Udacity Self-Driving Car Nanodegree
Project 2: Traffic Sign Classifier

Mark Helms, November 11, 2017

Goal/Summary

Make a traffic sign classifier for a select number of German roadway signs, using primarily Python, Tensorflow and NumPy. Peak accuracies achieved were as followed: training=99.0%, validation=99.2%, **test=97.4%**.

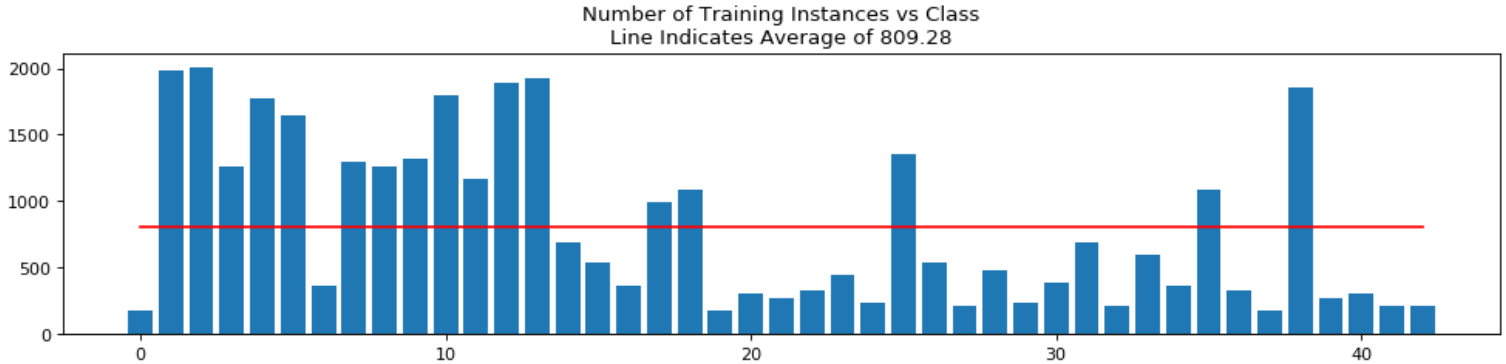
Approach

Step 1: Read and Analyze Files

- The size of the training set is 34799 images.
- The size of the validation set is 4410
- The size of test set is 12630
- The shape of a traffic sign image is 32x32 and has 3 channels of information (R,G,B)
- The number of unique classes/labels in the data set is 43

Some examples of the dataset are shown below (all classes can be seen in the iPython notebook):

The distribution of images in the training set is shown below:



Step 2: Preprocessing Steps

- Augmentation

  Samples of the original images are randomly contorted and appended to the list of images. The contortions performed include: rotation by up to 15 degrees, scaling up to 10%, stretching along the x-axis up to 10%, shearing up to 10 deg, shifting up to 2 pixels, and an occlusion rate around 5% in blocks of 4x4. Augmentation is performed so that 20x the original dataset size is added to the dataset. The resulting relative distribution of examples is the same, which is valid, because it also represents the validation distribution, indicating that the natural occurrence (i.e. base rate of incidence) of these images is already well-represented and any rebalancing would require weighting during training.
- Cropping

  The center of the images is extracted by cutting off border edges. Images are resized to 26x26.
- White Balancing

  For each image, the colors are expanded so that they range from 0 to 255. The center of the image is used to calculate what the scaling and shifting should be.
- Normalization

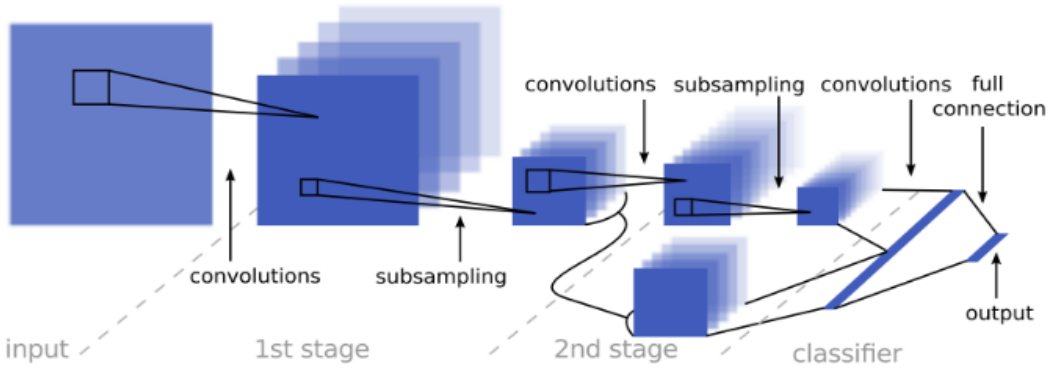
  All images are converted to float and shifted so that they have a mean of 0 and a maximum deviation of +/- 1.

