Traffic Sign Recognition

Build a Traffic Sign Recognition Project

The goals / steps of this project are the following:

- Load the data set (see below for links to the project data set)
- Explore, summarise and visualise the data set
- Design, train and test a model architecture
- Use the model to make predictions on new images
- Analyse the softmax probabilities of the new images
- Summarise 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.

WRITEUP / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. You can use this template as a guide for writing the report. The submission includes the project code.

You're reading it! And my project code is contained within the same zip file that includes this write up.

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.

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

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

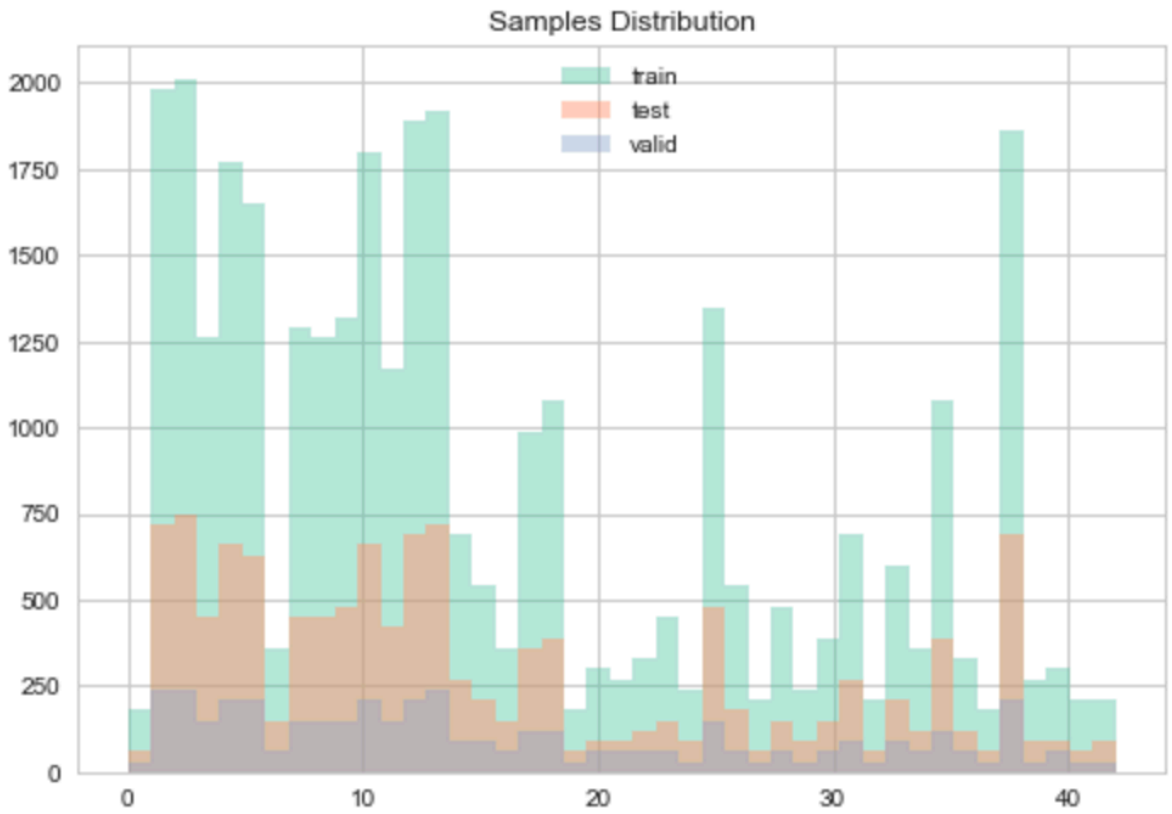
- The size of training set is 34799
- The size of test set is 12630
- The shape of a traffic sign image is 32x32x3
- The number of unique classes/labels in the data set is 43

