Traffic Sign Recognition

Stephan Brinkmann

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

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!

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 Python and NumPy for the calculations.

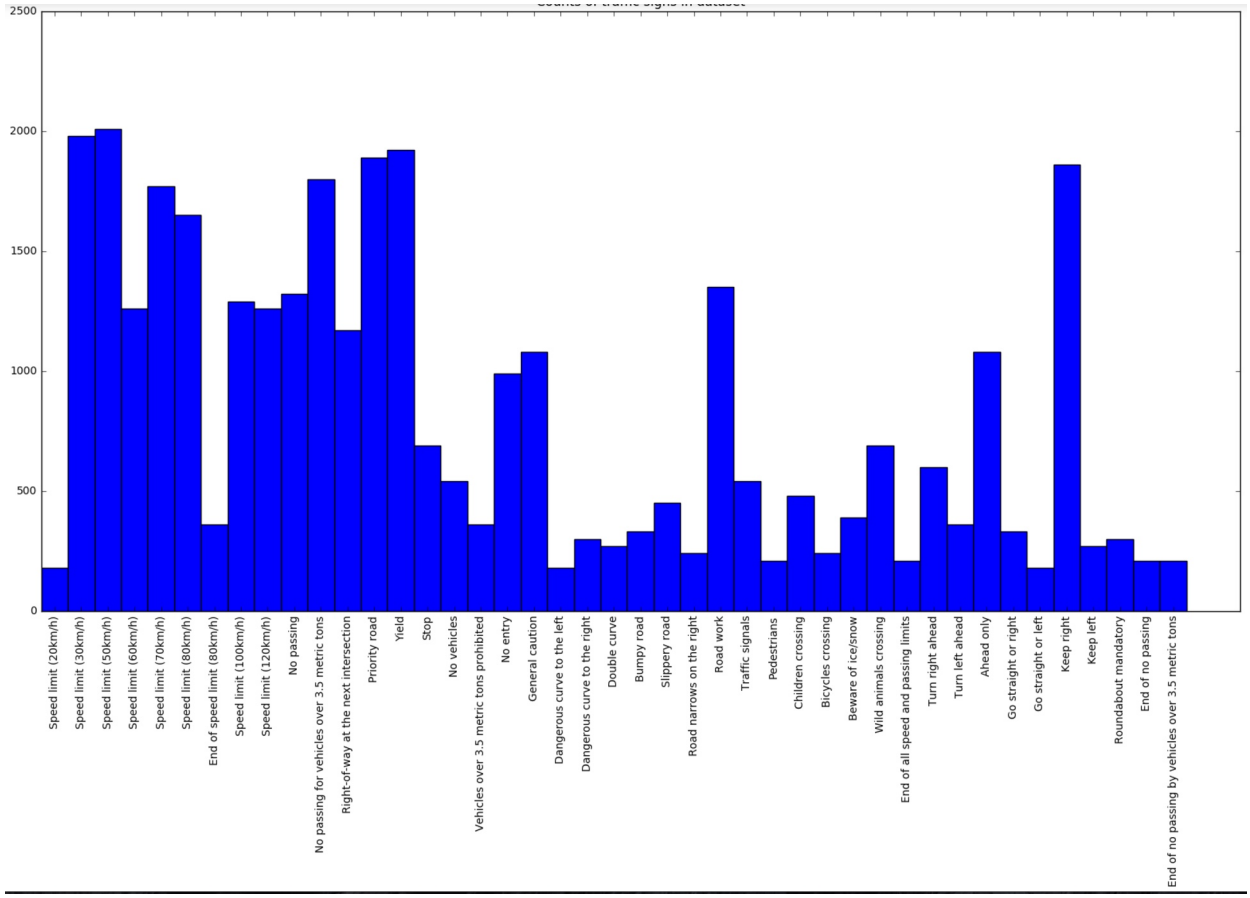
- Number of training examples = 34799
- Number of testing examples = 12630
- Number of validation examples = 4410
- Image data shape = (32, 32, 3)
- Number of classes = 43

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

The code for this step is contained in code cells 3–5.