Example images after data augmentation are shown below:

Step 3: Model Architecture

The model architecture chosen is shown below and is based on the architecture presented in [1]. It consists of two convolutional layers and two fully connected layers. Each convolutional layer is followed by a ReLU activation layer and a maxpooling layer. Each fully connected layer is followed by a ReLU layer. The output from the first convolutional layer is put through two maxpool operations and then merged with the output from the maxpool operation from the second convolutional layer before being flattened for processing by the fully connected layers.

The architecture from [1] is shown below:



The following table lists parameters for each layer, which were discovered by experimentation:

Layer Name	Size	Stride
Input	32x32x3	
1st Stage Convolution + ReLU	6x6x43	1
1st Stage Maxpooling	2x2x43	2x2
2nd Stage Convolution + ReLU	3x3x64	1
2nd Stage Maxpooling	2x2x64	2x2
Flatten*	2224	
1st Fully Connected + ReLU*	100	
2nd Fully Connected + ReLU	43	

Layers with an asterisk were followed by a dropout layer with a probability of keeping of 0.85. For all randomly initialized weights and biases, a mean of 0 and standard deviation of 0.15 were used.

Discussion

The original LeNet architecture provided was able to produce an accuracy above 90%. The LeNet network was intended to identify digits in grayscale photos. Because the images for this project are RGB and because the number of classes possible to predict is 43, it was necessary to expand the network depth-wise from beginning to end. With these changes alone, it's possible to achieve accuracies in the mid-90's. In order to approach 97%, the architecture from [1] was used, as recommended by the project notes. The basic approach with both architectures was to keep the framework as similar as possible to the proven model, while changing the depth of layers and the sizes of kernels. The convolutional layers provide locations of features that have been deemed important by the network. The flattening stage is preparation for the fully connected layers, which make decisions based upon which features the convolutional networks detected. Activation layers for these networks allow for non-linear detection and decision making. The dropout layers enforce a well-rounded network that won't overtrain or over-simplify the solution based on training data.

Step 4: Training and Testing

For training:

