**Traffic Sign Recognition**

**Data Set Summary**

I used the numpy library to calculate summary statistics of the traffic signs data set. The statistics are as follows:

1) Size of training set is 34799.

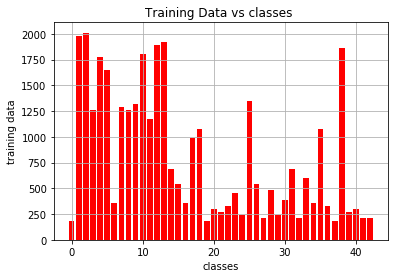
2) The size of the validation set is 4410, 32, 32, 3.

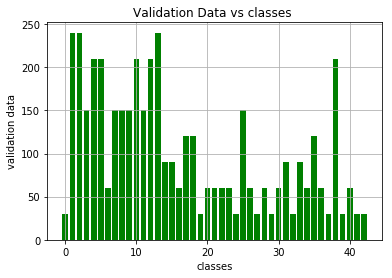
3) The size of the test set is 12630, 32, 32, 3.

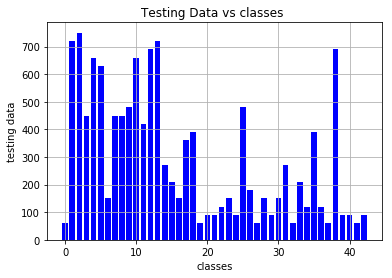
4) The shape of a traffic sign image is 32x32x3, where 32 is the height, 32 is the width and 3 is the number of colour channels since it’s a colour image.

5) The number of unique classes/labels in the data set is 43.

**Here is the bar chart showing how the classes is distributed over training, validation and test data set. As we can see below from the bar chart that the distribution is not same. Examples of some of the classes are more than the others**.

