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, 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

Image 1: Train Data Set Distinct Counts

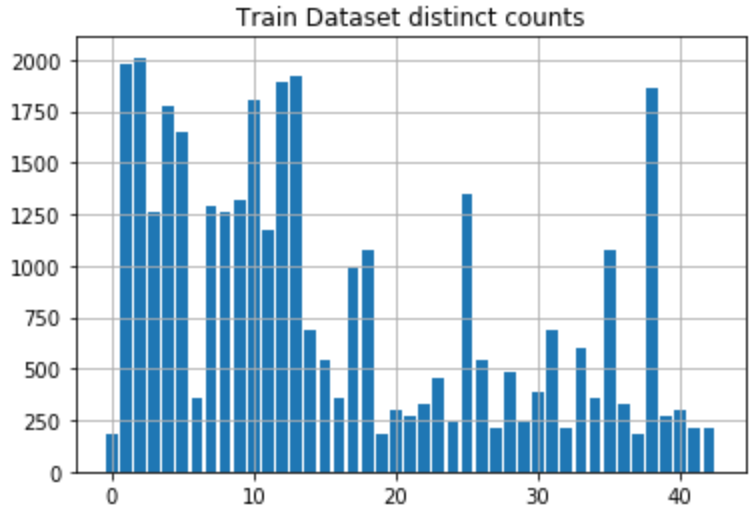


Image 2 : Valid Data Set Counts

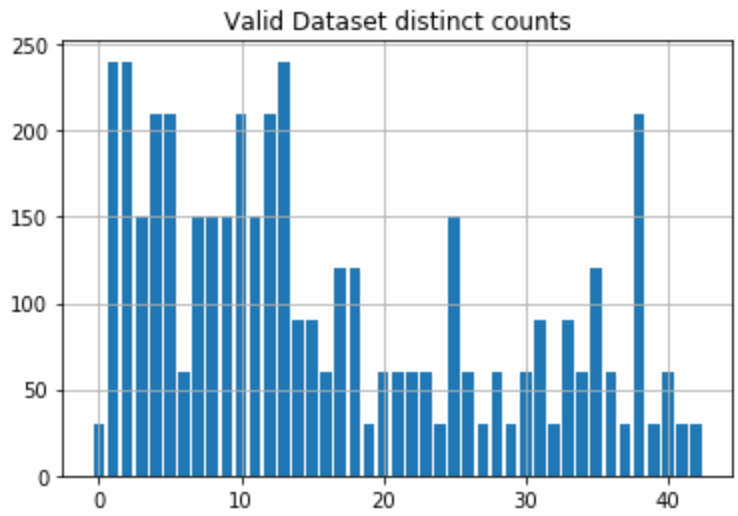


Image 3: Train Data Set Counts

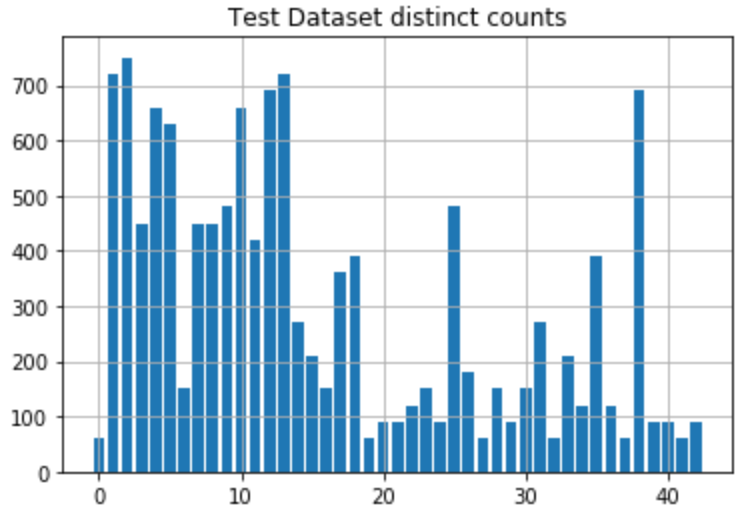


Image 4 Exclaim

Image 5 No Entry

