# Write up of Traffic Sign Classification

## 1.Image References (Test on New image)

**#number of images = 6 /   korea traffic Signs**

**[image1]:** ./png/test/1.png   **1.Speed limit(30km/h)**

**[image2]:** ./png/test/2.png   **4.Speed limit(70km/h)**

**[image3]:** ./png/test/3.png   **27.Pedestrian /crosswalk**

**[image4]:** ./png/test/4.png   **28.Children crossing**

**[image5]:** ./png/test/5.png   **22. Bumpy Road**

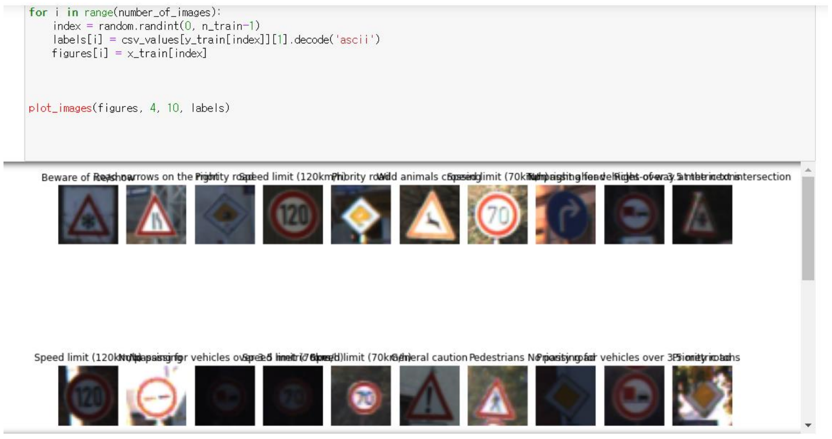
**[image6]:** ./png/test/6.png   **14.Stop**

**# German Traffic signs _ image data**

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

## 2. Include an exploratory visualization of the dataset.

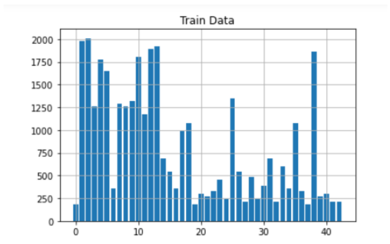
Please refer my result (ipynb File)

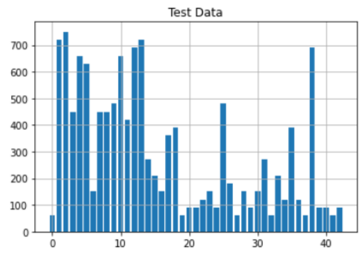


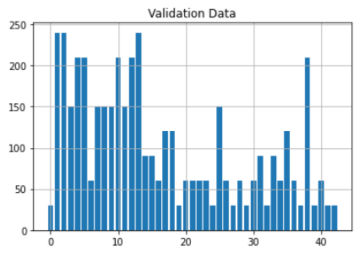
```
for i in range(number_of_images):
    index = random.randint(0, n_train-1)
    labels[i] = csv_values[y_train[index]][1].decode('ascii')
    figures[i] = x_train[index]



plot_images(figures, 4, 10, labels)
```







## 3. Design and Test a Model Architecture

**#1. Describe how you preprocessed the image data.**

I used a technique that is already well known. First  Grayscaling and  then normalization.

And added it into one function.

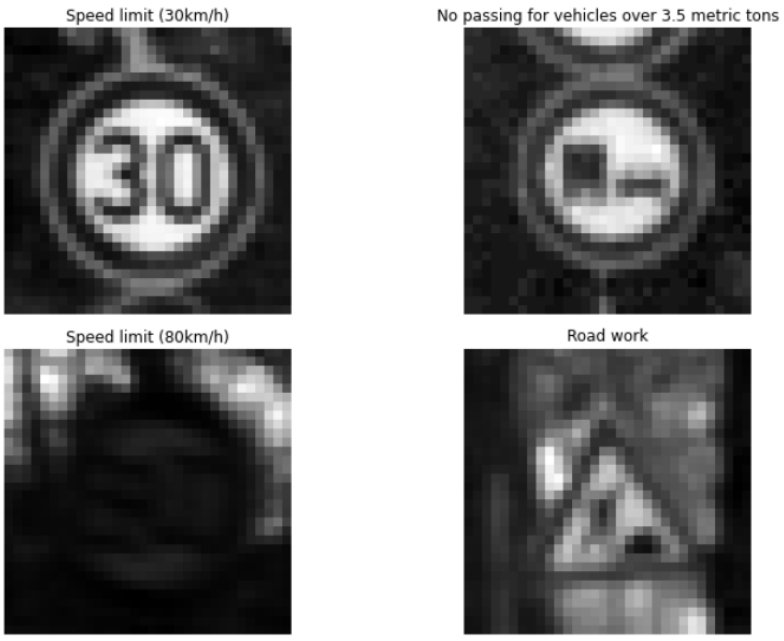
The result is shown below.

```
def image_processing(a): #grayscale and normalizing at one time
    # grayscale
    gray = np.sum(a/3 ,axis=3, keepdims=True)
    normal_gray = (gray-128)/128
    return normal_gray
```