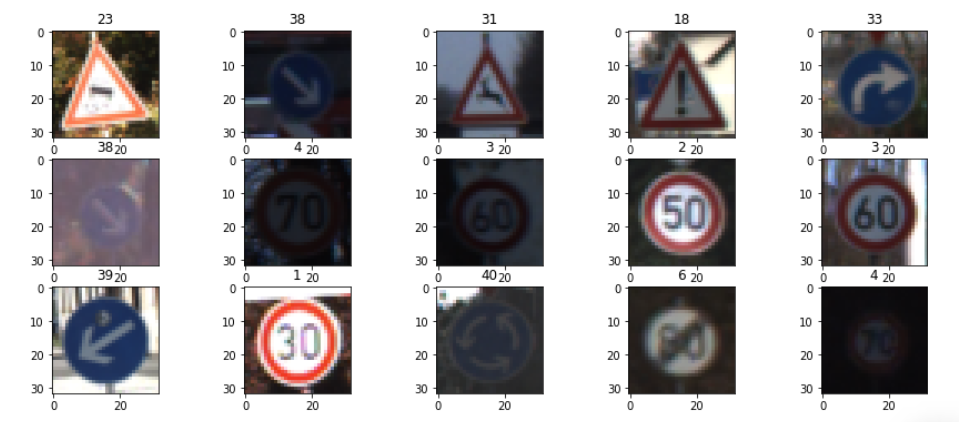




****

**Exploratory visualization of the dataset**

**These are the random traffic sign images from the data set:**



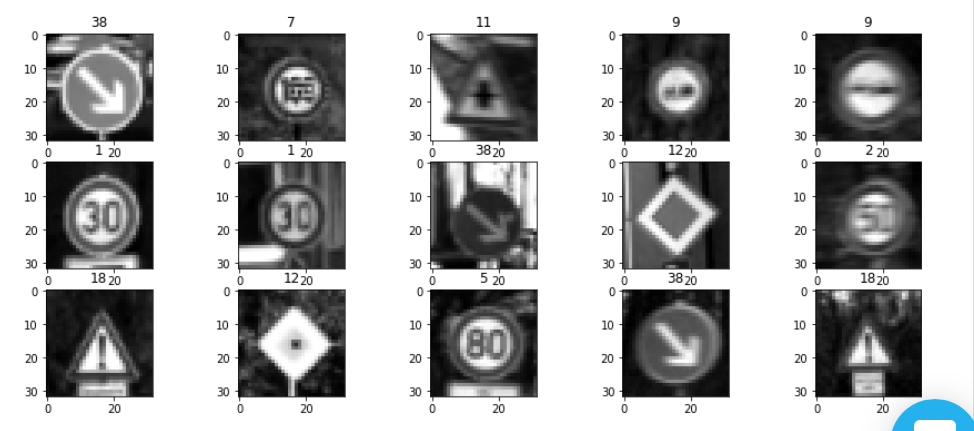
What we can observe from the dataset is that some images are dark, means they are not properly illuminated, because of which many of the signs are not clear.

**Design and Test a Model Architecture**

**Preprocessing**

**Grayscale:** As the first step of preprocessing, the images are converted into grayscale images which is as shown below:

**The corresponding grayscale images to the above colour images are:**



The images are converted to grayscale because it reduces the training time. The number of channels or the depth reduces from 3 to 1.

**Normalization:** As a last step, I normalized the image data to zero mean and equal variance. This is done because by normalizing all of our inputs to a standard scale, we're allowing the network to quickly learn the optimal parameters for each input node.

Computers lose accuracy when performing math operations on large or small numbers. Zero mean generally helps in easy calculations in algorithm. In other words, we get numerical stability.

If you have equal variance along dimensions, then doing gradient descending is faster to converge to the optimal solution. In other words, again, for the optimizer to work well, the problem should be wee conditioned.

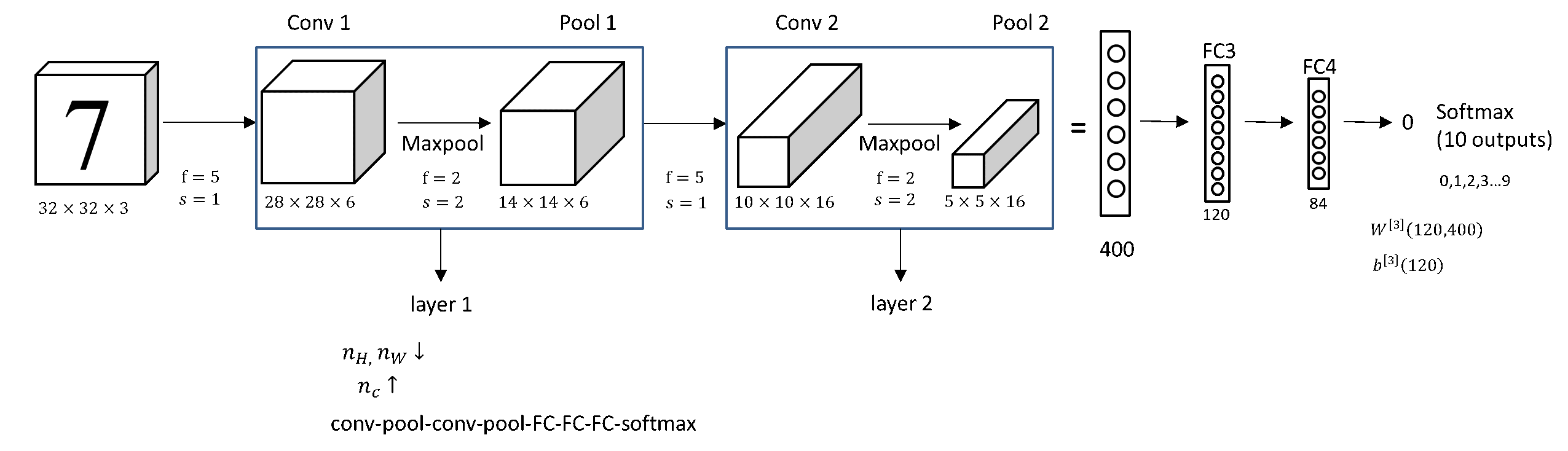
Moreover, if your inputs and target outputs are on a completely different scale than the typical -1 to 1 range, the default parameters for your neural network (i.e.. learning rates) will likely be ill-suited for the data.