Here's one random image from the dataset as an example, including the appropriate label loaded from the CSV file:

And here's a bar chart visualizing the numbers of images per traffic sign category in the data set:



Design and Test a Model Architecture

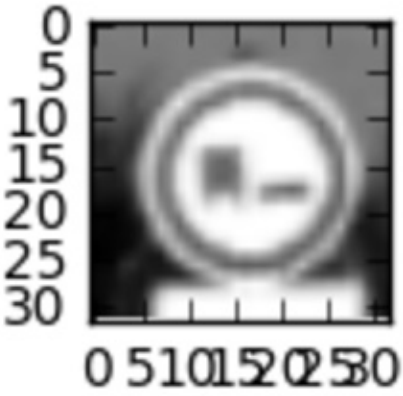
1. Describe how, and identify where in your code, you preprocessed the image data. What techniques were chosen and why did you choose these techniques? Consider including images showing the output of each preprocessing technique. Pre-processing refers to techniques such as converting to grayscale, normalization,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

The code for this step is contained in the sixth and seventh code cells of the IPython notebook.

I decided to go easy on the preprocessing and do only grayscale conversion and normalization (to values between 0.0 and 1.0).

Normalization was recommended throughout the lessons and in the Q&A on YouTube Live. Grayscale images are fast to train on, and don’t lose to many valuable information, but actually help the net focus on the more important features of traffic signs.

Here’s an example of a preprocessed image:



No passing for vehicles over 3.5 metric tons

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)

In the latest version of the project, the data comes already split, so no need to do that again.

In code cell 8, I simply shuffle the training data.

- Number of training examples = 34799
- Number of testing examples = 12630
- Number of validation examples = 4410

I decided not to augment the dataset, but I would have started by trying to balance the traffic sign categories, since there are huge differences, with some categories grossly underrepresented (see bar chart above).

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 ninth cell of the IPython notebook.

After some experiments with the LeNet architecture, I decided to play with an architecture similar to what's presented in the paper by Pierre Sermanet and Yann LeCun. Since the paper does not go into enough detail on specific layers, I implemented a crude mix of both architectures, trying to get the best of both.

My final model consisted of the following layers:

Layer	Description
Input	32x32x1 normalized grayscale image
Layer 1	
Convolution 3x3	1x1 stride, same padding, output: 32x32x12
RELU	
Max pooling	2x2 stride, output: 16x16x12
Layer 2	
Convolution 5x5	1x1 stride, valid padding, output: 12x12x16
RELU	
Max pooling	2x2 stride, output: 6x6x16
Dropout	50% keep chance
Layer 3	
Convolution 5x5	1x1 stride, valid padding, output: 2x2x32
RELU	
Max pooling	2x2 stride, output: 1x1x32
Dropout	50% keep chance
Concatenation	
Layer 1 max pooling & flatten	2x2 stride, output: 768
Layer 2 max pooling & flatten	2x2 stride, output: 144
Layer 3 flatten	Output: 32
Concatenate above 3 outputs	Output: 944
Layer 4	
Fully connected	Output: 472
Dropout	50% keep chance
Layer 5	
Fully connected / logits	Output: 43

For initializing the weights, I went with a Xavier initializer.

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

The code for training the model is located in cells 10–13 of the IPython notebook.

I used the standard Adam optimizer.

Hyperparameter	Description
Batch size	128–256
Epochs	A few 100 in total, usually around 100 at a time
Learning rate	0.0005, 0.001
L2 regularization rate	0.005, 0.01, 0.02
Dropout probabilities	50%

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 14th cell of the IPython notebook.

My final model results were:

- Training Accuracy = 1.000
- Validation Accuracy = 0.951
- Test Accuracy = 0.944

From the beginning, it was clear that some convolutional layers needed to be part of my architecture, since they allow it to adapt to different traffic sign locations within the images. Consequently, the first architecture I tried was the LeNet architecture from the lab, but I soon added some ideas from the Sermanet architecture, since it is a tried and tested model for traffic sign recognition. Most importantly, I decided to add the “skipping layers” concept from Sermanet.

