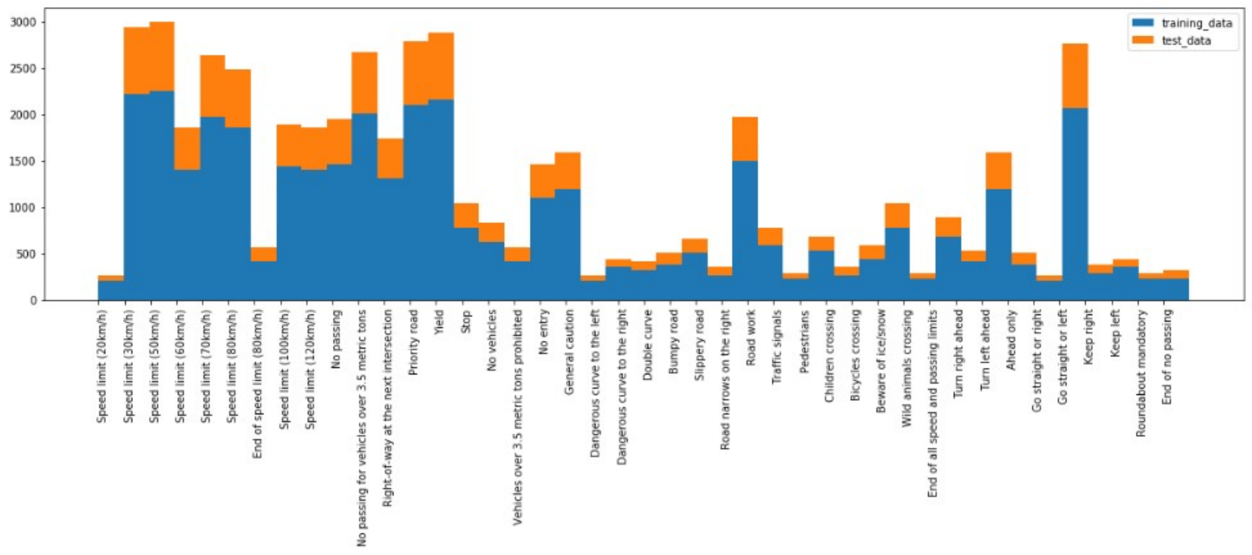
Traffic Sign Recognition

Data Set Summary & Exploration

1. Provide a basic summary of the data set and identify where in your code the summary was done. In the code, the analysis should be done using python, numpy and/or pandas methods rather than hardcoding results manually.

```
Number of training examples = 39209

Number of testing examples = 12630
Image data shape = (32, 32, 3)
Number of classes = 43

Following is the visualization of the dataset, which has been plotted in cell
3 (In[4])
```



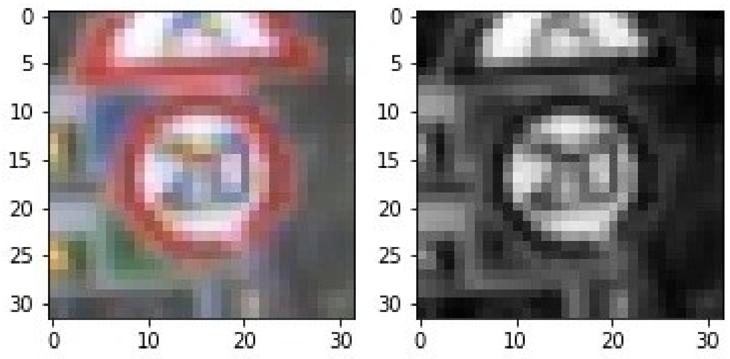
Design and Test a Model Architecture

1. Describe how, and identify where in your code, you preprocessed the image data. What tecniques were chosen and why did you choose these techniques? Consider including images showing the output of each preprocessing technique. Pre-processing refers to techniques such as converting to grayscale, normalization, etc.

The code for this step is contained in the fourth code cell of the IPython notebook.

As a first step, I decided to convert the images to grayscale because in this case, shape of the traffic sign matters and not color

Here is an example of a traffic sign image before and after grayscaling.