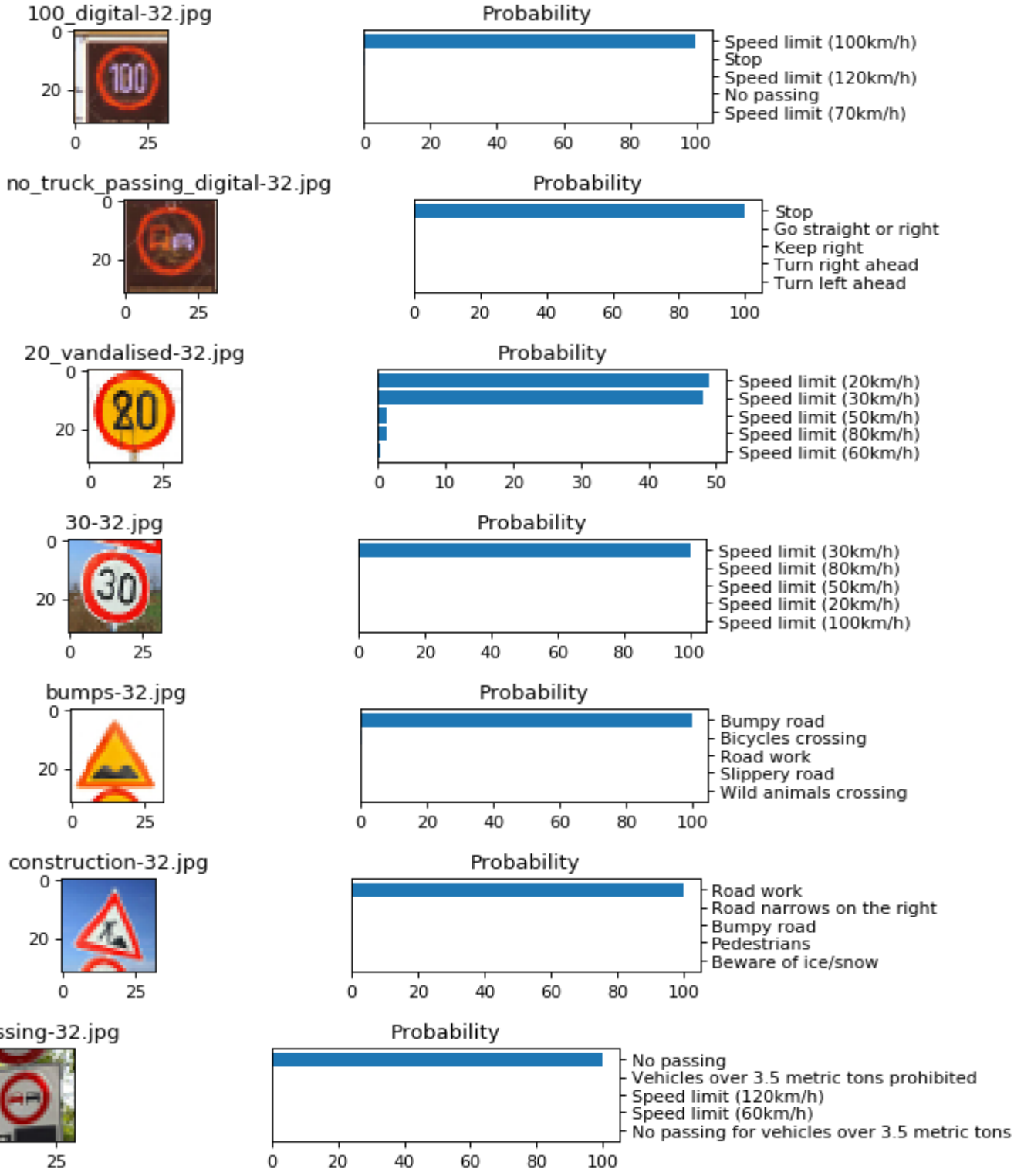
- The number of epochs used was 25
- The batch size was 4096
- Cost was calculated as the Mean of the Softmax of the Cross Entropy
- The Adam Optimizer was used with a learning rate of 0.001

```
EPOCH 1 ... Train. Acc. = 0.58939 Val. Acc. = 0.752
EPOCH 2 ... Train. Acc. = 0.78021 Val. Acc. = 0.883
EPOCH 3 ... Train. Acc. = 0.84874 Val. Acc. = 0.919
EPOCH 4 ... Train. Acc. = 0.89368 Val. Acc. = 0.946
EPOCH 5 ... Train. Acc. = 0.91956 Val. Acc. = 0.968
EPOCH 6 ... Train. Acc. = 0.93309 Val. Acc. = 0.971
EPOCH 7 ... Train. Acc. = 0.94362 Val. Acc. = 0.977
EPOCH 8 ... Train. Acc. = 0.95396 Val. Acc. = 0.985
EPOCH 9 ... Train. Acc. = 0.96006 Val. Acc. = 0.985
EPOCH 10 ... Train. Acc. = 0.96519 Val. Acc. = 0.988
EPOCH 11 ... Train. Acc. = 0.96951 Val. Acc. = 0.988
EPOCH 12 ... Train. Acc. = 0.97218 Val. Acc. = 0.990
EPOCH 13 ... Train. Acc. = 0.97453 Val. Acc. = 0.990
EPOCH 14 ... Train. Acc. = 0.97748 Val. Acc. = 0.991
EPOCH 15 ... Train. Acc. = 0.97955 Val. Acc. = 0.992
EPOCH 16 ... Train. Acc. = 0.98104 Val. Acc. = 0.990
EPOCH 17 ... Train. Acc. = 0.98225 Val. Acc. = 0.992
EPOCH 18 ... Train. Acc. = 0.98371 Val. Acc. = 0.989
EPOCH 19 ... Train. Acc. = 0.98478 Val. Acc. = 0.991
EPOCH 20 ... Train. Acc. = 0.98606 Val. Acc. = 0.992
EPOCH 21 ... Train. Acc. = 0.98718 Val. Acc. = 0.990
EPOCH 22 ... Train. Acc. = 0.98765 Val. Acc. = 0.989
EPOCH 23 ... Train. Acc. = 0.98862 Val. Acc. = 0.990
EPOCH 24 ... Train. Acc. = 0.98931 Val. Acc. = 0.991
EPOCH 25 ... Train. Acc. = 0.98998 Val. Acc. = 0.991
```

Testing on the test set resulted in an accuracy of **97.4%**.