I then trained Sermanet for a few dozen epochs on an i7, but quickly realized that it was overfitting (accuracy on training set at 100%, validation accuracy between 90–92%). The 100% training accuracy confirmed my architecture choice, so I decided to just add three

dropout layers (50% keep probability each) to combat overfitting. I also added L2 regularization (0.01) to prevent overfitting even more.

After over 200 epochs, the model seemed to max out at around 94% accuracy, still slightly overfitting, so I tuned the parameters again, increasing dropout and L2 rate even more.

This netted me the end result of 95.1% accuracy.

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.

Loading, outputting and preprocessing are done in code cells 15 and 16.

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

The first, second and last image should be pretty easy to classify, even though the last one is somewhat skewed, and the second one has a slight glare/reflection.

Number 3 and 4 are partially cut off, it remains to be seen how the model handles those. Number 3 is also in one of several speed limit categories, which might be hard to distinguish from one another.

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).

The code for making predictions on my final model is located in the 17th and 18th cell of the IPython notebook.

Here are the results of the prediction:

Image	Prediction	Certainty
Yield	Yield	100%
Dangerous curve to the left	Dangerous curve to the left	100%
Speed limit (30km/h)	Speed limit (30km/h)	77.89%
No passing	No passing	100%
Priority road	Priority road	100%

The model was able to correctly guess 5 of the 5 traffic signs, which gives it an accuracy of 100%. This compares favorably to the accuracy on the test set of 94.4%.

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. (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 19th and 20th cells of the IPython notebook.