2. Include an exploratory visualisation of the dataset and identify where the code is in your code file.

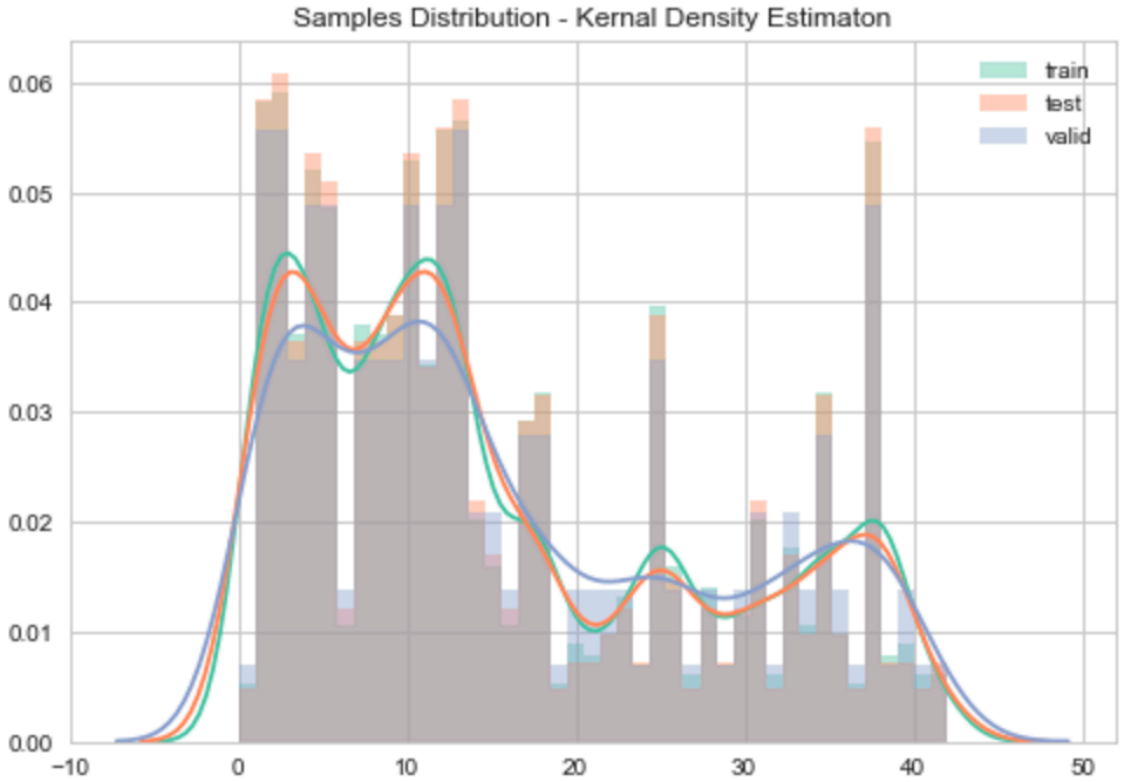
The code for this step is contained in the third to sixth code cells of the IPython notebook.

Here is an exploratory visualisation of the data set.

