Traffic Sign Classification

Samuel Navarro

June 18, 2019

1 Data Set Summary and Exploration

1. Provide a basic summary of the data set.

The training set is 34799, the validation set is 4410 and the number of testing examples are 12630.

The total number of classes are 43 and the Images have a shape of 32, 32, 3.

2. Include an exploratory visualization of the dataset

We can see that the data is unbalanced but given the obtainer accuracy, I don't think is necessary to augment the data. It could be done in the future to improve the accuracy further.

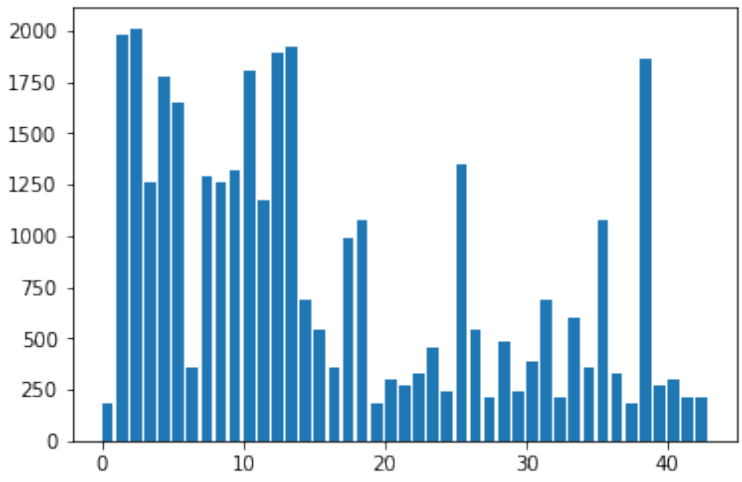

Figure 1: count

Here's a visualization of the images with the given labels:

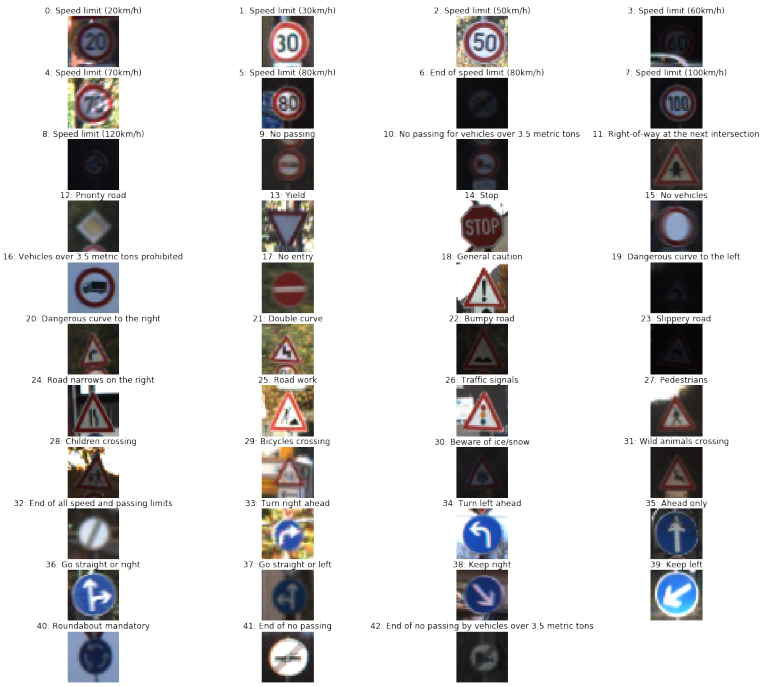

Figure 2: Images and Labels

2 Design and Test a Model Architecture

Describe how you preprocessed the image data. What techniques were chosen and why did you choose these techniques?

At the beginning I used the Function 1 to pre-process the data:

But, I don't know why it constantly gave me an `TypeError: Cannot interpret feed_dict as Tensor: The name 'save/Const:0' refers to a Tensor which does not exist ....`

That's the reason why I end up using plain `numpy` to sum up the channels to convert to greyscale and normalize the images. I think there is some problem with the sessions or the variables when using Tensorflow for pre-processing. If you can tell why that could be helpful in the future.

