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

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 here is a link to my project code

Data Set Summary & Exploration

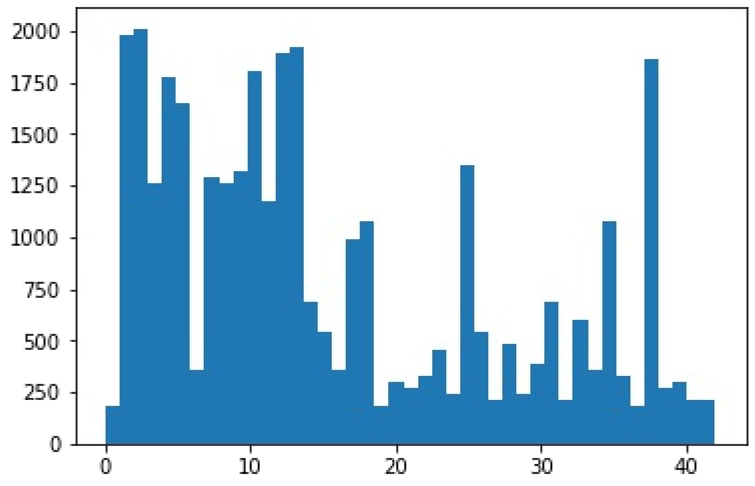
1. Provide a basic summary of the data set. In the code, the analysis should be done using python, numpy and/or pandas methods rather than hardcoding results manually.

I used the pandas library 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 32x32
- The number of unique classes/labels in the data set is 43

2. Include an exploratory visualization of the dataset.

Here is an exploratory visualization of the data set. It is a bar chart showing how many images are in each class. some classes have almost 200 images and some others have almost 2000. It's better that we have a balanced dataset.



Design and Test a Model Architecture

1. Describe how 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. (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, and provide example images of the additional data. Then describe the characteristics of the augmented training set like number of images in the set, number of images for each class, etc.)

I tested these preprocessing methods:

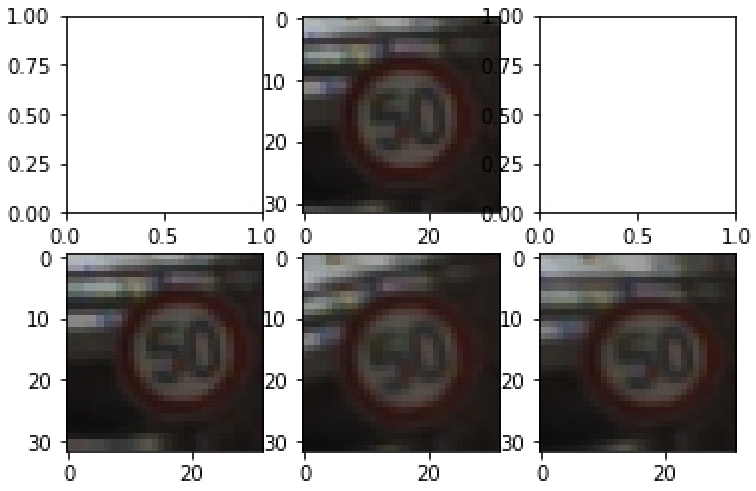
- Convert to greyscale: This method can't increase overall performace of designed network model and also it reduces a small amount of model parameters. So I decided not to use it.
- Converting to Other color spaces like YUV: I read in a paper which YUV color space will increase the performance of model but in this case it does not. So I did not use this.
- Normalization: First I normalize pixel values in [-1, 1] interval but with this interval the network model reaches almost 90% in validation set. Then I change the interval to [0, 1] and I got much better performance from the network model.

So the only preprocess I used for this problem is normalizing image pixels in [0, 1] interval.

I decided to generate additional data so the deep neural network I designed learn more information about each class. I used two type of augumentation:

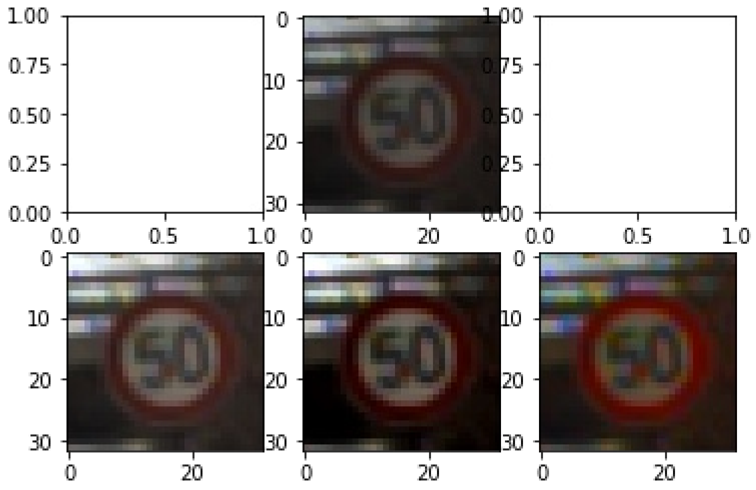
- Geometrical
  - Random translate
  - Random rotate from -15 to +15 degree
  - Random perspective

Here is an example of an original image and an augmented images: The first row contains original image and the second row from left to right consists of translated image, rotated image and the image modified by random perspective, respectively.