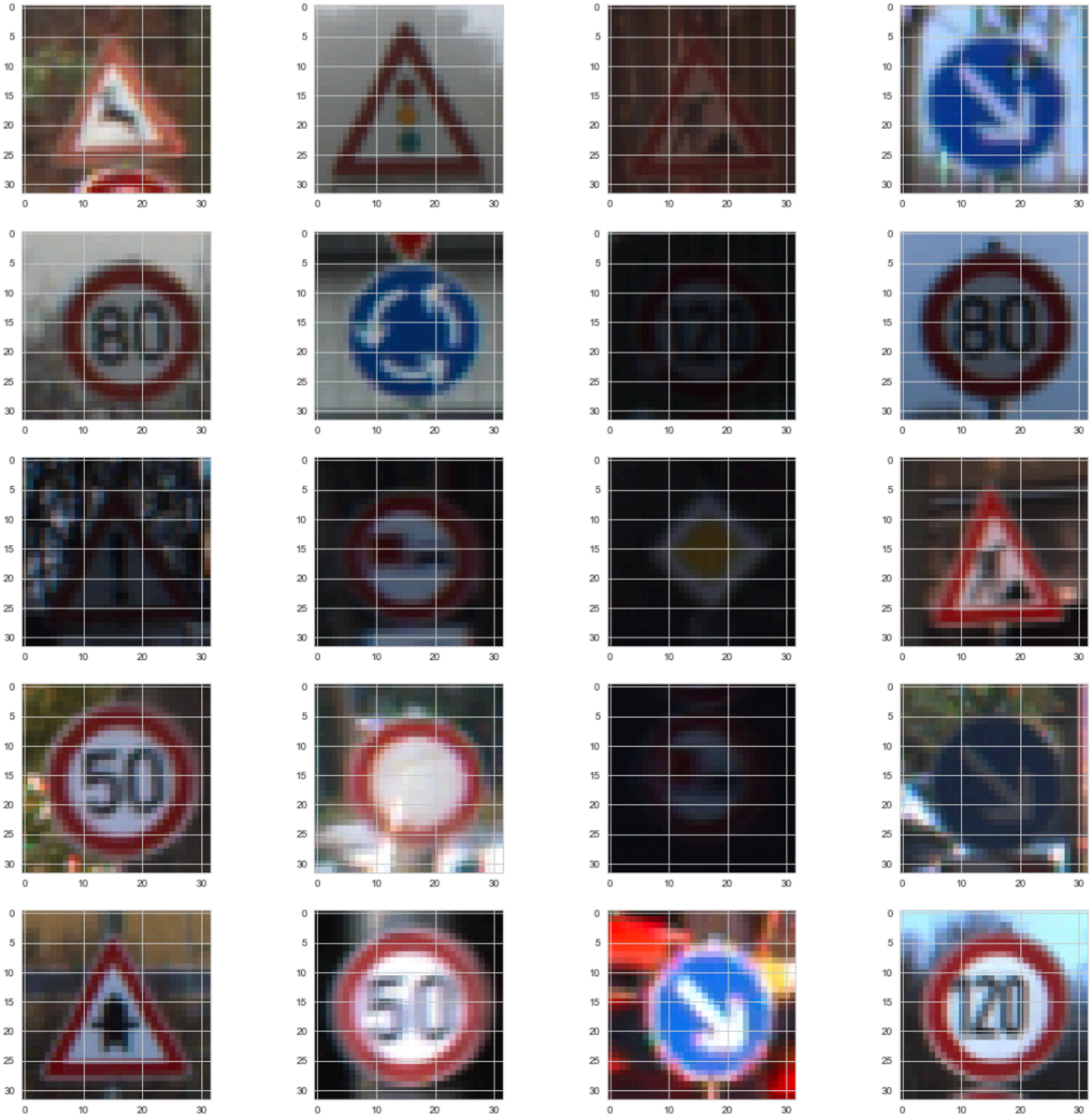
2.1 A samples distribution of the train, test, and validation labels:



2.2 A samples distribution with kernel density estimation for the train, test and validation labels:



2.3 A random sample set of 20 images from the training set:



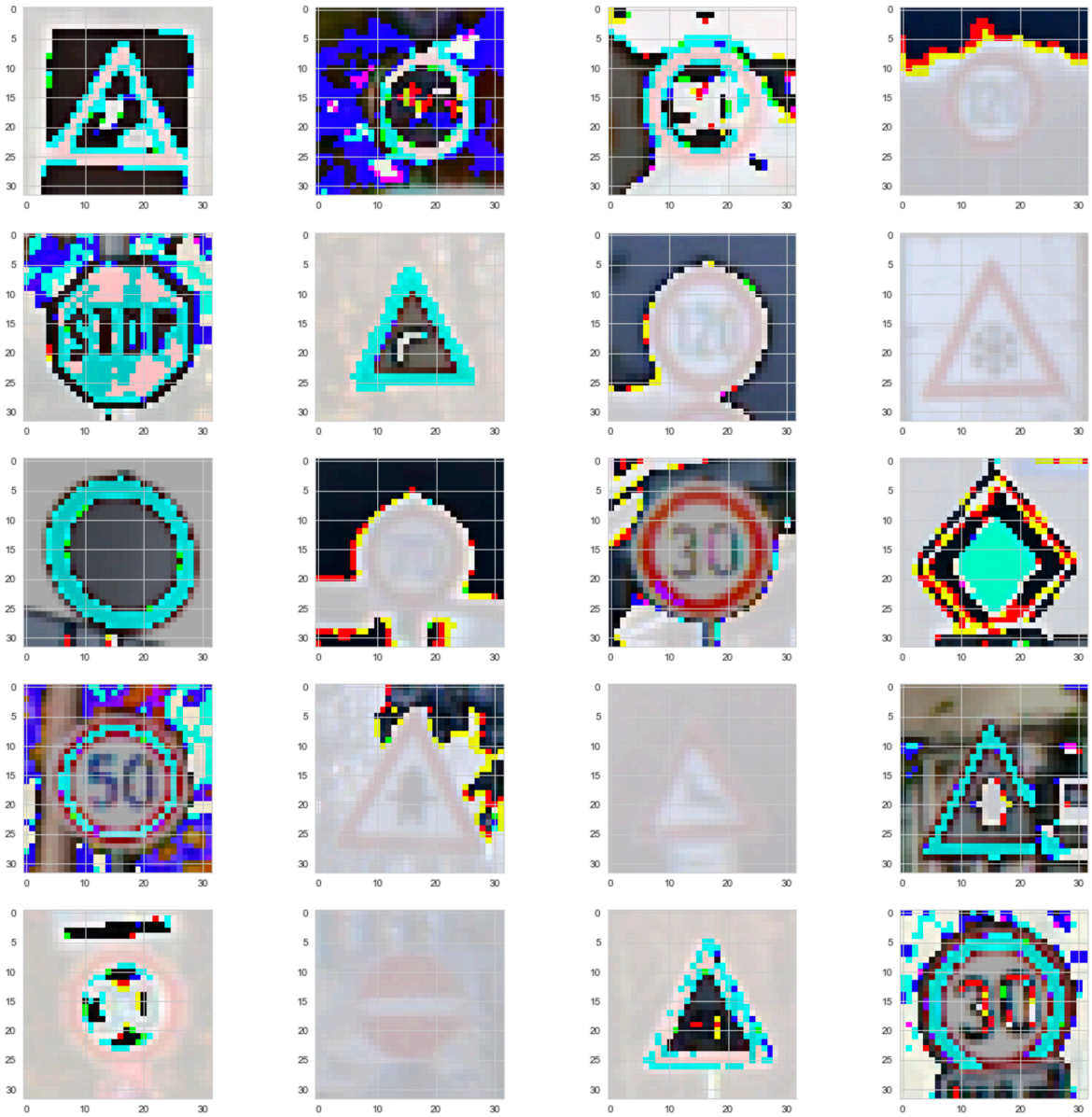
DESIGN AND TEST A MODEL ARCHITECTURE

1. Describe how, and identify where in your code, you preprocessed the image data. What techniques 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, normalisation, etc.

The code for this step is contained in the seventh code cell of the iPython notebook.

I decided to normalise the dataset as it gave me much greater accuracy when training the data.

1.1 The random sample set of 20 images from the training set when normalisation has been applied:



Please note, I also tried converting to grayscale however this reduced my training accuracy so I chose not to keep it in my pre-processing steps.

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.

The code for splitting the data into training and validation sets is contained in the first code cell of the iPython notebook.

To cross validate my model, I chose to use the training data file and validation data file. I did this by doing the following:

```
data_directory = './traffic-signs-data/'
training_file = data_directory + 'train.p'
validation_file= data_directory + 'valid.p'
testing_file = data_directory + 'test.p'

with open(training_file, mode='rb') as f:
    train = pickle.load(f)
with open(validation_file, mode='rb') as f:
    valid = pickle.load(f)
with open(testing_file, mode='rb') as f:
    test = pickle.load(f)

X_train, y_train = train['features'], train['labels']
X_valid, y_valid = valid['features'], valid['labels']
X_test, y_test = test['features'], test['labels']
```

My final training set had 34799 number of images. My validation set and test set had 4410 and 12630 number of images.

Additionally, I chose to shuffle the training data, this can be seen in code block 9 of the iPython notebook.

3. Describe, and identify where in your code, 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.

The code for my final model is located in the seventh cell of the iPython notebook.

My final model consisted of the following layers:

Layer	Layer Construct	Description
Layer 1	Convolutional	Input = 32x32x3. Output = 28x28x6
	Activation	relu
	Pooling	Input = 28x28x6. Output = 14x14x6
Layer 2	Convolutional	Output = 10x10x16
	Activation	relu
	Pooling	Input = 10x10x16. Output = 5x5x16
	Flatten	Input = 5x5x16. Output = 400
Layer 3	Fully Connected	Input = 400. Output = 120.
	Activation	relu
Layer 4	Fully Connected	Input = 120. Output = 84.
	Activation	
Layer 5	Fully Connected	Input = 84. Output = 43.

4. Describe how, and identify where in your code, you trained your model. The discussion can include the type of optimiser, the batch size, number of epochs and any hyperparameters such as learning rate.

The code for training the model is located in the eleventh and fourtenth cell of the iPython notebook.

To train the model, after a process of trial and error with the hyper-parameters, I finally chose the following as they gave the best performance and accuracy:

Type of optimiser: AdamOptimizer()

An algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments[1]

Epochs: 50

Learning Rate: 0.001

Batch Size: 128

5. 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.

The code for calculating the accuracy of the model is located in the ninth cell of the iPython notebook.