Image 6 Roadwork

Image 7 Speed Limit

Image 8 Stop Sign

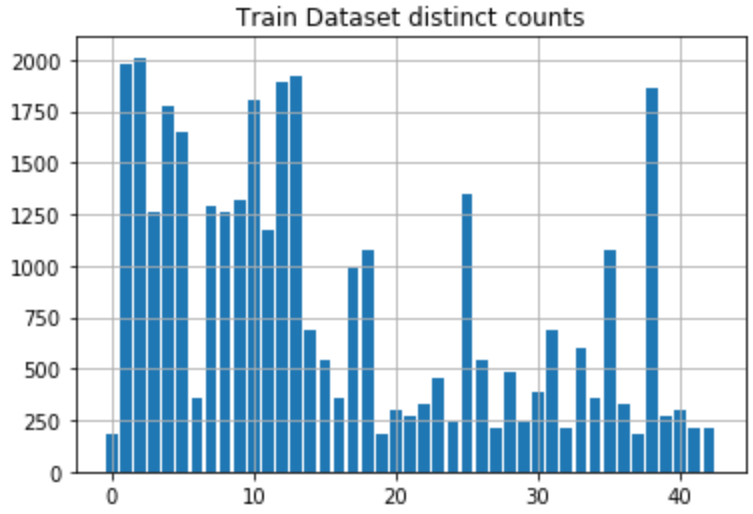
The project files are included in the zip file as ipynb and as exported html files

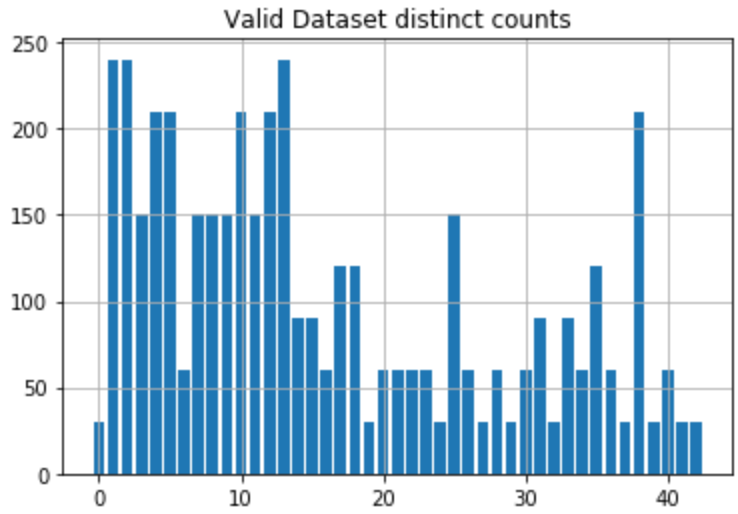
Data Set Summary & Exploration

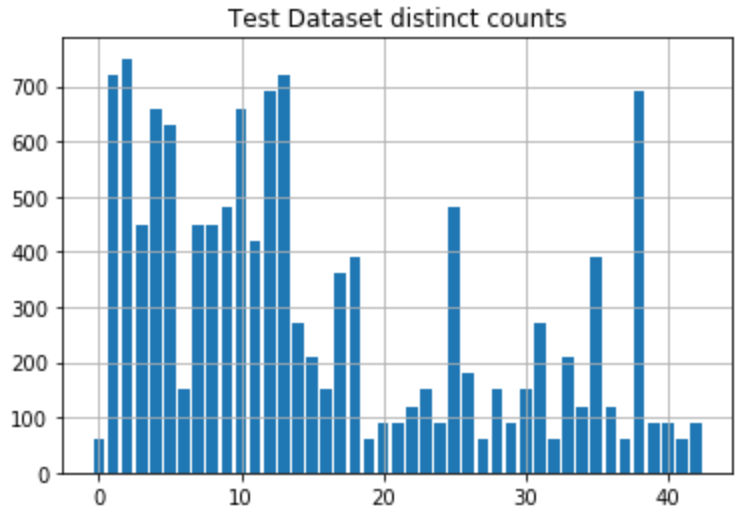
I used python and numpy to calculate summary statistics of the traffic
signs data set:

* The size of training set is 34799
* The size of the validation set is 4410
* The size of test set is 12630
* The shape of a traffic sign image is 32,32,3
* The number of unique classes/labels in the data set is 43

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







Data Pre Processing

I preprocessed the data in 3 ways to avoid a perfect world input for the neural network

Real world image normalizations I did
1. Converted to gray scale as color does not "add" additional domain information to what the neural network needs to perceive, for our domain, this is just noise, this helps reduce the input features, so that we can focus on the required output features

2. Normalized the input image pixel values to a zero mean by scaling down range from 0-255 to -128 to +128 and then further divding by 128 to avoid extremely high values and ranges, now the normalized range of input data is -1.0 to 1.0 with a mean of 0

3. I did image image augmentation again as real world images are not perfect. I did 3 augmentations to add to the input image training data

- I did image translation as when the car moves, the image will not be in same camera frame always (used a random scale for translation)
- I did image rotation (using a random amount for rotation), same reason, image is not always "perfect picture", camera mounted is not always upright with ground below as road/moving surface is not perfectly flat
- I did image perspective transformation, again for the above said reasons

Final Model Architecture

After image normalization, I had images of shape 32,32,1
this is the input the processing model/architecture

My final model consisted of the following layers:
The model was pretty much the Le net architecture, except I changed the number of featues to a higher value because of in this paper
http://yann.lecun.com/exdb/publis/pdf/sermanet-ijcnn-11.pdf, they did so with outstanding results
Here they are using 100+ features at each stage, so I decided to try this and processed more features

Layer	Description	Notes
Input	32 x 32 x 1	After preprocess to grey
Convolution 5x5x20	1x1 stride, no padding	Output 28x28x20
RELU		

Max Pooling	2x2 Stride	Output 14x14x20
Convolution 5x5x36	1x1 stride no padding	Output 10x10x36
RELU		
Max Pooling	2x2 Stride	Output 5x5x36
Flatten	Output 900 (5 x5x 36)	
Fully Connected	Input 900 output 300	
RELU		
Dropout	keep prob 0.5/1,0 (train/valid)	
Fully Connected	Input 300 output 84	
RELU		
Dropout	keep prob 0.5/1,0 (train/valid)	
Fully Connected	Input 84, output 43	

3. **Describe how you trained your model.**

To train the model
I used a learning rate of 0.000875 so that we do not hit local minima
I used epoch 13 to 14 as I found out during training , over 14 epochs, learning does not improve, so to prevent overfitting I stopped at 14 epochs. I also used dropouts in the model to prevent overfitting.
The batch size of around 162 has given good results
I stuck with the Adam optimizer as it gave good results

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.**

My final model results were:
* training set accuracy of 0.999
* validation set accuracy of 0.980

* test set accuracy of 0.960 (wow!) , ran just once

If an iterative approach was chosen:
*** What was the first architecture that was tried and why was it chosen?**

I did a hybrid approach, I chose a well known architecture and then fine tuned it
I started with the Le Net architecture in the Lab exercise. I chose this architecture, as it had already proven its capabilities in winning a championship in recognizing traffic signs.

* **What were some problems with the initial architecture?**
It did not account for over fitting i.e. did not have drop outs so was training against a perfect world

* **How was the architecture adjusted and why was it adjusted? Typical adjustments could include choosing a different model architecture, adding or taking away layers (pooling, dropout, convolution, etc), using an activation function or changing the activation function. One common justification for adjusting an architecture would be due to overfitting or underfitting. A high accuracy on the training set but low accuracy on the validation set indicates over fitting; a low accuracy on both sets indicates under fitting.**

I then ran the architecture against the raw images provided and got a validation accuracy of 89%
As this was obviously low and did not meet the pass criteria, I started making iterative changes, the first step was to normalize the input data i.e. grayscale + mean normalization. This yielded a train accuracy of 93%. More room for improvement.