As a last step, I normalized the image data because that helps to bring all the pixel value to a certain scale. This is necessary to achieve global minima during optimization.

2. Describe how, and identify where in your code, you set up training, validation and testing data. How much data was in each set? Explain what techniques were used to split the data into these sets. (OPTIONAL: As described in the "Stand Out Suggestions" part of the rubric, if you generated additional data for training, describe why you decided to generate additional data, how you generated the data, identify where in your code, and provide example images of the additional data)

For this, I randomly selected 90% of training examples and put them in my test set, I used rest 10% for validation set. For testing, I am considering all the test examples.

3. Describe, and identify where in your code, what your final model architecture looks like including model type, layers, layer sizes, connectivity, etc.)

I am using the LeNet architecture for my model.

Layer 1: Convolutional. Input = 32x32x1. Output = 28x28x6. This is followed by relu activation and Pooling: Input = 28x28x6. Output = 14x14x6.

Layer 2: Convolutional. Output = 10x10x16. This is followed by relu activation and Pooling Input = 10x10x16.. Output = 5x5x16

Layer 3: Fully Connected. Input = 400. Output = 120. This is followed by relu activation

Layer 4: Fully Connected. Input = 120. Output = 84. This is followed by relu activation

Fully Connected. Input = 84. Output = 43. This is followed by relu activation

4. Describe the approach taken for finding a solution. 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.

My final model results using following hyperparamter were:

`EPOCHS = 20`

`BATCH_SIZE = 128`

`rate = 0.001`

```
EPOCH 1 ...

Validation Accuracy = 0.730

EPOCH 2 ...
Validation Accuracy = 0.866

EPOCH 3 ...
Validation Accuracy = 0.920

EPOCH 4 ...
Validation Accuracy = 0.934

EPOCH 5 ...
Validation Accuracy = 0.936

EPOCH 6 ...
Validation Accuracy = 0.954

EPOCH 7 ...
Validation Accuracy = 0.951

EPOCH 8 ...
Validation Accuracy = 0.961
```

```
EPOCH 9 ...
Validation Accuracy = 0.963

EPOCH 10 ...
Validation Accuracy = 0.965

EPOCH 11 ...
Validation Accuracy = 0.976

EPOCH 12 ...
Validation Accuracy = 0.972

EPOCH 13 ...
Validation Accuracy = 0.974

EPOCH 14 ...
Validation Accuracy = 0.974

EPOCH 15 ...
Validation Accuracy = 0.975

EPOCH 16 ...
Validation Accuracy = 0.968

EPOCH 17 ...
Validation Accuracy = 0.976

EPOCH 18 ...
Validation Accuracy = 0.971

EPOCH 19 ...
Validation Accuracy = 0.980

EPOCH 20 ...
Validation Accuracy = 0.981

Model saved
Test Accuracy = 0.900
```

•

If an iterative approach was chosen:

- What was the first architecture that was tried and why was it chosen? LeNet was used because it is a standard architecture and proven to work well on 32x32x1 images.
- What were some problems with the initial architecture? It is designed for 10 classes
- How was the architecture adjusted and why was it adjusted? I changed the output layer to give out 43 outputs instead of 10

•Which parameters were tuned? How were they adjusted and why? Number of epochs, learning rate and batch size, because these are the only hyperparameters that we have after we have designed the layers.

I ran another set of hyperparameters as follows and got the following accuracies:

EPOCHS = 50

BATCH_SIZE = 128

rate = 0.001

```
Training...


EPOCH 1 ...
Validation Accuracy = 0.766

EPOCH 2 ...
Validation Accuracy = 0.879

EPOCH 3 ...
Validation Accuracy = 0.921

EPOCH 4 ...
Validation Accuracy = 0.932

EPOCH 5 ...
Validation Accuracy = 0.952

EPOCH 6 ...
Validation Accuracy = 0.963

EPOCH 7 ...
Validation Accuracy = 0.962

EPOCH 8 ...
Validation Accuracy = 0.972

EPOCH 9 ...
Validation Accuracy = 0.974

EPOCH 10 ...
Validation Accuracy = 0.967

EPOCH 11 ...
Validation Accuracy = 0.974
```

```
EPOCH 12 ...
Validation Accuracy = 0.975

EPOCH 13 ...
Validation Accuracy = 0.974

EPOCH 14 ...
Validation Accuracy = 0.976

EPOCH 15 ...
Validation Accuracy = 0.976

EPOCH 16 ...
Validation Accuracy = 0.967

EPOCH 17 ...
Validation Accuracy = 0.979

EPOCH 18 ...
Validation Accuracy = 0.977

EPOCH 19 ...
Validation Accuracy = 0.980

EPOCH 20 ...
Validation Accuracy = 0.974

EPOCH 21 ...
Validation Accuracy = 0.983

EPOCH 22 ...
Validation Accuracy = 0.988

EPOCH 23 ...
Validation Accuracy = 0.975

EPOCH 24 ...
Validation Accuracy = 0.984

EPOCH 25 ...
Validation Accuracy = 0.986

EPOCH 26 ...
Validation Accuracy = 0.984

EPOCH 27 ...
Validation Accuracy = 0.985

EPOCH 28 ...
Validation Accuracy = 0.986

EPOCH 29 ...
Validation Accuracy = 0.983

EPOCH 30 ...
```

```
Validation Accuracy = 0.985

EPOCH 31 ...
Validation Accuracy = 0.984

EPOCH 32 ...
Validation Accuracy = 0.982

EPOCH 33 ...
Validation Accuracy = 0.986

EPOCH 34 ...
Validation Accuracy = 0.988

EPOCH 35 ...
Validation Accuracy = 0.989

EPOCH 36 ...
Validation Accuracy = 0.990

EPOCH 37 ...
Validation Accuracy = 0.988

EPOCH 38 ...
Validation Accuracy = 0.981

EPOCH 39 ...
Validation Accuracy = 0.985

EPOCH 40 ...
Validation Accuracy = 0.985

EPOCH 41 ...
Validation Accuracy = 0.982

EPOCH 42 ...
Validation Accuracy = 0.986

EPOCH 43 ...
Validation Accuracy = 0.986

EPOCH 44 ...
Validation Accuracy = 0.989

EPOCH 45 ...
Validation Accuracy = 0.991

EPOCH 46 ...
Validation Accuracy = 0.991

EPOCH 47 ...
Validation Accuracy = 0.990

EPOCH 48 ...
Validation Accuracy = 0.992
```

```
EPOCH 49 ...
Validation Accuracy = 0.992

EPOCH 50 ...
Validation Accuracy = 0.992

Model saved
Test Accuracy = 0.931
```

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.

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

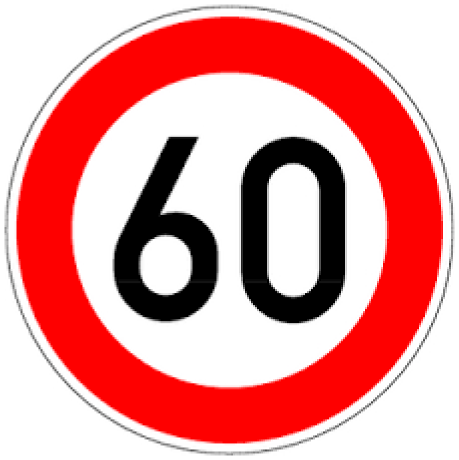


Illustration 1: 1 Keep 60

1.Keep 60: For this image I believe the number 60 will be difficult to classify, since there are other speed limit signs like Speed limit 30, Speed limit 80 and so on. So, the classifier might find Speed limit 80 instead of Speed limit 60

Illustration 2: 2 No Vehicle

2**.No Vehicle:** This image seems easy to me to classify, but the problem can come after converting to gray scale because the gray scale image will be similar to gray scale image of Priority road traffic sign

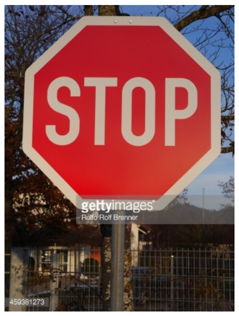


Illustration 3: 3 Stop sign

3. Stop: The problem with this image is the extra background that is there. In addition to STOP sign there us a house at the back and so are the trees. The image also doesnot seem bright

Illustration 4: Traffic signal

4. Traffic Signal: I don't see any problem in classfying this one, should be straightforward after converting to gray scale.