And then I check the result of the function.

```
def grayimages(sample_images):
    for i in range(sample_images):
        idx = random.randint(0, n_train-1)
        gray_labels[i] = csv_values[y_train[idx]][1].decode('ascii')
        gray_imgs[i] = gray_x_train[idx].squeeze()
        random_imgs.append(idx)
    return gray_labels,gray_imgs

(gray_labels,gray_imgs) = grayimages(sample_images)
plot_images(gray_imgs,4,2,gray_labels)
```



**2. Describe what your final model architecture looks like including model type, layers, layer sizes, connectivity, etc.) Consider including a diagram and/or table describing the final model.**

| Layer | Description |
|---|---|
| **Input** | 32x32x1 Grayscale image |
| **1st Convolution 5x5** | 1x1 stride, VALID padding, outputs 28X28X6 |
| **RELU** | |
| **1st .Dropout** | Keep probability = 0.7 |
| **Max pooling** | 2x2 stride, outputs 14x14x6 |
| **2nd Convolution 5x5** | 1x1 stride, VALID padding, outputs 10x10x16 |
| **RELU** | |
| **2nd Dropout** | Keep probability = 0.7 |
| **Max pooling** | 2x2 stride, outputs 5x5x16 |
| **(1)_Fully connected** | Output = 400. |
| **3rd Dropout** | Keep probability = 0.7 |
| **(2)_Fully connected** | Output = 120. |
| **RELU** | |
| **4th Dropout** | Keep probability = 0.7 |

| | |
|---|---|
| **(3)_Fully connected** | Output = 84. |
| **RELU** | |
| **5th Dropout** | Keep probability = 0.6 |
| **(4)_Fully connected** | Output = 43. |

**3. Describe how you trained your model. The discussion can include the type of optimizer, the batch size, number of epochs and any hyperparameters such as learning rate.**

To make a train the model, Initially I used same Lenet Model as given in class tutorial,

But result was just zigzag graph.

So, I realized that it requires Dropout layers.  I added some of dropout layers.

Increase of dropout layers, the result is almost same. Accuracy remained in 85~95%.

Finally , I just leaved it as final test model (5 dropout layers)

**4. Describe the approach taken for finding a solution and getting the validation set accuracy to be at least 0.93. Include in the discussion the results on the training, validation and test sets and where in the code these were calculated. Your approach may have been an iterative process, in which case, outline the steps you took to get to the final solution and why you chose those steps. Perhaps your solution involved an already well known implementation or architecture. In this case, discuss why you think the architecture is suitable for the current problem.**

I tried to get the best Accuracy result.

I worked and adjusted the dropout layers and various parameters.

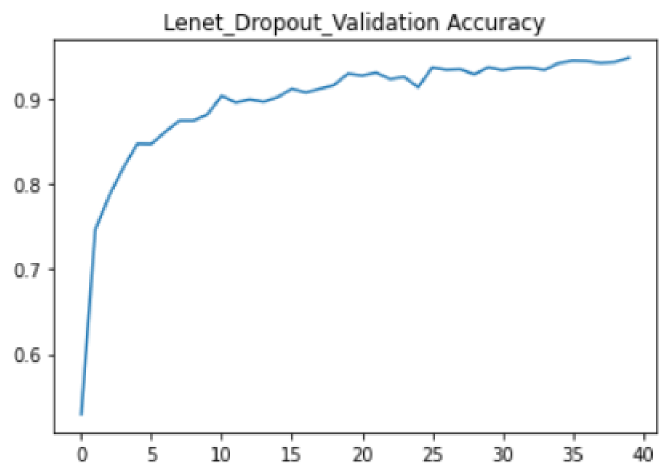
It includes like learning rate, epoch's.

- Learning Rate = 0.001 is good for this analysis.