I then read the link to http://yann.lecun.com/exdb/publis/pdf/sermanet-ijcnn-11.pdf
which discussed using more features, so I fine tuned the architecture to increase the number of features to 16 from 6 and to 36 from 20. This gave a validation accuracy of 95%
Not there yet, so I did more image augmentation i.e. image translation, image rotation and image perspective changes. These augmented images were added to the XTrain and YTrain data sets and gave me validation accuracy around 97 o 98%. The training accuracy at this point was 99.9, so I knew I had reached the maximum

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

The number of features processed were changed, because the initial architecture had low number of features compared to whatwas published and tested in the paper http://yann.lecun.com/exdb/publis/pdf/sermanet-ijcnn-11.pdf

I also changed the epoch to 14 , to avoid over fitting and also lowered the learn rate to 0.000875 from 0.001 to avoid peaking too soon i.e. local minima

*** 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?**

Some of the choices were to add drop outs(I added 2 drop outs), so to compensate for that I added 1 more fully connected layer The final result was a stunning 96% on the test set

If a well known architecture was chosen:

*** What architecture was chosen?**

The LeNet architecture was chosen first

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

This is a very generic architecture that has already proven in the field and won accolades

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

The training accuracy was 99.9%,
The valid accuracy was 98%
and test set accuracy was 96.2%

####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:


gettyimages

####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).

Here are the results of the prediction:

Image	**Prediction**
Exclaim Warning	**100%**
No Entry	**100%**
Road Work	**98%**
30 km/h speed limit	**100%**
Stop Sign	**100%**

|

The model was able to correctly guess 5 of the 5 traffic signs, which gives an accuracy of 100%. This compares favorably to the accuracy on the test set of 98%

####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)

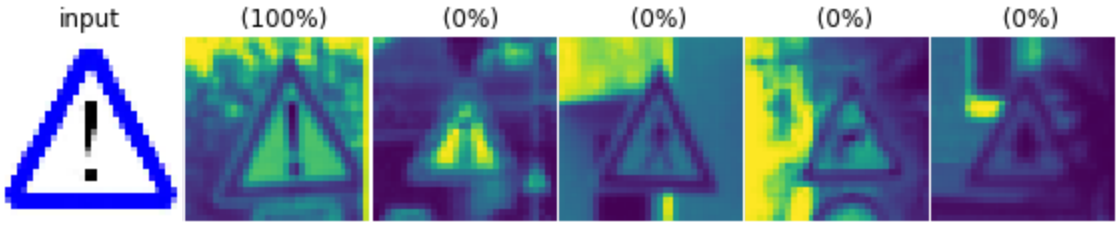
The code for making predictions on my final model is located in the 18th cell of the Ipython notebook.

For the first image, the model is relatively sure that this is a Exclaim Warning (probability of 1.0), and the image does contain a stop sign. The top five soft max probabilities were