Illustration 5: Turn Ahead Left

5. Turn Ahead left: May confuse with “Turn ahead right”, or “Ahead only”

2. Discuss the model's predictions on these new traffic signs and compare the results to predicting on the test set. Identify where in your code predictions were made. 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).

See IN [15] in the jupyter notebook to see the prediction made for the above 5 traffic signs.

Here are the results of the prediction for following hyper parameter:

EPOCHS = 20

BATCH_SIZE = 128

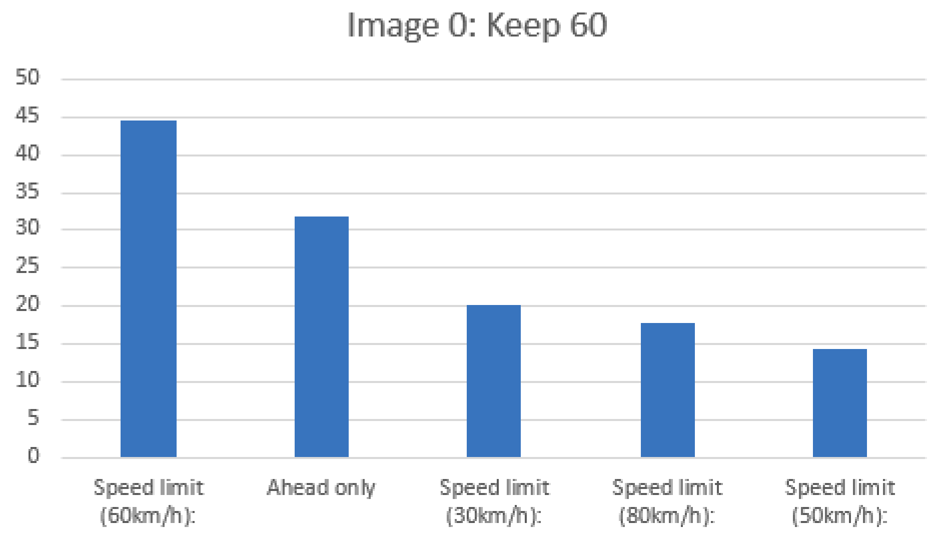
rate = 0.001

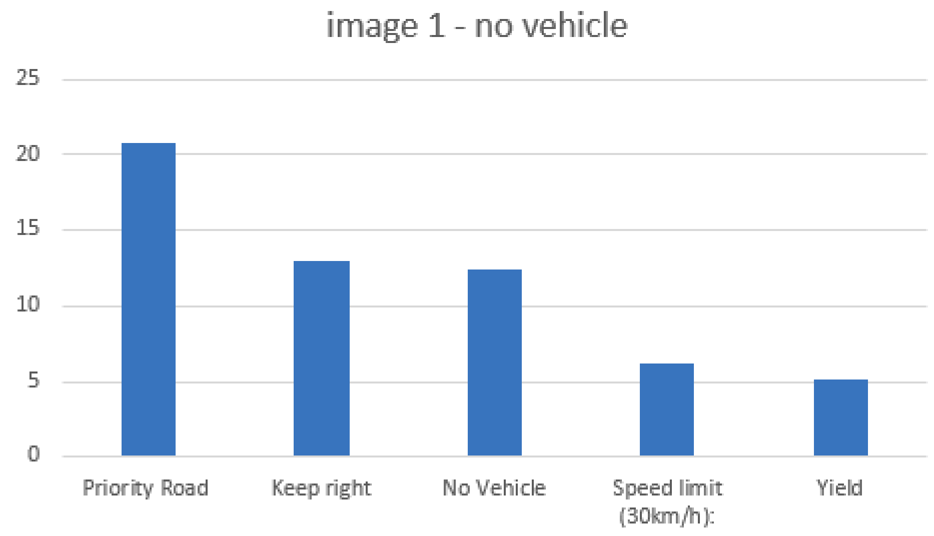
```
Image 0 – keep 60

Speed limit (60km/h): 44.604
Ahead only: 31.805%
Speed limit (30km/h): 20.215
Speed limit (80km/h): 17.762
Speed limit (50km/h): 14.397

Image 1 – no vehicle
Priority road: 20.742
Keep right: 13.025
No vehicles: 12.416
Speed limit (30km/h): 6.202
Yield: 5.162

Image 2 – stop
Speed limit (60km/h): 13.390
Children crossing: 6.459
No passing: 3.678
Bicycles crossing: 1.796
Speed limit (100km/h): -0.425

Image 3 – traffic signals
Traffic signals: 60.830
General caution: 50.463
```

```
Children crossing: 22.021
Dangerous curve to the right: 21.220
Road narrows on the right: 19.328

Image 4 – turn ahead left
Turn left ahead: 39.975
Speed limit (30km/h): 12.438
Right-of-way at the next intersection: 9.985
Keep right: 8.124
End of all speed and passing limits: 6.650

Test Accuracy = 0.600
```