- Epoch = 40 (when Over 25 Epochs , Accuracy result remained in the range(92~95%)
- batch Size = 100 (I realized that small batch size more time to train)
- From 2~5 Dropout Layers (Accuracy rate is same)

## Summary of My LeNet Result

```
Train Accuracy = 0.983
Test Accuracy = 0.926
Valid Accuracy = 0.949
```



**If an iterative approach was chosen:**

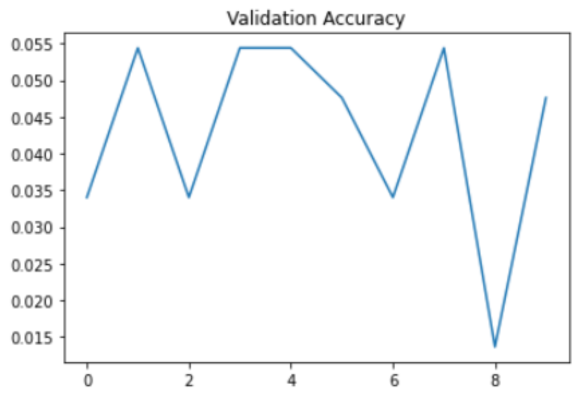
* **What was the first architecture that was tried and why was it chosen?**

Udacity Lesson gave me the LeNet Architecture and I used it.

* **What were some problems with the initial architecture?**

The given LeNet model has no dropout layers. First time, I didn't knew about dropout layers.

The result is below graph.



*** Which parameters were tuned? How were they adjusted and why?**

**Epochs** : over 25 Epochs, the result is almost same. But finally I decided to use 40 Epochs.

I gave a chance to make a good result for my computer.

**Learning Rate** : I adjust Learning rate from 0.1 to 0.001. when I setting it as 0.1, It shows errors (overshotting). But from 0.01 to 0.001, this architecture show me the result of accuracy (85~95%)

* **What are some of the important design choices and why were they chosen? For example, why might a convolution layer work well with this problem? How might a dropout layer help with creating a successful model?**

I tested dropout layers from 2 to 5 Layers. But Result is almost same!

So, I'm considering 2 Dropout layer is enough for this analysis.

Finally, I leaved it as Final test model ( 5 Dropout Layers).

**If a well known architecture was chosen:**

*** What architecture was chosen?**

LeNet (it was given from Udacity)

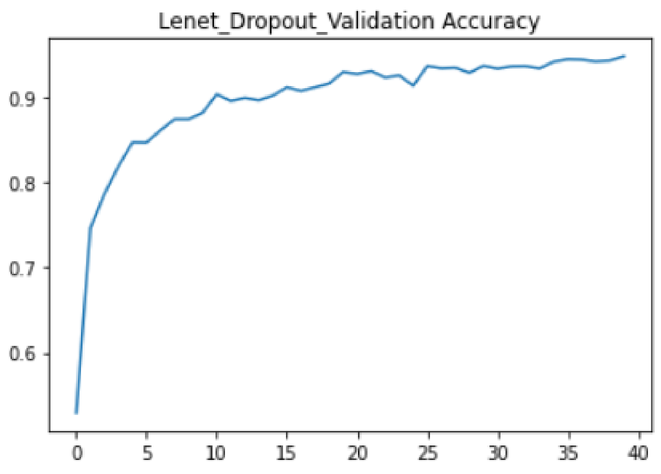
* **Why did you believe it would be relevant to the traffic sign application?**

CNN is the best solution for image prediction so far. A well known solution.

LeNet has a CNN layer and it was enhanced from original CNN architecture

*** How does the final model's accuracy on the training, validation and test set provide evidence that the model is working well?**

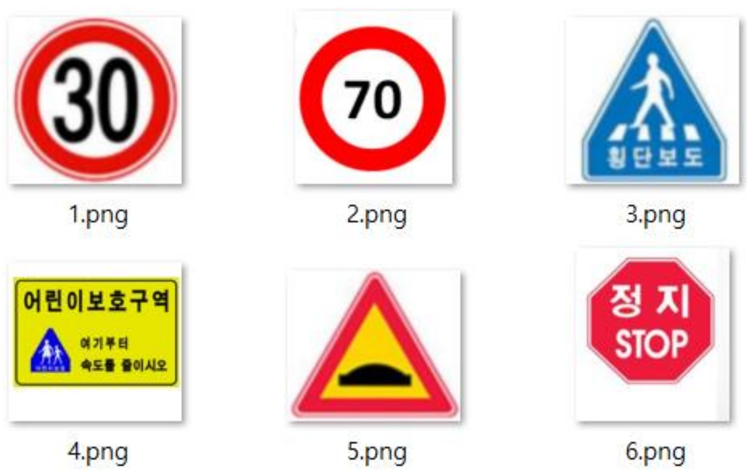
Final accuracy is 0.949(almost 95%). It is reliable.