Step 5. Training on Extra Images

The extra images found on the web and their corresponding classification are shown below:

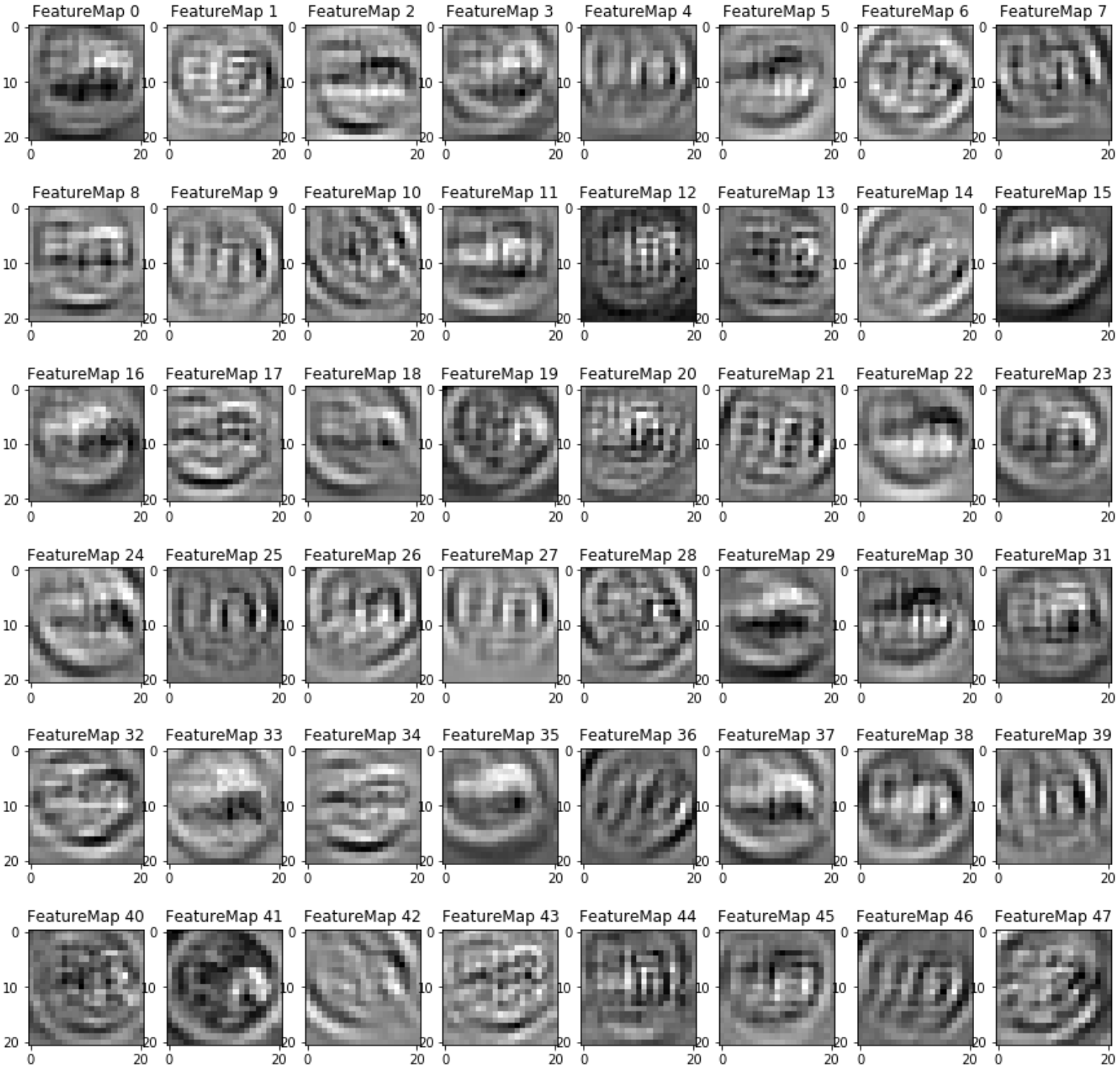


The first three images are somewhat unusual compared to the training data. The first two images have a black background, because they are from a digital display. The third image has been spray-painted to make the 20 kph limit look like 80 kph, which confuses the network substantially. The other images may be rotated or tilted slightly, but are generally clean images.

All but the second image were correctly identified, resulting in an accuracy of about 85%. Due to the low number of images tested, this accuracy cannot be properly compared with the training or validation accuracies, but the tests do show promise of a well-trained network.

Step 6. Visualizing the Network

The network was passed in the last test image from Step 5 above, “no_truck_passing_digital-32.jpg”. The output from the first convolutional layer appears as shown below. Rotations and edge detections can be easily seen, among other complicated activations.



References

1. Sermanet and Lecun. “Traffic Sign Recognition with Multi-Scale Convolution Networks,” Aug, 2011.