# Project 2 - Traffic Sign Recognition  Denise R. James

*Two neural networks are analyzed in the traffic sign classification project. A linear neural network allowed 87% training accuracy with little CPU time. This linear neural network followed the process of the TensorFlow lab in the lectures. A second notebook attempts to classify with convolution neural network, CNN, based on an Hvass tutorial . I ran into CPU memory issues. The kernel stops at 30% accuracy. I plan to run these CNN notebook on AWS when an appropriate AMI is available.*

1. Describe the techniques used to preprocess the data.

The main three techniques required to setup up training and testing of the German Traffic Signs data are normalizing all of the data, hot-one encoding all of the labels and flattening all of the features. Since I was restricted to Docker Toolbox that does not have cv2, skimage tools, I choose to flatten the features. There is inherit loss of accuracy due to all three of these preprocess steps. I also defined tensorflow place holders.

The names of the signs were extracted from the provided signnames.csv file. The 43 traffic signs in the German Traffic Sign in test.p data file are:

| Label Number | Name of the Traffic Sign |
|---|---|
| 1 | Speed limit (20km/h) |
| 2 | Speed limit (30km/h) |
| 3 | Speed limit (50km/h) |
| 4 | Speed limit (60km/h) |
| 5 | Speed limit (70km/h) |
| 6 | Speed limit (80km/h) |
| 7 | End of speed limit (80km/h) |
| 8 | Speed limit (100km/h) |
| 9 | Speed limit (120km/h) |
| 10 | No passing |
| 11 | No passing for vehicles over 3.5 metric tons |
| 12 | Right-of-way at the next intersection |
| 13 | Priority road |
| 14 | Yield |
| 15 | Stop |
| 16 | No vehicles |
| 17 | Vehicles over 3.5 metric tons prohibited |
| 18 | No entry |
| 19 | General caution |
| 20 | Dangerous curve to the left |
| 21 | Dangerous curve to the right |

| | |
|---|---|
| 22 | Double curve |
| 23 | Bumpy road |
| 24 | Slippery road |
| 25 | Road narrows on the right |
| 26 | Road work |
| 27 | Traffic signals |
| 28 | Pedestrians |
| 29 | Children crossing |
| 30 | Bicycles crossing |
| 31 | Beware of ice/snow |
| 32 | Wild animals crossing |
| 33 | End of all speed and passing limits |
| 34 | Turn right ahead |
| 35 | Turn left ahead |
| 36 | Ahead only |
| 37 | Go straight or right |
| 38 | Go straight or left |
| 39 | Keep right |
| 40 | Keep left |
| 41 | Roundabout mandatory |
| 42 | End of no passing |
| 43 | End of no passing by vehicles over 3.5 metric tons |

2. Describe how you set up the training, validation and testing data for your model. If you generated additional data, why?

The pickled data dictionary was used to read in the German traffic sign dataset the following attributes.

1. Image pixel values and color, (32 pixels , 32 pixels, 3 channels)

2. Labels - the label of the traffic sign

3. Features, the image pixel values.

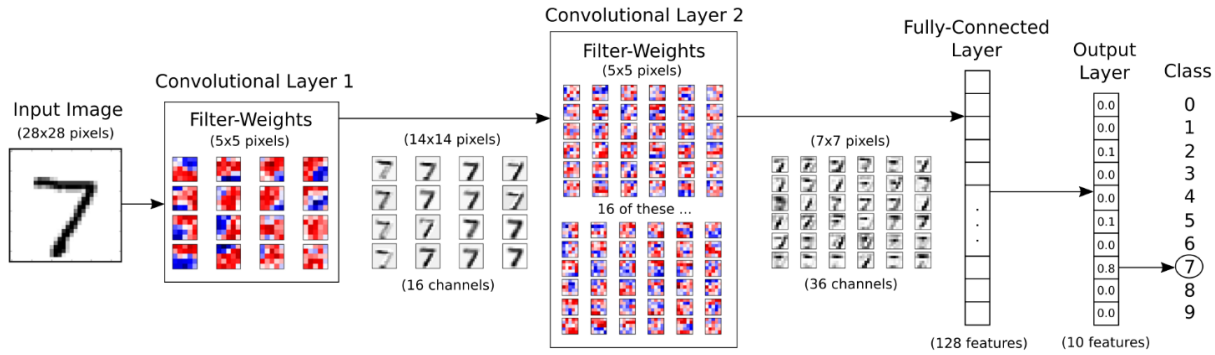
4. Size feature was used to plot the images and sampling areas.