**Test a Model on New Images**

**1. Choose five German traffic signs found on the web and provide them in the report. For each image, discuss what quality or qualities might be difficult to classify.**

Unfortunately, I decided to use Korea traffic signs in this analysis.



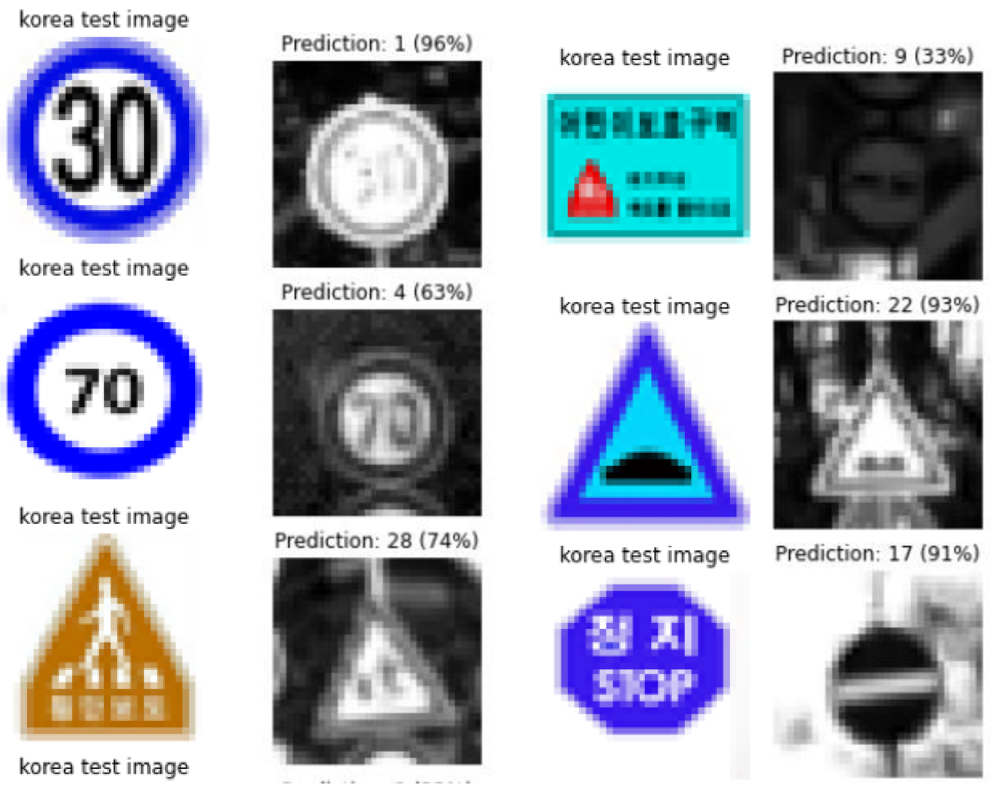
The prediction reate is 0.5.   I'm considered korean alphabet caused analysis difficulties.

Some of sign has quite different style.   Korean alphabet might be cause prediction failure.

Number 3 image (pedestrian) has a quite big size of Korean alphabet.

And number 4 image(Child protection zone )has quite different in style from German signs.

**2. Discuss the model's predictions on these new traffic signs and compare the results to predicting on the test set. At a minimum, discuss what the predictions were, the accuracy on these new predictions, and compare the accuracy to the accuracy on the test set (OPTIONAL: Discuss the results in more detail as described in the "Stand Out Suggestions" part of the rubric).**



3. Describe how certain the model is when predicting on each of the five new images by looking at the softmax probabilities for each prediction. Provide the top 5 softmax probabilities for each image along with the sign type of each probability. (OPTIONAL: as described in the "Stand Out Suggestions" part of the rubric, visualizations can also be provided such as bar charts)

| Korea Test Images | | Prediction | |
|---|---|---|---|
| 1.Png | Speed limit(30km/h) | Speed imit (30km/h) | 96% |
| 2.Png | Speed limit(70km/h) | Speed limit (70km/h) | 63% |
| 3.Png | Pedestrian /crosswalk | Pedestrian | 74% |
| 4.Png | Children crossing/Child Protection zone) | ??? | 33% |
| 5.Png | Bumpy Road | Bumpy Road | 93% |
| 6.Png | Stop | Stop | 91% |