Exclaim

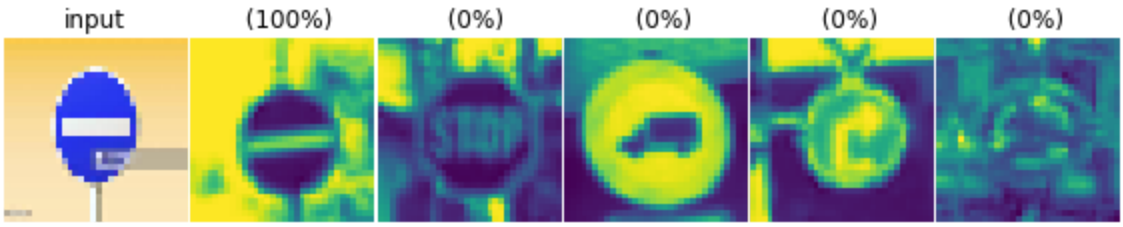
Probability	Prediction
1.0	Exclaim

0.0	Pedestrian 1
0.0	Pedestrian 2
0.0	Right Curve
0.0	Straight Arrow



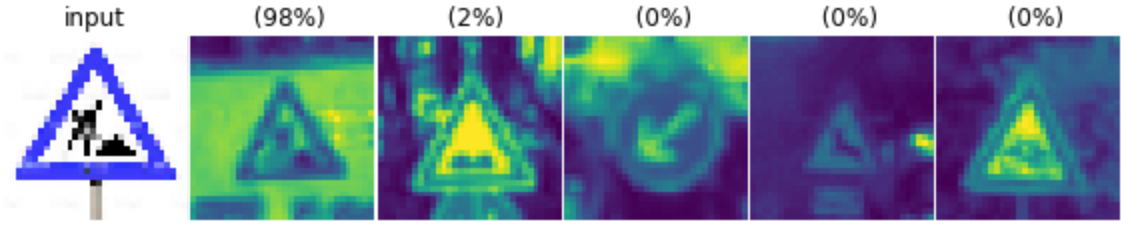
For the second image ... No Entry

Probability	Prediction
1.0	No Entry
0.0	Stop
0.0	Truck
0.0	Right curves
0.0	Round About



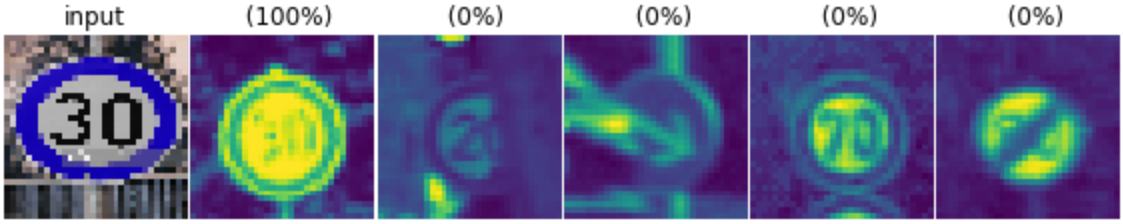
For the third image Road Work

Probability	Prediction
0.98	Road Work
0.02	Hump
0.00	Left corner arrow
0.00	Dog
0.00	Cycle

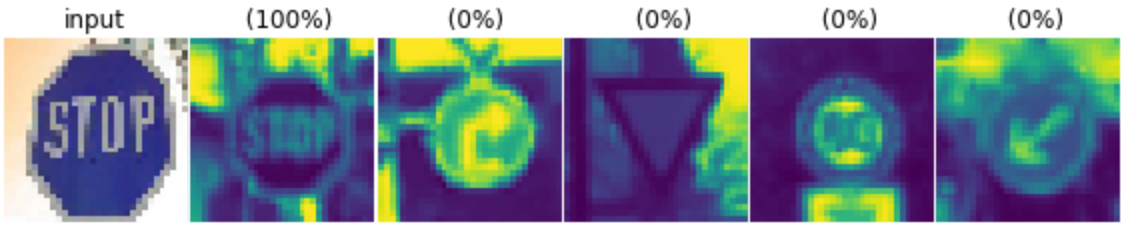


For the fourth image Speed Limit 30 km/h

Probability	Prediction
1.0	30 km/h
0.00	20 km/h
0.00	Right corner arrow
0.00	70 km/h
0.00	80 km/h with a strike through



Probablity	Prediction
1.00	Stop
0.00	2 Curves
0.00	2 curves
0.00	60
0.00	Left bottom corner arrow



(Optional) Visualizing the Neural Network (See Step 4 of the Ipython notebook for more details)

####1. Discuss the visual output of your trained network's feature maps. What characteristics did the neural network use to make classifications?