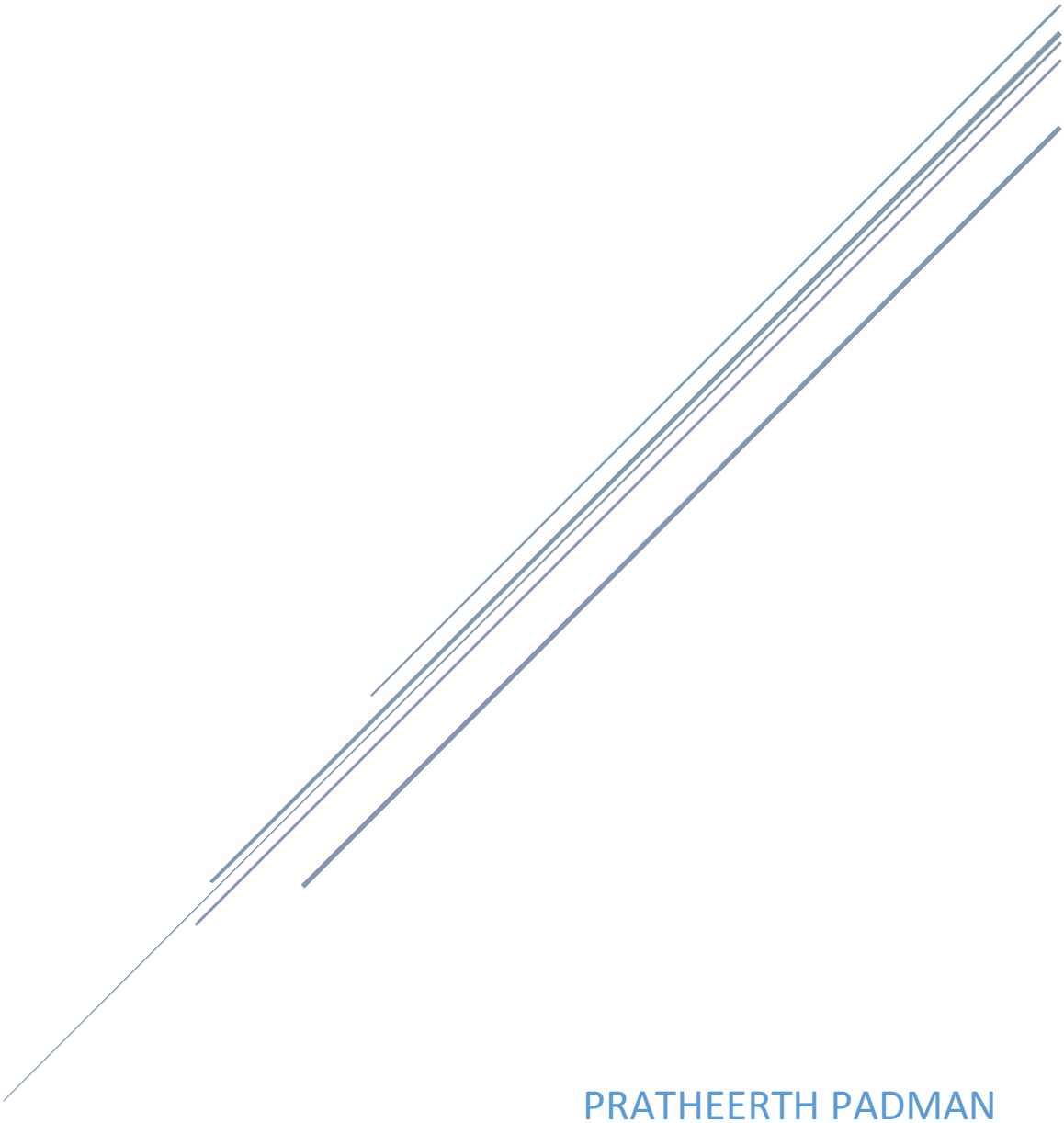
# P2 – TRAFFIC SIGN CLASSIFIER

## SELF DRIVING CAR NANODEGREE



PRATHEERTH PADMAN

# Goals

The goals / steps of this project are the following:

- Load the data set (see below for links to the project data set)
- Explore, summarize and visualize the data set
- Design, train and test a model architecture
- Use the model to make predictions on new images
- Analyze the softmax probabilities of the new images
- Summarize the results with a written report

# Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

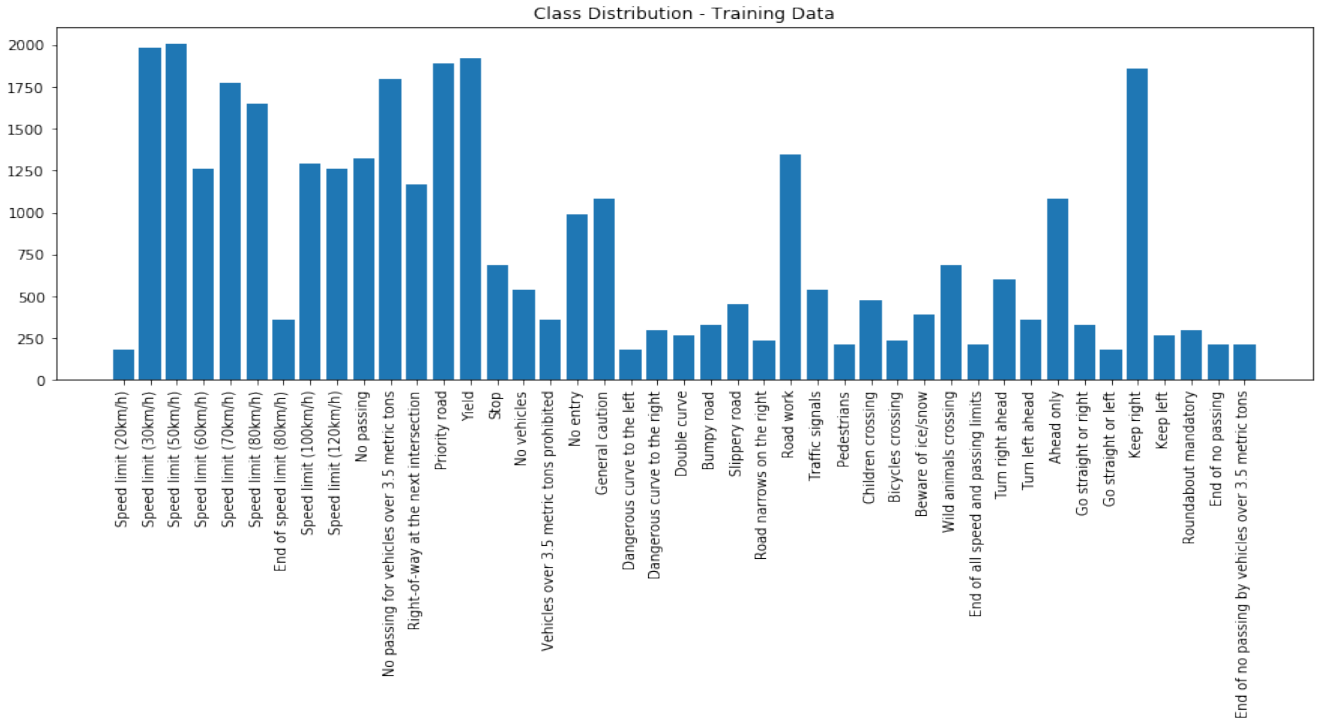
**Data Set Summary & Exploration:**

I used the pandas library to calculate summary statistics of the traffic signs data set:

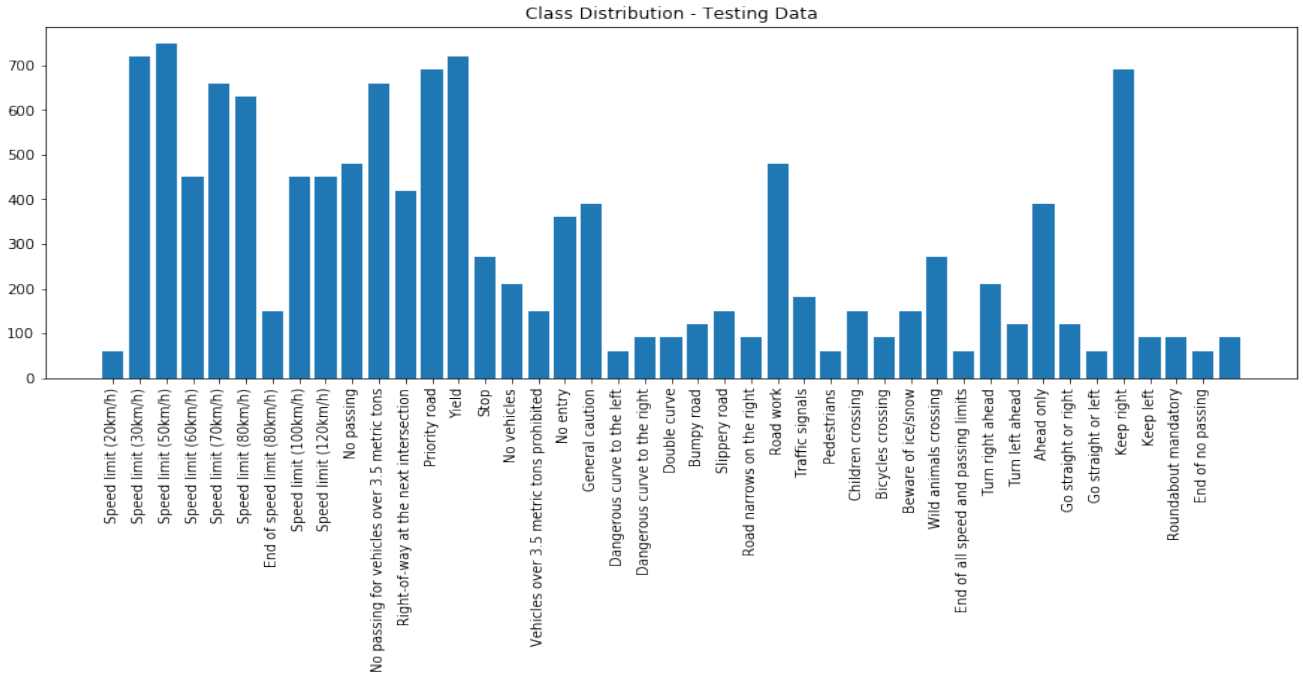
- The size of training set is - 34,799
- The size of the validation set is – 4,410
- The size of test set is – 12,630
- The shape of a traffic sign image is – (32, 32, 3)
- The number of unique classes/labels in the data set is – 43