Image 1

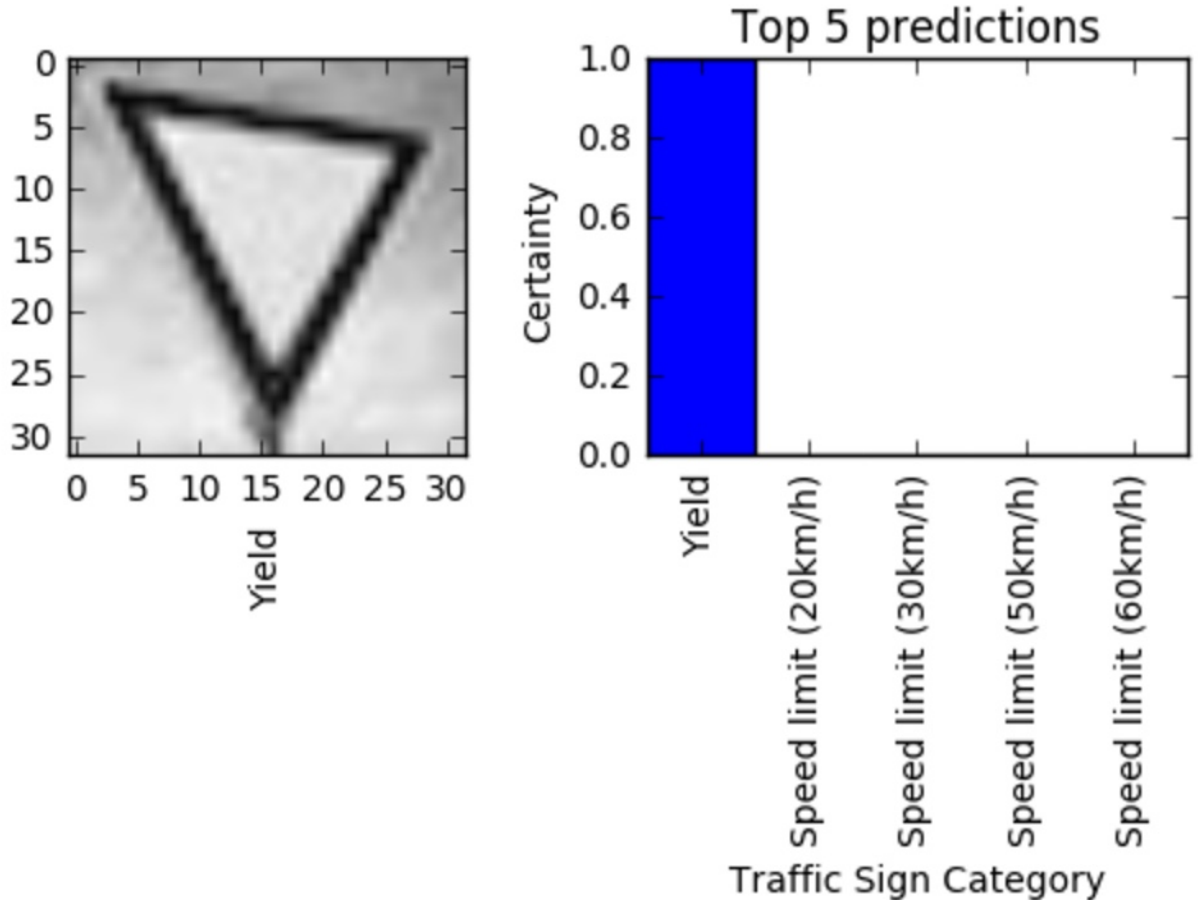

0
5
10
15
20
25
30
0 5 10 15 20 25 30
Yield
Top 5 predictions
1.0
0.8
0.6
0.4
0.2
0.0
Certainty
Yield
Speed limit (20km/h)
Speed limit (30km/h)
Speed limit (50km/h)
Speed limit (60km/h)
Traffic Sign Category