For the above hyper parameters, the model was able to correctly guess 3 of the 5 traffic signs, which gives an accuracy of 60%. **This is lesser compared to the testing accuracy (90%) on the testing dataset set for the above hyper parameters, which shows that the model is overfitting.**

The above results have been shown in the bars below:





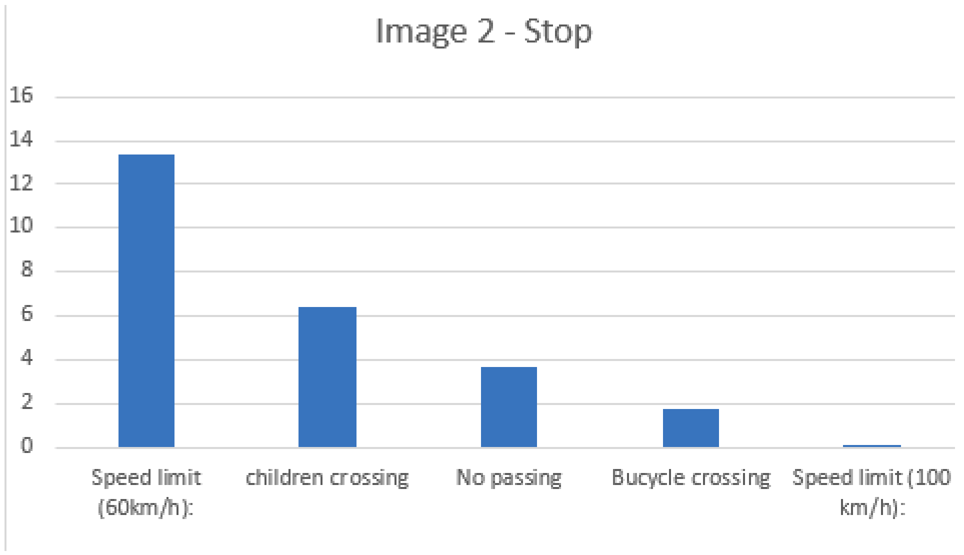


Image 2 - Stop
16
14
12
10
8
6
4
2
0
Speed limit (60km/h):
children crossing
No passing
Bucycle crossing
Speed limit (100 km/h):

