: New test images for testing

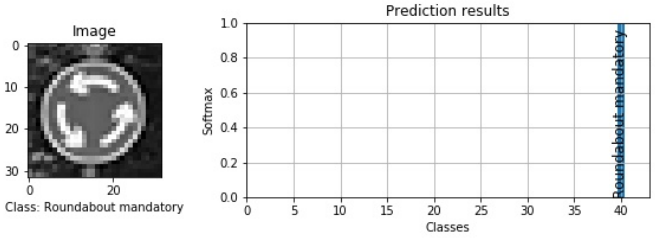

Figure 9: Roundabout test image

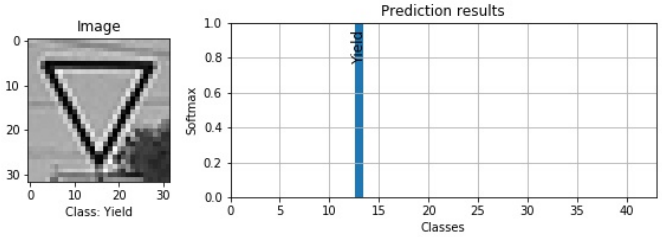

Figure 10: Yield test image

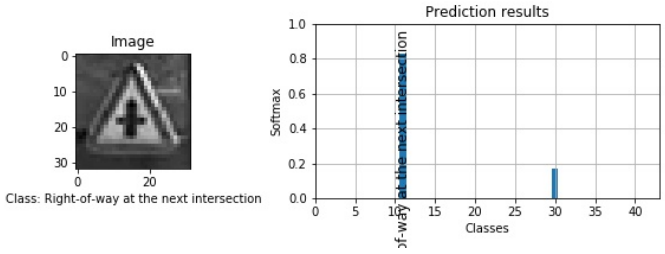

Figure 11: Right-of-way at the next intersection test image

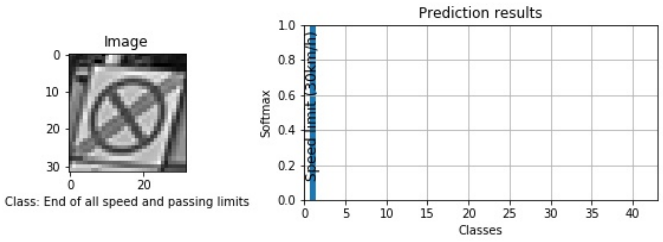

Figure 12: Misclassified image

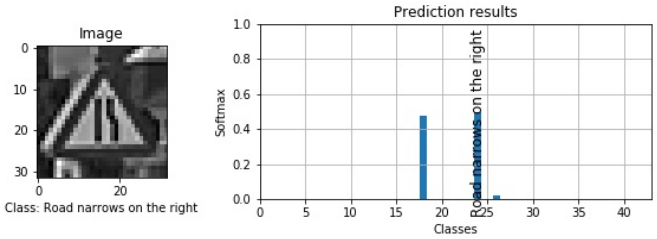

Figure 13: Not confident classification

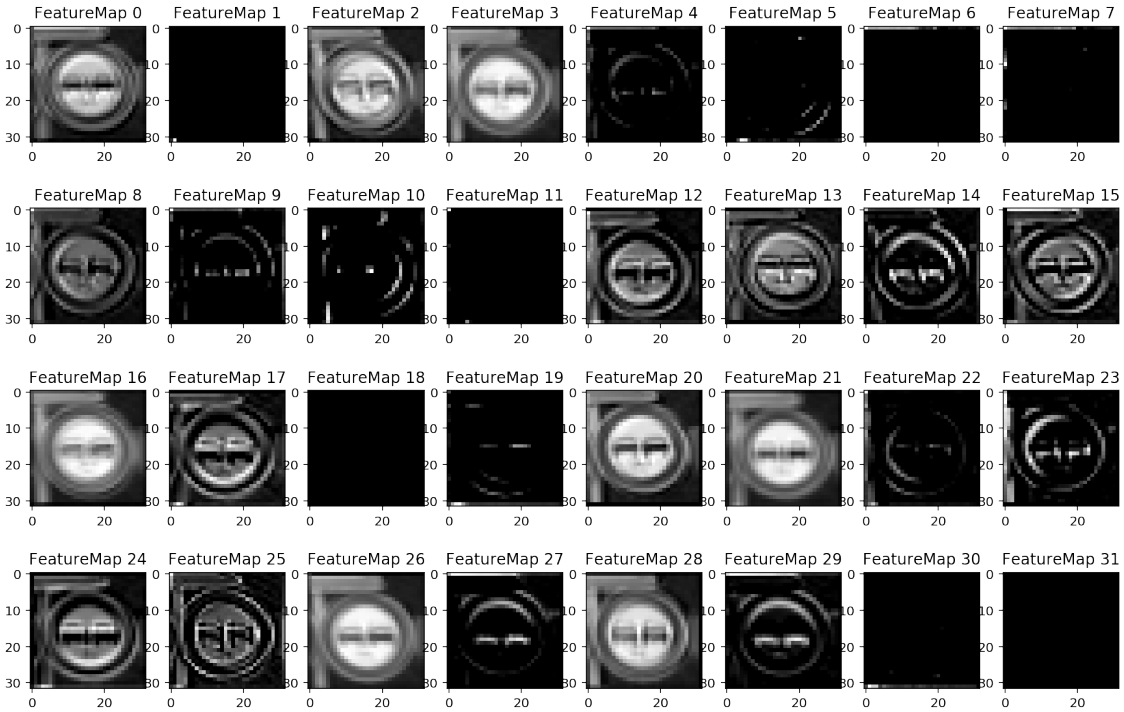

Figure 14: Feature map visualization