Image 2

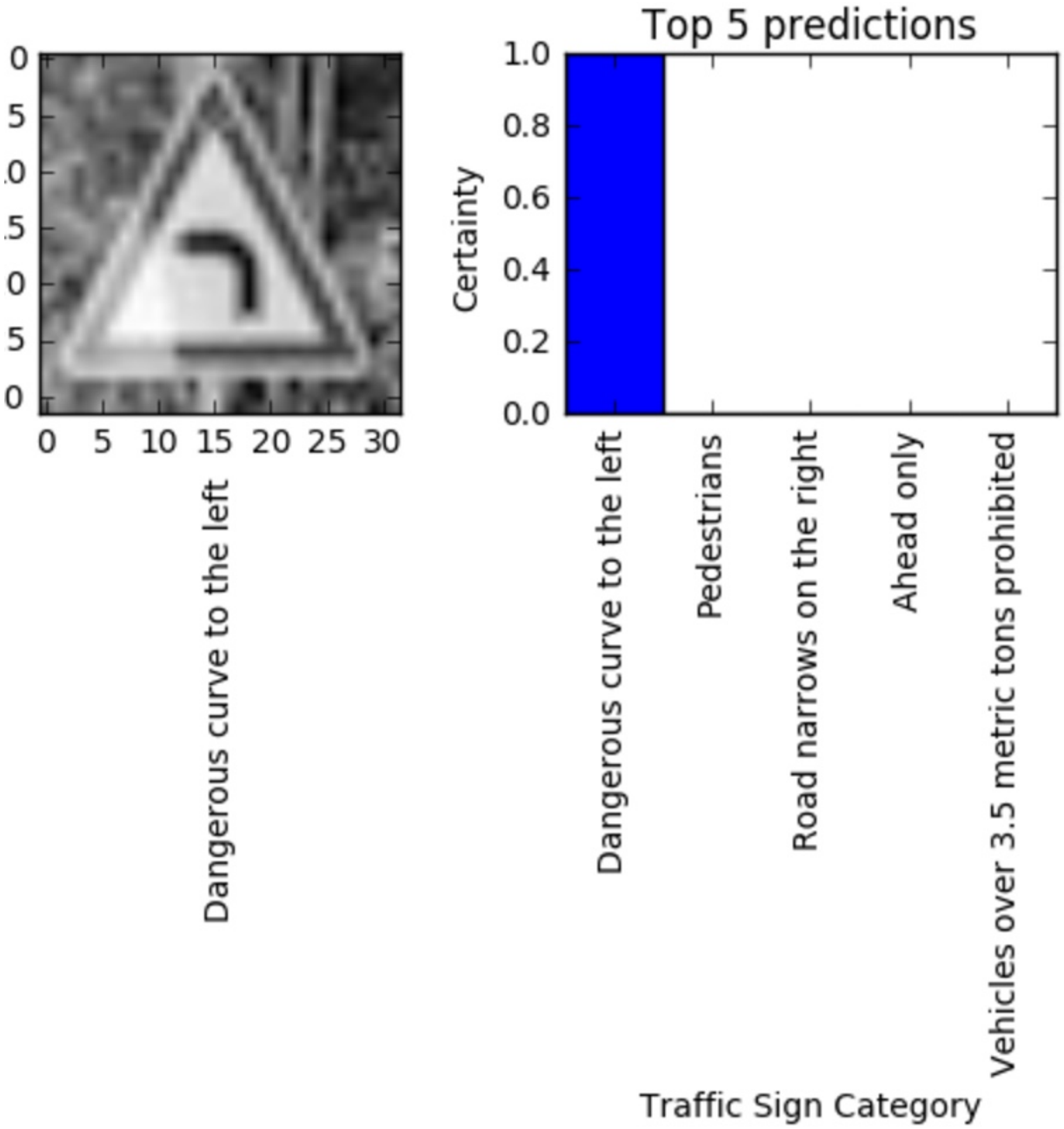