**Final model architecture**

1. **Model type:**

Le-Net model

1. **Layers:**
2. Layer 1: Convolutional plus maxpool
3. Layer 2: Convolutional plus maxpool
4. Layer 3: Fully connected layer
5. Layer 4: Fully connected layer
6. Layer 5: Fully connected layer
7. **Diagram:**



1. **Table of the model**

|  |  |
| --- | --- |
| **Layer** | **Description** |
| Input | 32x32x1 |
| Convolution(5x5) | 1x1 stride, valid padding, outputs 28x28x6 |
| RELU | Activation Layer |
| Max pooling | 2x2 stride, outputs 14x14x6 |
| Convolution(5x5) | 1x1 stride, same padding, outputs 10x10x6 |
| RELU | Activation Layer |
| Max pooling | 2x2 stride, outputs 5x5x6 |
| Fully connected | Input – 400, output-120 |
| RELU | Activation Layer |
| dropout | Input – 120, output-120 |
| Fully connected | Input – 120, output-84 |
| RELU | Activation Layer |
| dropout | Input – 84, output-84 |
| Fully connected | Input – 84, output-43 |

**How the model was trained?**

1) **SoftMax:** Once we get the logits from the training model, we apply SoftMax to get the probabilities of the image belonging to the particular class.

2) **one hot encoding:** The labels or the classes are then one hot encoded.

3) **Cross-entropy** :Now, we have two vectors: the logits and the one-hot encoded labels. We measure the distance between the two vectors with a process called cross-entropy**.** When the distance is low, the class is supposed to be correct for the particular input, which in our case is the image.

4) **Training loss or average cross entropy:** Since we have a large number of examples in our training set, we average the distance over the entire training set, over all the inputs and all the labels available. This is called either training loss or average cross entropy.

5) **Adams optimizer**: Our goal is to get the smallest distance possible, the lowest loss. In order to do so, we use an optimizer, in our case, Adams optimizer and we run the minimize function on the optimizer which uses backpropagation to update the network and minimize the training loss. Adam optimization method is used over simple gradient descent, the ‘momentum’ in Adam optimizer provides much faster convergence.

6) **Batch size:** Batch size is selected as 64, It was observed that larger batch size did not yield to high accuracy. I tried 128 and 256.

7) **Number of epochs:** This solely depends upon the computation ability, I tried 20, though within 10 also the results were good.

8) For learning rate, default value of 0.001 is used. Low learning is better for better accuracy though it increases the number of epochs.

**My final model results were**:

1) training set accuracy of **99.4%**

2) validation set accuracy of **96%**

3) test set accuracy of **93.4%**

**CNN architecture**

The architecture that was chosen was the **LeNet architecture**. LeNet is small and easy to understand — yet large enough to provide good results.

Initially I was not getting the desired 93 percent accuracy. In order to achieve that accuracy or more, I had to tune some hyperparameters like the batch size and the number of epochs. **I reduced the batch size to 64 and increased the number of epochs to 20.**

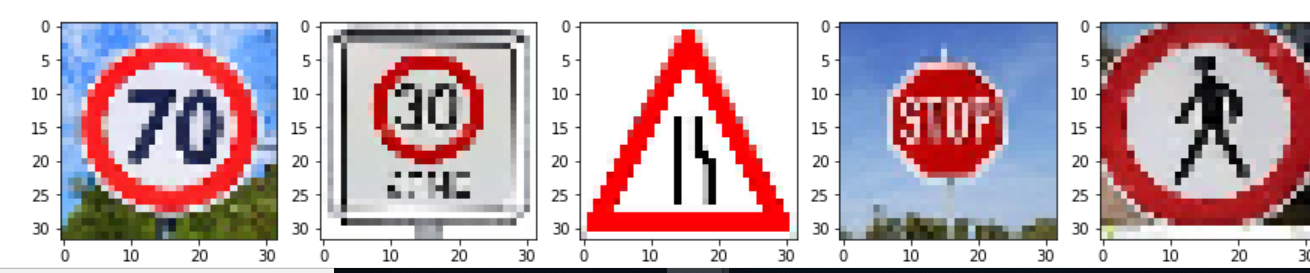
The design choices that were made for the LeNet was the 2 layers of convolutional plus max pooling layer and then 3 layers of fully connected layer.