## Visualization:

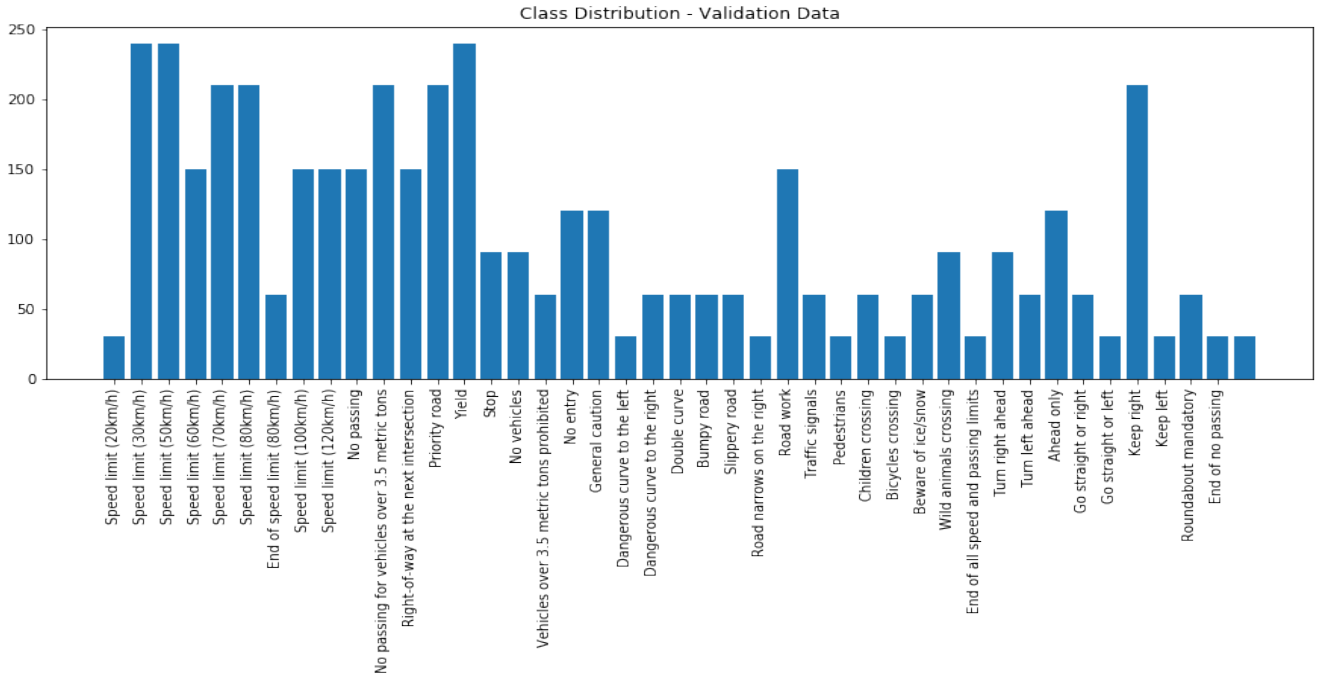
1. Training Data




2. Testing Data



3. Validation Data




4. Image Sampling

I looked at 15 random images from the training data to see the quality and type of images we're dealing with.

**Design and Test a Model Architecture:**

The model architecture is contained in Code Cell 17 in the Jupyter Notebook.

My model is a modification of the famous LeNet model for image classification.

Firstly, I decided to grayscale the given images. Since this is a comparatively simple model, gradient descent will have a simpler time converging because it has to optimize the weights and biases only on a single color channel.

Next, I normalized the image data within a range of -1 to +1. I did it using the formula:

X_normal = (X/122.5) – 1

Normalizing our image data is important as it keeps our input feature values within a fixed range which helps with the networks learning ability. If input features aren't of a similar scale, there is always a probability that one feature, due to its values, completely overwhelms the others.

To make it "harder" for the network during the training phase, so that it has an easier time during the testing phase, I decided to augment the training dataset.

I augmented the dataset by applying affine transformations (transformations where parallel lines before transformation remain parallel even after transformations) to each image in the training dataset such as rotations, translations and shearing.

The augmentation process can be found in Code Cells 7 and 8.

While I noticed the difference in the frequency of occurrence of each class in the datasets, I decided not to make them equal in order to give the

model a more “real-world” dataset to work with.

**Model Description:**

My final model consisted of the following layers:

| Layer | Description |
|---|---|
| Input | 32x32x1 Grayscale image |
| Convolution 5x5 | 1x1 stride, “VALID” padding, outputs 28x28x6 |
| ReLU | Activation Layer |
| Dropout | Keep Probability – 0.75 |
| Max pooling | 2x2 stride, outputs 14x14x6 |
| Convolution 5x5 | 1x1 stride, “VALID” padding, outputs 10x10x16 |
| ReLU | Activation Layer |
| Dropout | Keep Probability – 0.75 |
| Max Pooling | 2x2 stride, outputs 5x5x16 |
| Flatten | Outputs 400 |
| Fully Connected | Output - 120 |
| ReLU | Activation Layer |
| Dropout | Keep Probability – 0.75 |
| Fully Connected | Output - 84 |
| ReLU | Activation Layer |
| Dropout | Keep Probability – 0.75 |
| Fully Connected | Output - 43 |

**Model Training:**

The code and the hyper parameters used can be found on Code Cells 19, 20 and 21.

I used an Adam Optimizer as specified in the LeNet Model. The batch size was 256, learning rate was kept at 0.001 and the number of epochs used was 30, which as you can see from the Loss graph below was just enough.

**Training Approach:**

Initially, I had trained my model using the LeNet model as it stood, only changing small factors inorder to properly reflect our dataset. It gave me an expected validation accuracy of ~ 0.89.

I gray scaled and normalized the images which increased the validation accuracy to ~0.91

Including the augmented data in the training set and using dropout layers (which force the network to learn redundant information about the input features while reducing co-adaptability and co-dependence, thus preventing overfitting) are what really caused the accuracy to take off and settle at around ~0.96+

My final model results were:
- training loss – 0.084
- validation set accuracy - 0.965
- test set accuracy - 0.932

If an iterative approach was chosen:

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

  The first architecture that was tried was the LeNet model as it was in order to gauge the performance of the model on our dataset and to set a baseline

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

  The model was too simple to reach a validation accuracy of 0.93 +

- **How was the architecture adjusted and why was it adjusted?**

  As explained above.

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

  The learning rate was kept at 0.001 as decreasing it would mean increasing the number of epochs to stabilize which would mean increase in the time required to train.

  The number of epochs was raised to 30. This represented enough time to properly train the model without compensating model performance for time taken. It also does not overfit as seen from the loss graph.

  The probability of keeping a unit in the dropout layer was at 0.75 which represented an ideal tradeoff between optimal training and under-training.

