Self-driving Car Nanodegree Program



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Project 3: Traffic Sign Recognition

Due: 01/22/2019

# **Objectives**

The goals of this project are:

* Load the German Traffic Sign dataset
* Explore, summarize and visualize the dataset
* Design, train and test a model architecture
* Use the model to make predictions on new images
* Analyze softmax probabilities of the new images

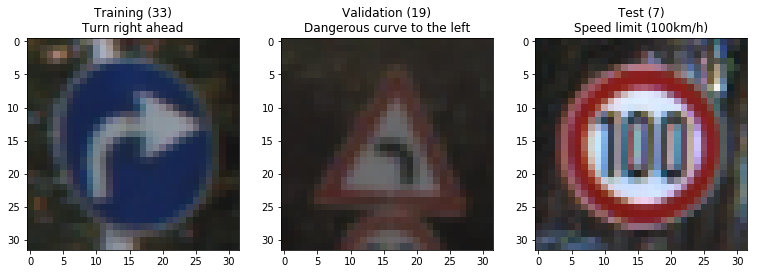
# **Procedures**

## Dataset Summary and Exploration

For this project, we will implement a convolutional neural network (CNN) to classify traffic signs provided by German dataset. After loading datasets for Training, Validation and Test sets, below are summary of these dataset:

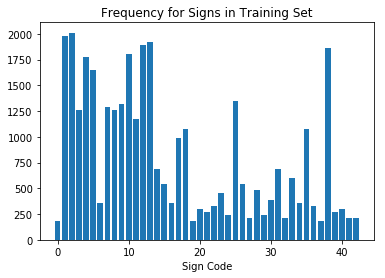
* Number of training examples: 34799
* Number of validation examples: 4410
* Number of testing examples: 12630
* Image data shape: (32, 32, 3)
* Number of classes/labels: 43

Examples of traffic signs from Training, Validation and Test sets are shown in Figure 1.



*Figure 1: Traffic Signs examples*

It is worth to note the number of occurrences (frequencies) that each sign is appeared in the training set since ideally, we would want to have equal number of training examples for each sign class. Figure 2 shows the number of occurrences for each class in the training set.



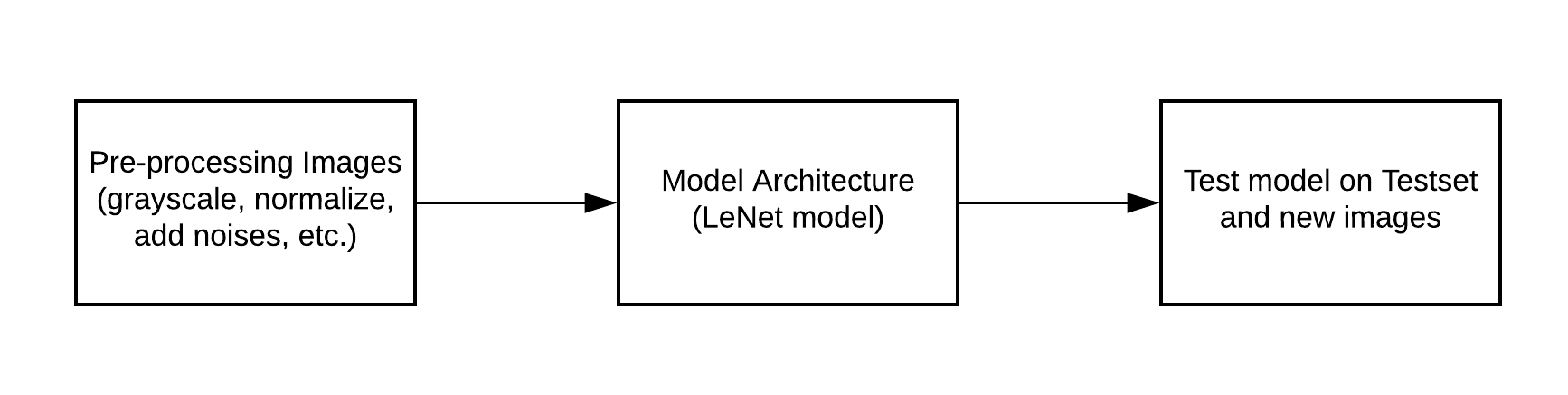
*Figure 2: Number of occurrences for each sign class in training set*