**Convolutional Layer**: The convolutional layer’s main objective is to extract high level features like edges, colour and gradient orientation which is important for traffic sign classification. Then this convolved image is reduced in dimensions by valid padding.

**Max-Pooling Layer**: Similar to convolutional layer, pooling layer is also responsible for reducing the dimensions of the convolved layer. This is to reduce the computational power and to extract dominant features are rotational and positional invariant.

**Testing Model on New Images**

**Here are five German traffic signs that I found on the web:**



Some of the images will be difficult to classify because:

1. The letters are not vert clear like that of stop and under the 30 signs.
2. The shape of the pedestrian will be difficult to detect.
3. Also, compared to the test images, these images are slightly brighter, so it might not be able to detect the images correctly.

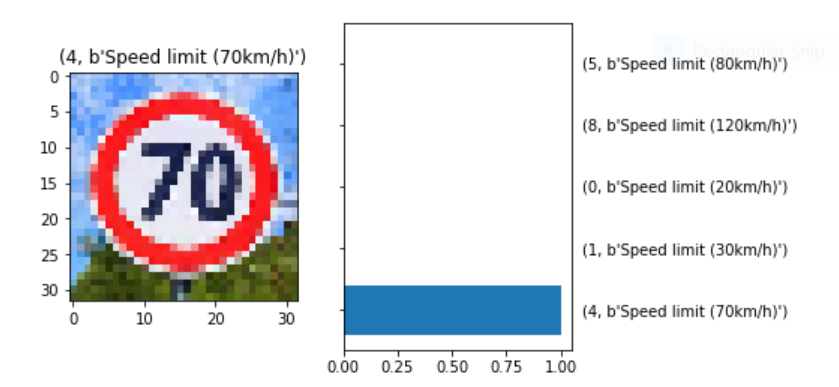
**Performance:**

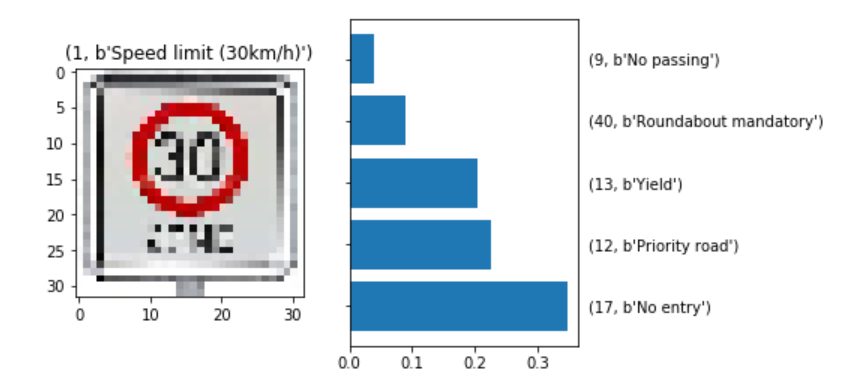
The performance on the new images with my training model is 40 percent which means it ahs only detected two signs correctly, which is the ” speed limit of 70km/hour “ and the “road narrows on the right”.