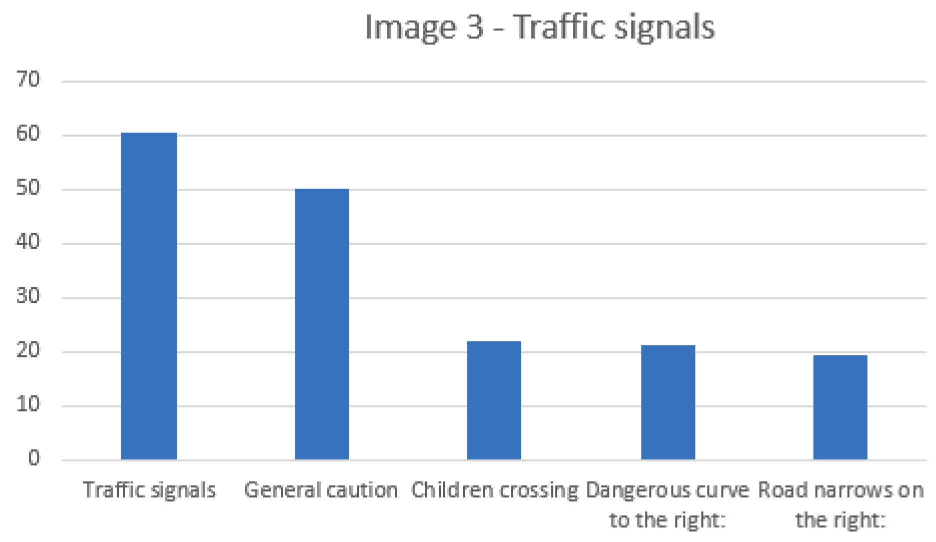
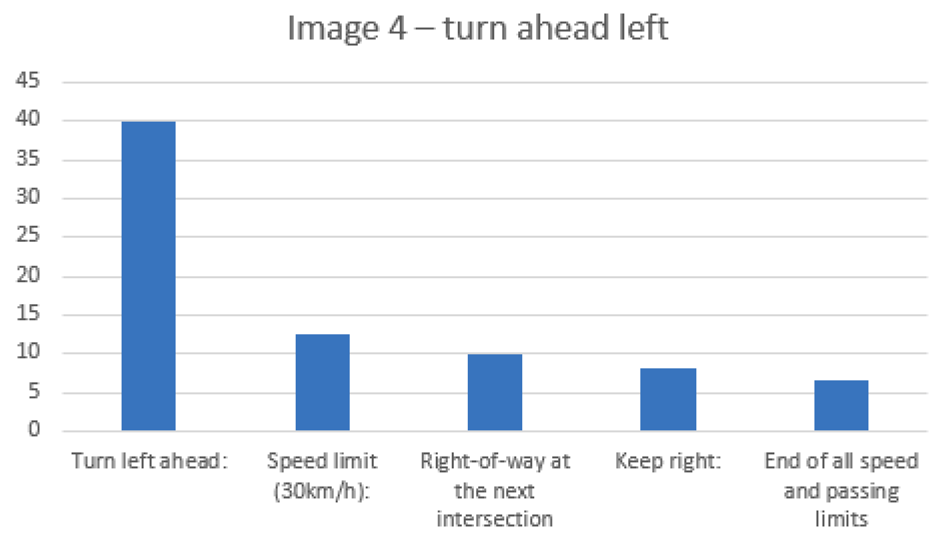


Image 3 - Traffic signals
70
60
50
40
30
20
10
0
Traffic signals
General caution
Children crossing
Dangerous curve to the right:
Road narrows on the right:



```
===========================================================
===========================================================
```

Here are the results of the prediction for following hyper-parameter:

EPOCHS = 50

BATCH_SIZE = 128

rate = 0.001

```
Image 0– keep 60

Speed limit (50km/h): 64.595
Wild animals crossing: 45.531
Speed limit (30km/h): 40.190
Speed limit (60km/h): 28.408
Bicycles crossing: 16.753

Image 1 - no vehicle
Priority road: 19.091
No vehicles: 9.983
```

```
Keep right: 4.999
Roundabout mandatory: 2.037
Speed limit (30km/h): 1.687

Image 2 – stop
No passing: 36.026
Slippery road: 25.793
No passing for vehicles over 3.5 metric tons: 16.017
Dangerous curve to the left: 5.613
Yield: 1.679

Image 3 – traffic signals
Traffic signals: 82.397
General caution: 41.115
Go straight or left: 12.258
Speed limit (20km/h): 1.178
Turn right ahead: 0.815

Image 4 turn ahead left
Turn left ahead: 52.396
End of all speed and passing limits: 31.632
Keep right: 27.363
No vehicles: 12.164
Speed limit (20km/h): 0.148

Test Accuracy = 0.400
```

For the above hyper parameters, the model was able to correctly guess 2 of the 5 traffic signs, which gives an accuracy of 40%. **This is lesser compared to the testing accuracy (93.1%) on the testing dataset for the above hyper parameters, which shows that the model is overfitting.**