The top 5 most occurrences signs are:

* Sign class 2 (2010 examples): Speed limit (30km/h)
* Sign class 1 (1998 examples): Speed limit (20km/h)
* Sign class 13 (1920 examples): Yield
* Sign class 12 (1890 examples): Priority Road
* Sign class 38 (1860 examples): Keep right

## Design and Test a Model Architecture

The main pipeline is described in the flow chart below.



*Figure 3: Project 3 Pipeline*

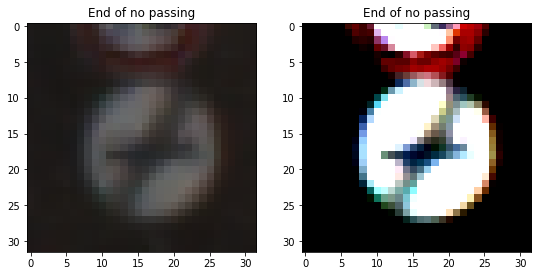
We will discuss each of these steps in details in the following subsections.

### Image Pre-processing

For this step, besides converting from RGB to gray scale and normalization to have image pixels of zero mean, we will do some additional data image augmentation techniques, including:

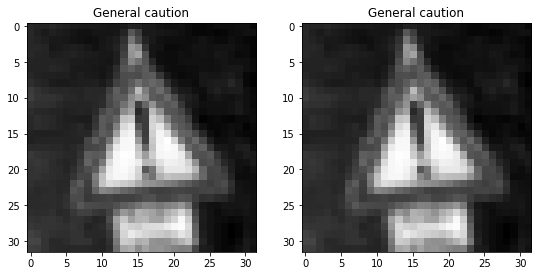
* Generate more data for labels that occur less than 500 times so that the minimum number of training images for each class is 500 examples.
* Adjust brightness randomly on color RGB images
* Adjust contrast on color image
* Add Gaussian noise (and optionally salt and pepper noise, and speckle noise) to the images

Figure 4 shows an example of random contrast on an image



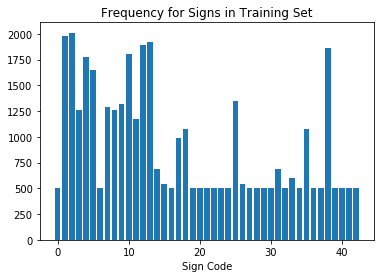
*Figure 4: Random contrast (left: original image, right: random contrast image)*

Figure 5 shows an example after applying random contrast and adding Gaussian noise to an image.



*Figure 5: Random contrast, convert to gray scale and add Gaussian noise*

Applying these random image augmentation techniques, we can generate more training examples so that each class has at least 500 examples for training. Figure 6 demonstrates the updated frequency for each class after generating more training examples.