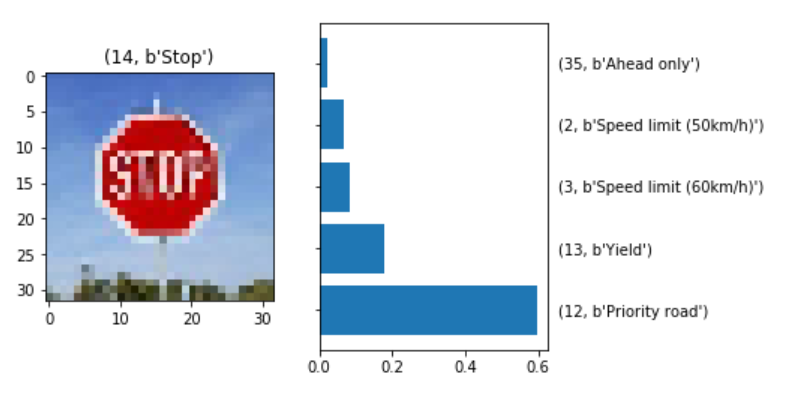
As compared to the accuracy results of the test set which is 93.8%, it is quite low. It is because the given data set images vary vastly from the images taken from the net in terms of brightness and clarity.

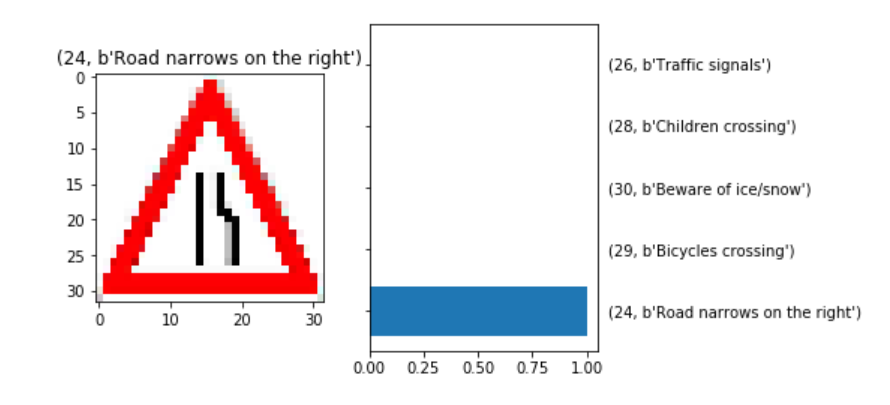
**Model Certainty - SoftMax Probabilities**

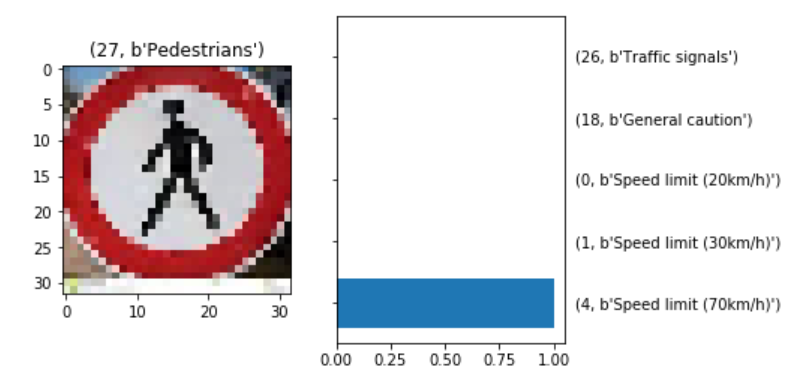
**Here are the top 5 SoftMax probabilities of each image:**











|  |  |  |  |
| --- | --- | --- | --- |
| **Signs** | **Accuracy** | **Detection** | **Reason** |
| Speed limit 70 | correct | Speed limit 70 | The numbers might be detected accurately |
| Speed limit 30 | Incorrect | No entry | The words under the 30 symbols was unclear |
| stop | Incorrect | Priority road | Again, the letters might be unclear |
| Right lane curve | Correct | Right lane curve | The symbol shape of right curve must have been detected clearly along with triangular sign. This is because rest of the probabilities of the symbols are mostly inside a triangle. |
| pedestrian | Incorrect | Speed limit 70 | Pedestrian is also encircled by round circle encircling it since the top 3 probabilities are of the speed limits which are encircled |