After review: The reason why I used the greyscale step is because the only thing that matters in the sign are the shapes. In fact, there's a lot of similarity between the colors of the signs so it was better to just work with the shapes. I normalized the data to get approximately zero mean and equal variance. Another thing is that this is better so the optimization process could converge faster.

2. Describe what your final model architecture looks like include model type, layers, layer sizes, etc.

3. Describe how you trained your model.

```
IMAGE_SIZE = 32

def tf_resize_images(images):
    X_data = []
    tf.reset_default_graph()
    X = tf.placeholder(tf.float32, (None, None, 3))
    tf_img = tf.image.per_image_standardization(X)
    tf_img = tf.image.rgb_to_grayscale(tf_img)
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        for image in images:
            output_image = sess.run(tf_img, feed_dict = {X: image})
            X_data.append(output_image)

    X_data = np.array(X_data, dtype = np.float32) # Convert to numpy
    return X_data


## And the implementation would be

X_train = tf_resize_images(X_train)
```

Listing 1: Old Pre-process function

Table 1: Model Architecture

Layer	Description
Input	32x32x1 Greyscale img
Conv 5x5	1x1 stride, VALID padding, outputs 28x28x12
Relu	
Max pool	2x2 strides, outputs 14x14x12
Conv 5x5	1x1 stride, VALID padding, outputs 10x10x32
Relu	
Max pool	2x2 strides, outputs 5x5x32
Flatten	
Dropout	keep_prob = 0.5 when training.
Fully Connected	Input: 800. Output: 256
RELU	
Dropout	keep_prob = 0.5 when training.
Fully Connected	Input. 256. Output: 128
RELU	
Dropout	keep_prob = 0.5 when training.
Fully Connected	Input: 128. Output: 43

The learning rate I used was 0.0005, the number of epochs was 30 and the `batch_size = 128`.

For the loss I used the function `tf.nn.softmax_cross_entropy_with_logits_v2` with the one hot encoded values as labels.[1]

4. Describe the approach taken for finding a solution and getting the validation set accuracy to be at least 0.93.

My validation accuracy was 0.969 and the Test accuracy was 0.958.

After review: the Process was:

- I search through different kernel sizes and depth.
- I try higher depth of the kernels to see if they can extract more information in the images but that was not the case.
- After trying different combinations without success (the accuracy on the validation set was below 70 % and it looked like there's won't be much improvement besides how much I increase the number of epochs) I ended up with the same architecture as LeNet.
- Another thing I tried, was to add more fully connected layers at the end but that was useless too.
- After I decided to stick with the LeNet architecture, I added dropout to every fully connected layer to avoid overfitting, that gave me more accuracy in the validation set confirming that I was overfitting in the train set.
- In the optimization process: At the beginning I used a very higher learning rate so I ended up with the problem mentioned in the lectures of faster convergence but to a lower loss. So I decreased the learning rate.

With the architecture, I was interested in playing with the size of the Convolutional kernels, strides, etc. At the end, it seems obvious to me that I don't have an idea of the particular things that make a particular architecture to work. This is something I will dive into in the future.

3 Test a Model on New Images

3.0.1 Choose five German traffic signs from the web

Here are the images from the web used after pre-processing:

[1] In particular, I believe it would be interesting to address using the embeddings approach mentioned in the lectures.