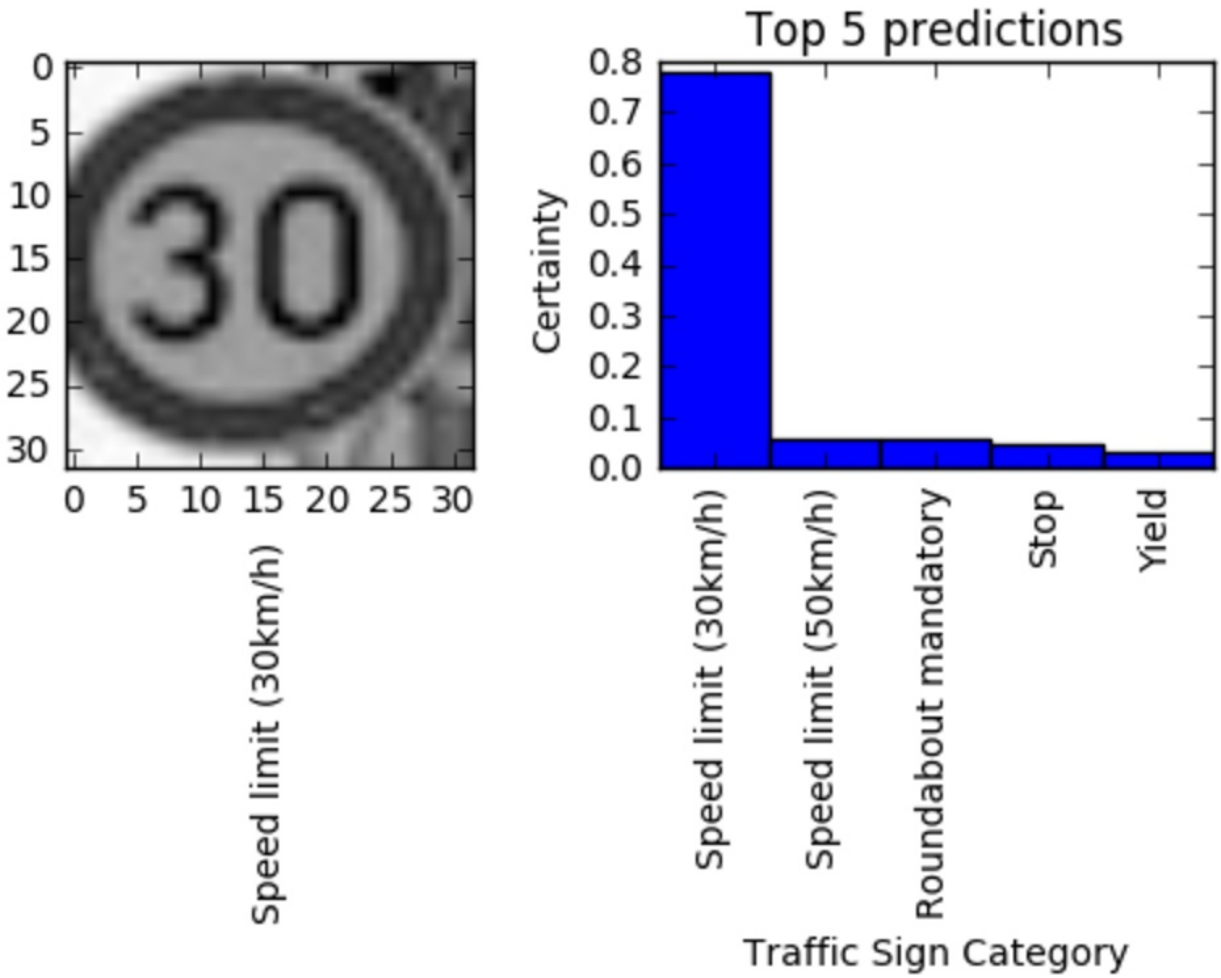