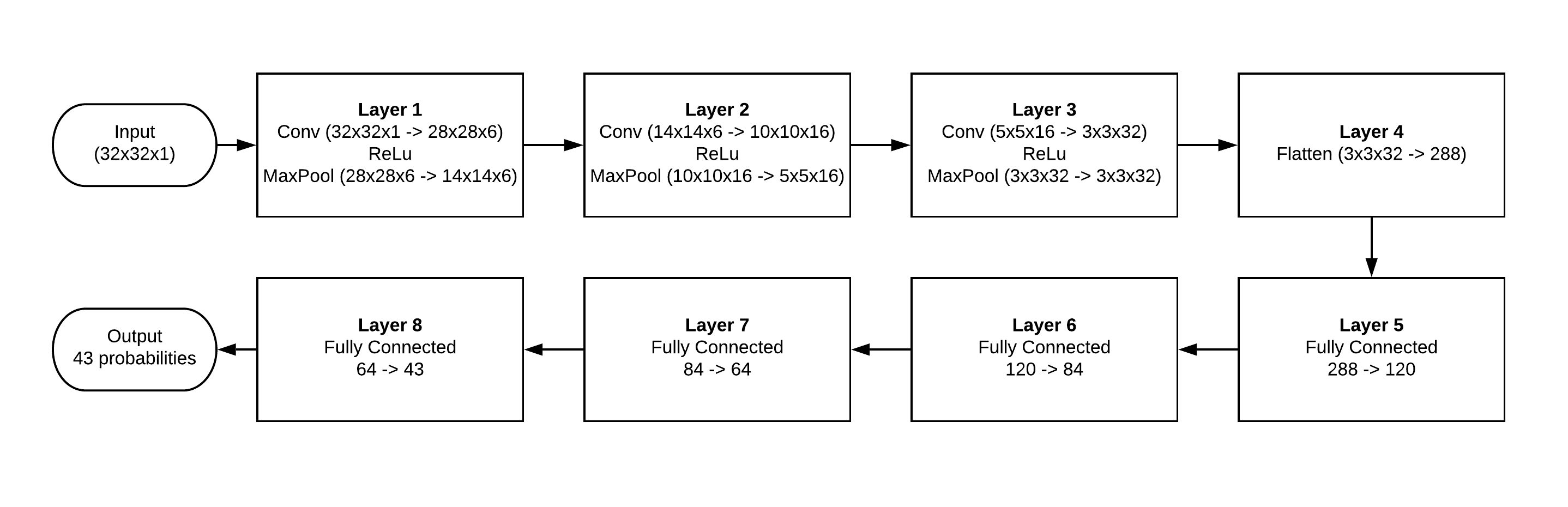


*Figure 6: Number of occurrences after generating more images for training*

As we can see, the minimum number of examples for each class is 500. It will be better for the training set as the model will have better amount of examples to learn from each class and thus, generalize the model. Note that we can even try to get more examples for the lower-frequency class for better model performance.

### Model Architecture

The process for lane finding can be found below:



*Figure 7: Model Architecture*

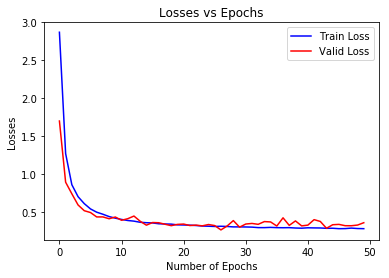
The model is pretty much similar with the original LeNet architecture suggested in class. However, since the original structure only provides around 89% accuracy on validation set, I tried to modify the structure by adding one more convolutional layer and one fully connected layer as shown in Figure 7. This helps provide a maximum of **94.5%** in accuracy of the validation set.

### Train and Validate the Model

To train the model and verify on the validation data, the structure is followed closely as shown in lecture quizzes. The parameters used for the final model are:

* Number of epochs: 50
* Number of batches: 128
* Dropout (probability to keep units): 0.8
* Learning rate: 0.001
* Optimizer: Adam Optimizer

The graph below plots the training loss versus the validation loss during training the model.



*Figure 8: Training Loss vs Validation Loss*

As we can see in Figure 8, the validation loss starts to cross the training loss at around 26-27 epochs. We can stop early there to prevent the model from overfitting the data. Again, with this architecture, I am able to obtain an accuracy of **94.5%** on the validation set.

## Test Model on New Images

First of all, we run the model on the test set of 12630 examples and we get **92%** of accuracy. Some predictions of the model on our test set are shown here:



*Figure 9: Predictions on test set*

Then we can download some traffic signs from website to test our model. One important thing to note here is to resize the image to 32x32 before feeding into the model and pre-processing steps discussed above. To do this, I used *cv2.resize()* function. Below are the predictions of our model on the new images.



*Figure 10: Predictions on the new images with accuracy of 90%*

As we can see in Figure 10, only one sign in the bottom left corner is wrong, resulting in 9/10 correct predictions by our model. Plot below shows the top 5 probabilities for each class for the new images.



# **Discussions (Further improvements)**

Further improvements can be done such as:

* Generate more training examples with more and advanced image augmentation techniques
* Tuning parameters such as number of batches, number of epochs, learning rate
* Try different model architectures