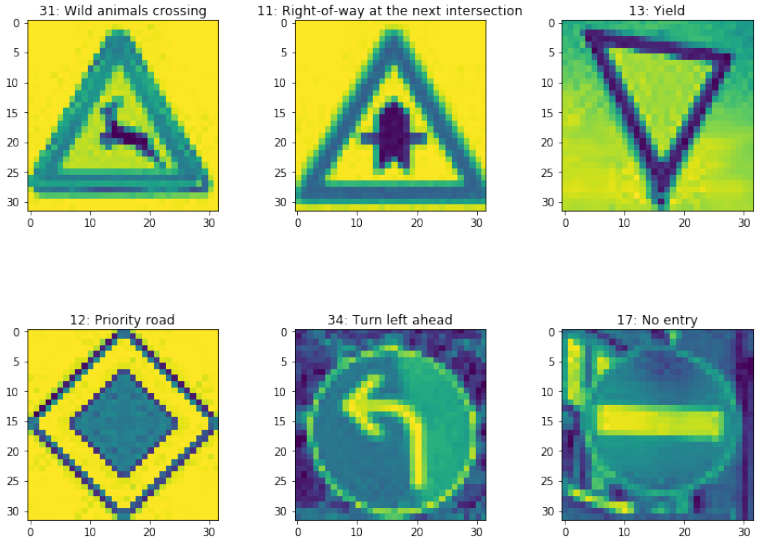

Figure 3: Web Images

Review: The test accuracy on the images was 1.00. This is mainly because this are pretty basic images. They are well centered and they have no noise on them. This made easy to my model classify the images.

In the Figure 4 we can see the top 5 Softmax Probabilities for each images with the values. This tells us that the confidence of the model is pretty high. The first column is the image with the highest probability, the second is the 2nd guess and so on. Each input image is in each row.

This is a pretty useful tool to check the similarities between the images with the highest probability.

The output from the activations are in the Figure 5.

4 Discussion

The things I would like to try in the future or some areas where I thing I can improve the implementation of this model are:

- Augment the data.
- Use tensorboard.
- **Try to understand better why some particular architectures work**
- Use the embeddings approach instead of one hot encoded labels.

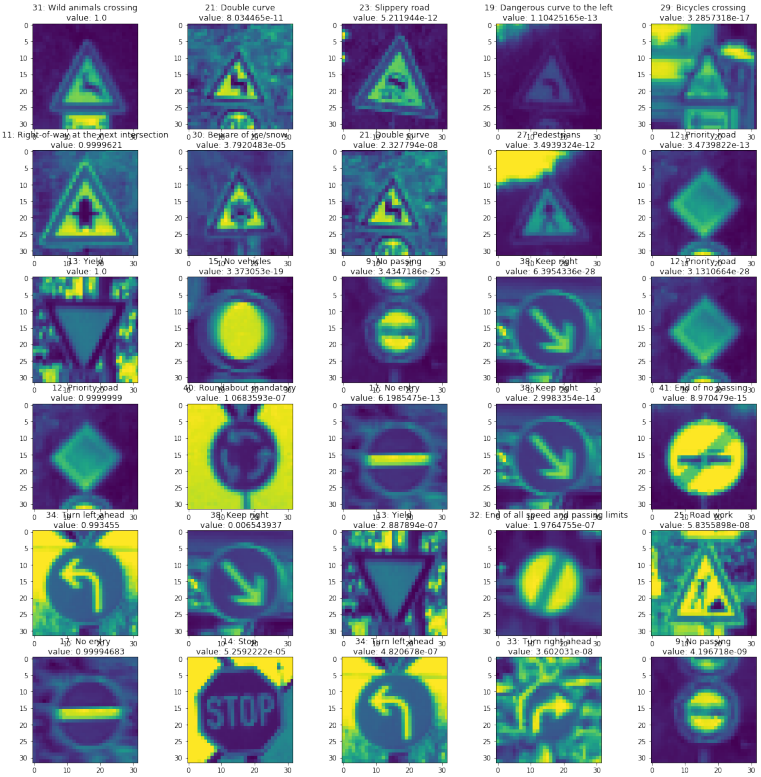

Figure 4: Top K

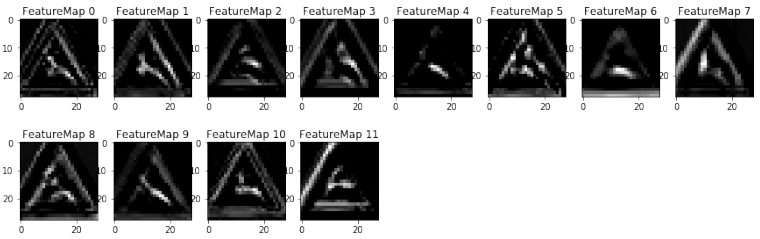

Figure 5: Feature Map