Top 5 predictions
Certainty
1.0
0.8
0.6
0.4
0.2
0.0
Dangerous curve to the left
Pedestrians
Road narrows on the right
Ahead only
Vehicles over 3.5 metric tons prohibited
Traffic Sign Category
0 5 10 15 20 25 30
Dangerous curve to the left

Image 3



For this image, the model's second guess (5.79%) is "Speed Limit (50 km/h)", which is very understandable. The other guesses seem less logical:

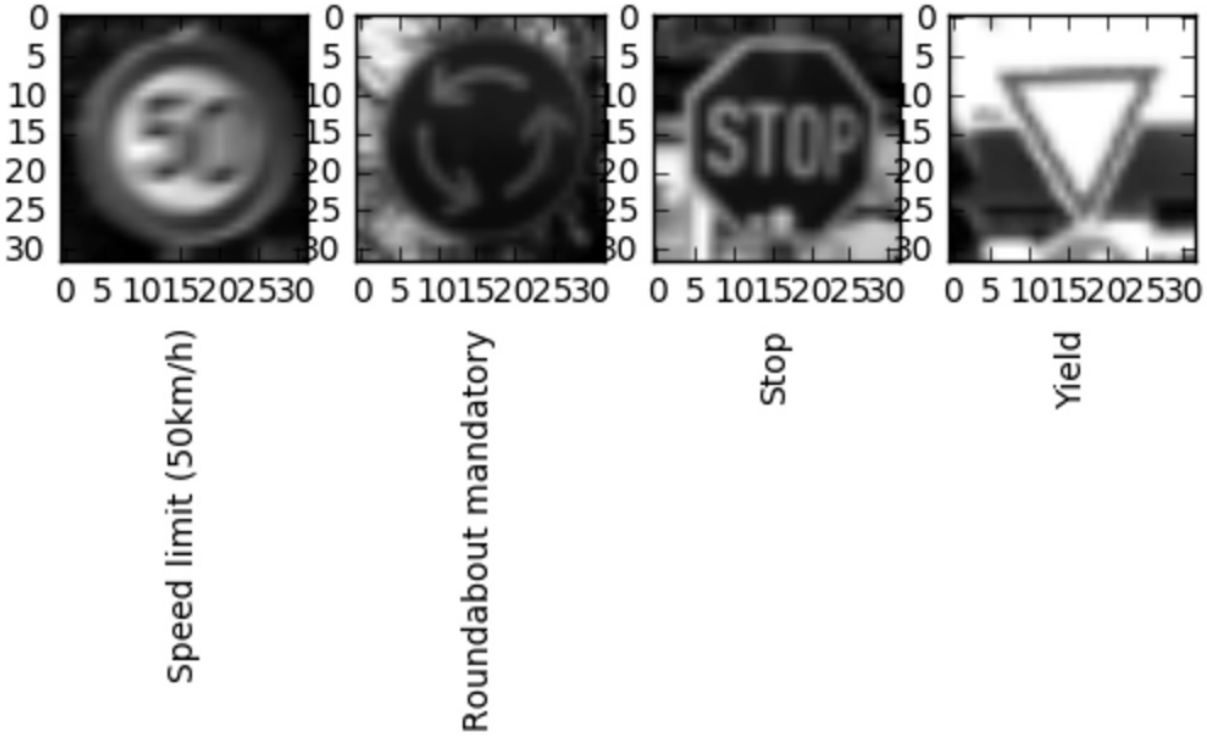


Image 4

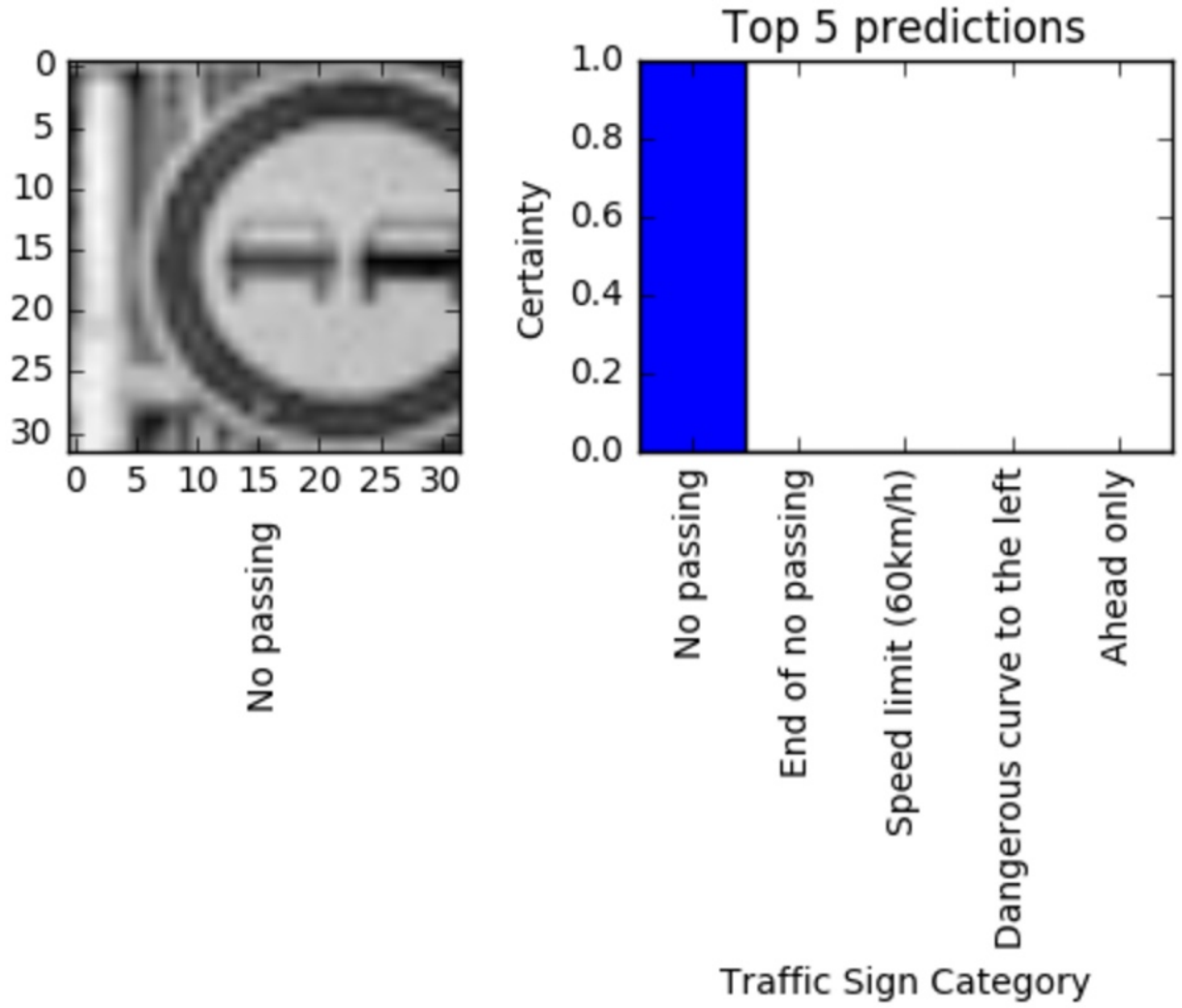