The training and test data is initially extracted from the German traffic signs dataset. Validation data is generated by duplicating 5% of the training data.

3. What does your final architecture look like? (Type of model, layers, sizes, connectivity, etc.)

The final architecture of the linear model similar to the TensorFlow lab begins with the Linear Function or logits ,WX + b. The softmax is estimated the prediction of the traffic sign. Cross Entropy and loss are calculated. Loss is the the significant parameter as this calculated value is used to predict the accuracy.

The CNN model is based on the following diagram except color pictures are used, 32X32 pixel




The steps in the CNN are

| Layer Name | Description |
|---|---|
| conv1 | convolution and rectified linear activation. |
| pool1 | max pooling. |
| norm1 | local response normalization. |
| conv2 | convolution and rectified linear activation. |
| norm2 | local response normalization. |
| pool2 | max pooling. |
| local3 | fully connected layer with rectified linear activation. |
| local4 | fully connected layer with rectified linear activation. |
| softmax_linear | linear transformation to produce logits. |

4. How did you train your model? (Type of optimizer, batch size, epochs, hyperparameters, etc.)

For the linear neural model, experimented and decided upon learning rate of epoch from a fellow cohort of epoch of 100. I assumed 10 was more than enough. The forums and slack channels are vital in my success of completing project 2.

5. What approach did you take in coming up with a solution to this problem? I Google examples of linear neural networks, watched the Udacity lectures over, camped out in the P2 traffic signs slack

channel and learned from my amazing cohorts. I got much value from http://www.hvass-labs.org/ tutorials. He states all of this work is done by one person.

6. Choose five candidate images of traffic signs and provide them in the report. Are there any particular qualities of the image(s) that might make classification difficult? It would be helpful to plot the images in the notebook.

I choose five German traffic signs from the web. All five are plotted in the linear neural notebook. It appears one of my images was not detect. Perhaps the red stop sign and the red do not enter sign are too similar.

7. Is your model able to perform equally well on captured pictures or a live camera stream when compared to testing on the dataset?

While I did not test on live camera stream, I believe the loss will go down because the pictures I downloaded from the internet are clear pictures, not jittered.

8. Use the model's softmax probabilities to visualize the certainty of its predictions, tf.nn.top_k could prove helpful here. Which predictions is the model certain of? Uncertain? If the model was incorrect in its initial prediction, does the correct prediction appear in the top k? (k should be 5 at most)

The follow return array from tf.nn.top_k is values: The k largest elements along each last dimensional slice.

```
TopKV2(values=array([
       [ 1.,  0.,  0.,  0.,  0.],
       [ 1.,  0.,  0.,  0.,  0.],
       [ 1.,  0.,  0.,  0.,  0.],
       [ 1.,  0.,  0.,  0.,  0.],
       [ 1.,  0.,  0.,  0.,  0.]], dtype=float32)
```

This second return array is the indices of values within the last dimension of input

```
       [20,  0,  1,  2,  3],
       [33,  0,  1,  2,  3],
       [37,  0,  1,  2,  3],
       [25,  0,  1,  2,  3],
       [31,  0,  1,  2,  3]], dtype=int32))
```

9. If necessary, provide documentation for how an interface was built for your model to load and classify newly-acquired images.

I used png images from the web. Since I had not access to a Docker Toolbox image that has the CV2 library I used PIL. The four images were stacked into a newdata array.

```
from PIL import Image

img=Image.open('60kmph.png')

img_rgb = img.convert('RGB')

img_rgb.thumbnail((32,32), Image.ANTIALIAS)

X_tr4 = np.reshape(img_rgb, (1, 32, 32, 3))

plt.imshow(img_rgb)
```

In conclusion Convolution Neural Networks require more CPU time but are better at training the data. I was able to get 90%  accuracy with the linear neural network.  If I had the hardware, I believe I would have above 95% training accuracy for the CNN.