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

  LeNet model is known to work well on digit classification tasks such as the MNIST model. It would thus also make sense on something like a traffic sign classification application.

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

  Firstly, the model can be seen as starting from a high training and validation loss which reduces with each epoch.

  We can see that the model does not overfit because of the gap in the validation and training loss.

  Lastly, the model can be seen generalizing well as seen from the test set accuracy of 0.932.

**Test a Model on New Images:**

Here are 10 German traffic signs that I found on the web:


I have the benefit of hindsight here, as while using an earlier trained model with similar test set accuracy, I found that the model suffered with predicting “real world” images which included weird angles, additional imagery and text. The model seemed to do well with properly cropped, “official” traffic sign images.

With this in mind, I expect it to have some difficulty in predicting the 30 km/h speed limit sign as well as the priority road sign.

**Model's performance on new images:**

Here are the results of the prediction:

| Image | Prediction |
|---|---|
| Right of way at next intersection | Right of way at next intersection |
| Keep Right | Keep Right |
| Stop | Yield |
| 30 km/h speed limit | 30 km/h speed limit |
| Bumpy Road | Road narrows on the right |
| Priority Road | Priority Road |
| No vehicles | Keep Right |
| No passing | No passing |
| No entry | No entry |
| Children crossing | Children Crossing |

The model correctly classifies 7 out of 10 images correctly for an accuracy of 70%.

This obviously is well below the testing and validation set accuracy which might point to overfitting. It could also be down to the fact that we have only taken 10 new images and misclassifying even one of them leads to a 10% reduction in accuracy. The model might do better on a larger dataset.

**Softmax Probabilites:**

The code for the softmax probabilities is in Code Cell 63.

For the first image, the model is completely sure that this is a right of way sign and the image does contain it. The top five soft max probabilities were

| Probability | Prediction |
|---|---|
| 1.0 | Right of way at next intersection |
| 0 | Priority Road |
| 0 | Beware of ice |
| 0 | Slippery road |
| 0 | Double curve |

For the second image:

| Probability | Prediction |
|---|---|
| 1.0 | Keep right |
| 0 | Turn left ahead |
| 0 | Go straight or right |
| 0 | Slippery road |
| 0 | Yield |

For the third image:

| Probability | Prediction |
|---|---|
| 0.99000001 | Yield |
| 0.007 | Go straight or right |
| 0.001 | Right of way |
| 0.001 | Keep Right |
| 0.001 | Speed 60 km/h |

For the fourth image:

| Probability | Prediction |
|---|---|
| 0.8333 | Speed 30km/h |
| 0.145 | Speed 50km/h |
| 0.021 | Speed 20km/h |
| 0 | Speed 80 km/h |
| 0 | Speed 100 km/h |

For the fifth image:

Here, in one of the misclassified image, the model seems to have trouble and it nearly predicted the right sign.

| Probability | Prediction |
|---|---|
| 0.563 | Road narrows on right |
| 0.3829 | Bumpy road |
| 0.053 | Bicycles crossing |
| 0.001 | Traffic signals |
| 0 | Pedestrians |
| | |

For the sixth image:

| Probability | Prediction |
|---|---|
| 0.985 | Priority Road |
| 0.009 | Yield |
| 0.004 | Ahead only |
| 0.001 | Speed 60 km/h |
| 0 | Turn left ahead |

For the seventh image:

Here, even though the second option is correct, the model seems to overwhelmingly believe that the first, wrong prediction is right.

| Probability | Prediction |
|---|---|
| 0.995 | Keep right |
| 0.003 | No vehicles |
| 0 | Ahead only |
| 0 | End of speed limits |
| 0 | Priority road |

For the eighth image:

| Probability | Prediction |
|---|---|
| 1.0 | No passing |
| 0 | Priority road |
| 0 | No entry |
| 0 | No vehicles > 3.5 T |
| 0 | End of no passing |

For the ninth image:

| Probability | Prediction |
|---|---|
| 1.0 | No entry |
| 0 | Stop |
| 0 | Turn left ahead |
| 0 | No passing for vehicles > 3.5 T |
| 0 | Vehicles over 3.5 T prohibited |

For the tenth image:

| Probability | Prediction |
|---|---|
| 1.0 | Children crossing |
| 0 | Dangerous curve to right |
| 0 | Right of way |
| 0 | Go straight or right |
| 0 | Slippery road |