My final model results were:
- Training set accuracy of 1.000 (100%)
- Validation set accuracy of 0.942 (94.2%)
- Test set accuracy of 0.931 (93.3%)

Parameters tuned included the learning rate and the number of epochs. Initial I had set the learning rate much smaller, values around the 0.0001 mark and my accuracy was much lower. I finally settled on 0.001 and this made the rate of learning much quicker and allowed a much better accuracy. As for the number of epochs, I tried much higher numbers for very little gain as the additional time the increase gave did not help the accuracy. I finally settled on 100 as this gave enough time for the learning to get to its optimum accuracy.

If a well known architecture was chosen:
- I chose to use the LeNet architecture[2]
- I believed it would be relevant as it was used also for training on large image datasets
- The model's final accuracy provides good support that the correct architecture was chosen, as the accuracy is good.

---

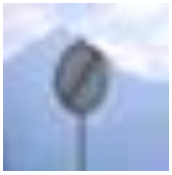
[1] Adam: A Method for Stochastic Optimization: https://arxiv.org/abs/1412.6980

[2] Lecun: http://yann.lecun.com/exdb/lenet/

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:



32 - End of all speed and passing limits

36 - Go straight or right (this is a best intuitive guess as the actually sign is ‘do not turn left’, however this is not included in the original traffic sign images csv)

Therefore, this image might be difficult to classify because it was not in the original dataset.

4 - Speed limit (70km/h)

40 - Roundabout mandatory

14 - Stop

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

The code for making predictions on my final model is located in the twentieth and twenty-first cell of the iPython notebook. Here are the results of the prediction:

The model was not able to correctly guess any of the 5 traffic signs, which gives an accuracy of 0.000 (0%). This of course does not compare favourably to the accuracy on the test set of 0.942 (94.2%).

It looks like I would need to add even more examples of these types of traffic signs in order for the model to become better at predicting similar images. On another iteration I will also look to crop the

new images more tightly so that more of the sign is prominent in the image to see if this makes a difference.
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 and identify where in your code softmax probabilities were outputted. Provide the top 5 softmax probabilities for each image along with the sign type of each probability.

The code for making predictions on my final model is located in the twenty-second cell of the ipython notebook.

For the first image, the model is slightly more sure that this is a roundabout mandatory (probability of 0.35), and the image does not contain a stop sign. However, the next highest probability (0.17) is for end of speed limit, which is relatively close as the end of speed limit signs also have the black diagonal strips on them. The top five soft max probabilities were:

Probability	Prediction
0.3521353	**40 - Roundabout mandatory**
0.17273335	6 - End of speed limit (80km/h)
0.15927175	5 - Speed limit (80km/h)
0.15833339	12 - Priority road
0.15752624	1 - Speed limit (30km/h)

For the second image, the model is fairly sure that this is a speed limit sign (probability of 0.40), and the image does not contain a speed limit sign. The top five soft max probabilities were:

Probability	Prediction
0.40460932	**1 - Speed limit (30km/h)**
0.14884785	4 - Speed limit (70km/h)
0.14884765	12 - Priority road
0.14884765	8 - Speed limit (120km/h)
0.14884765	38 - Keep right

For the third image, the model is more sure that this is a turn right ahead sign (probability of 0.6), and the image does not contain a turn right ahead sign. The top five soft max probabilities were:

Probability	Prediction
0.2830826	**33 - Turn right ahead**
0.2413003	26 - Traffic signals
0.15854041	30 - Beware of ice/snow
0.15853864	39 - Keep left
0.15853807	37 - Go straight or left

For the fourth image, the model is more sure that this is a speed limit sign (probability of 0.6), and the image does not contain a speed limit sign. However, the correct sign is within the top 5 probabilities, with a probability of 0.16. The top five soft max probabilities were:

Probability	Prediction
0.31536779	**2 - Speed limit (50km/h)**
0.21144035	5 - Speed limit (80km/h)
0.15862012	**40 - Roundabout mandatory**
0.15728587	1 - Speed limit (30km/h)
0.15728585	11 - Right-of-way at the next intersection

For the fifth image, the model is relatively sure that this is a yield sign (probability of 0.6), and the image does contain a yield sign. The top five soft max probabilities were:

Probability	Prediction
0.40460971	**13 - Yield**
0.14884759	14 - Stop
0.14884759	33 - Turn right ahead
0.14884759	26 - Traffic signals
0.14884759	15 - No vehicles