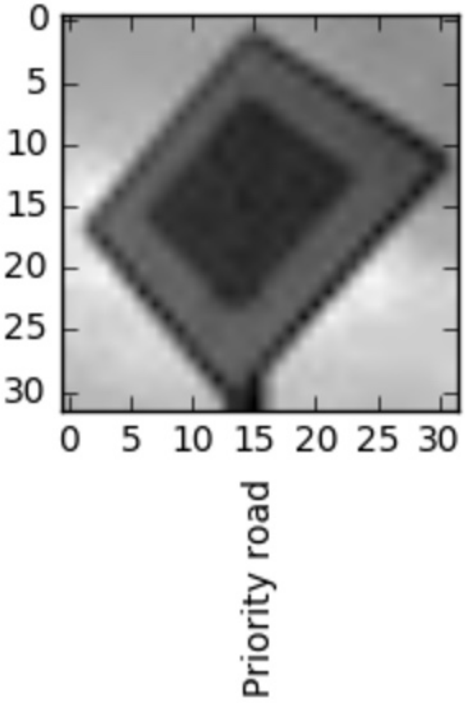
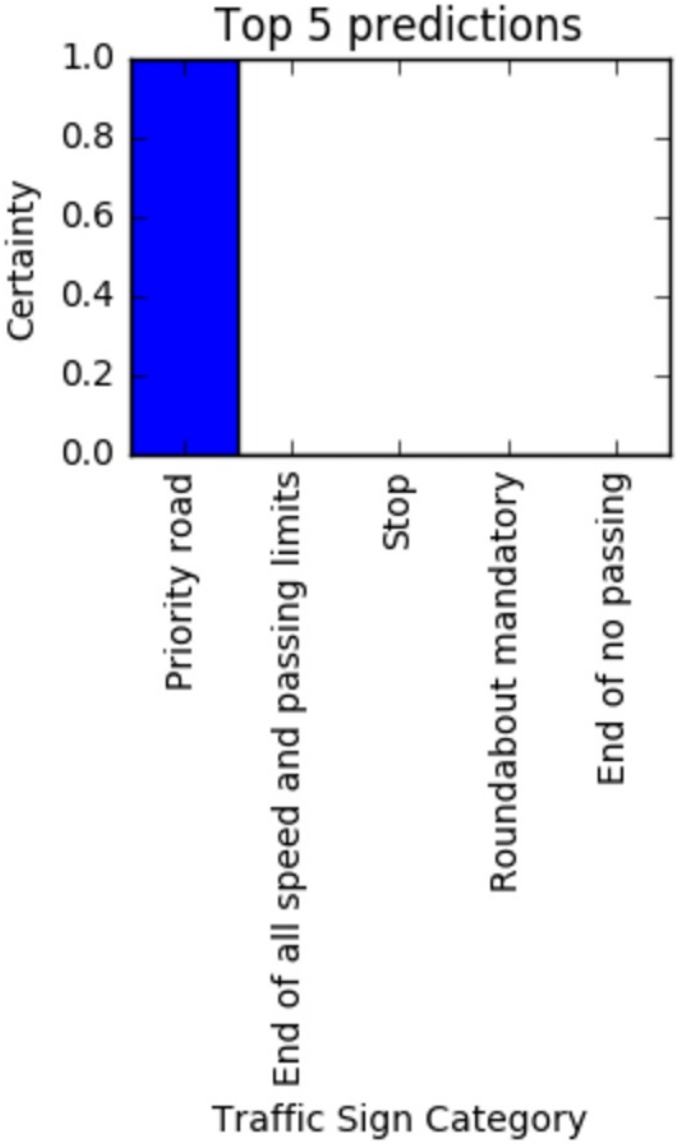

0
5
10
15
20
25
30
0 5 10 15 20 25 30
No passing
Top 5 predictions
1.0
0.8
0.6
0.4
0.2
0.0
Certainty
No passing
End of no passing
Speed limit (60km/h)
Dangerous curve to the left
Ahead only
Traffic Sign Category

Image 5


0
5
10
15
20
25
30
0 5 10 15 20 25 30
Priority road


Top 5 predictions
Certainty
1.0
0.8
0.6
0.4
0.2
0.0
Priority road
End of all speed and passing limits
Stop
Roundabout mandatory
End of no passing
Traffic Sign Category