- Color
  - Random brightness
  - Random contrast
  - Random color saturation

The first row contains original image and the second row from left to right consists of images modified by random brightness, random constrast and random color saturation, respectively.

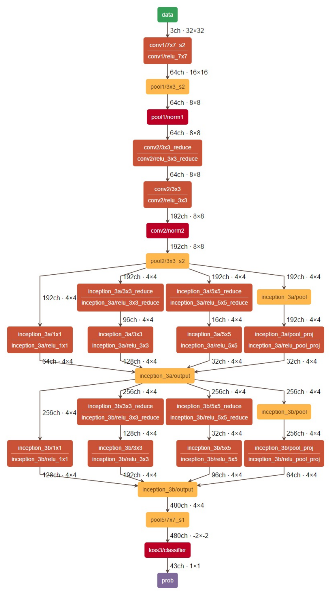


2. Describe what your final model architecture looks like including model type, layers, layer sizes, connectivity, etc.) Consider including a diagram and/or table describing the final model.

I tested these models:

1. LeNet
2. Multiscale LeNet
3. Modified SqueezeNet
4. Modified GoogLeNet

and I got much better performance from Modified GoogLeNet. GoogLeNet consists of many incpetion modules and I only use the first two inception modules as you can see in this figure.



3. Describe how 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.

I used AdamOptimizer with 0.001 as learning rate. I used 256 as batch size and 100 epochs in most trains. Some time I train the network for more epoch so I can make sure that there will be no more improvements.

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 ? 99.9
- validation set accuracy of ? 98.2
- test set accuracy of ? 97.0

If an iterative approach was chosen:

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

  First I've choosen Lenet betcause it's simple and works well in some cases, but as I've guessed it did not give a good accuracy (almost 93%).

  Then I changed the LeNet network in a way to convert it to a Multiscale network, So I concat output of first convnet with the second convnet. This network lacks a good accuracy (Under 93%)

```
                  +
                  |
              +----v-------+
              |           |
              |   CONV    |
              |           |
              +-----------+
          +---+   RELU    |
          |   +----+------+
          |        |
          |   +----v------+
          |   |           |
          |   |   CONV    |
          |   |           |
          |   +-----------+
          |   |   RELU    |
          |   +----+------+
          |        |
          |   +----v------+
          +--->  CONCAT   |
              +-----------+
                  |
                  v
```

I do another attemp to design a network for this problem with modified and minified SqueezeNet with no luck (Under 90%)

Finally I choose GoogLeNet as base model and remove all incpetion modules except the first two. With training of this model I achieve accuracies between 96%-98%.

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

  I think LeNet is a bit small for this problem and I think that the model should have more parameters so it can learn more features from traffic signs.

- **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 tested some imagenet architectures (SqueezeNet, GoogLeNet) but I think these architectures has many parameters and I need an architecture larger than LeNet but smaller than imagenet architectures, So I decided to modify them. SqueezeNet modification was not good but GoogLeNet modification gave me the good accuracy.

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

  As I used AdamOptimizer, I only have one paramter for optimizer and 0.001 was a good choice for me.

- **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?

I think inception micro-architecture is a good choice for this kind of pattern matching. So I chose GoogLeNet inceptions micro-architecture because they prove themselves in imagenet challenge.

If a well known architecture was chosen:

- **What architecture was chosen?**

  GoogLeNet

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

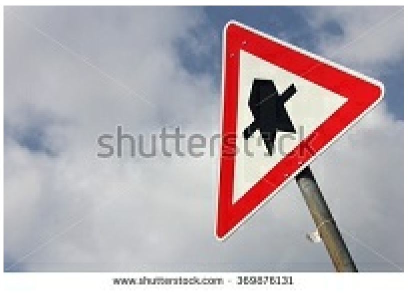
  I think inception micro-architecture is a good choice for this kind of pattern matching. So I chose GoogLeNet inceptions micro-architecture because they prove themselves in imagenet challenge.

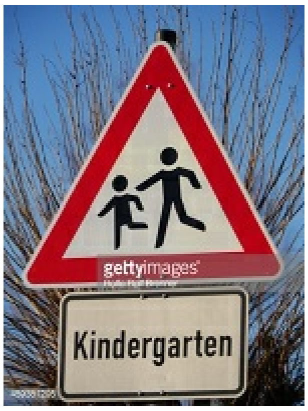
- How does the final model's accuracy on the training, validation and test set provide evidence that the model is working well? The modified version of GoogLeNet with only two inception modules gave me 99.9% on training set, 98.2% on validation set and 97.0% on test set. As we can see it gives us 97% accuracy on test set which I think it's the proof.

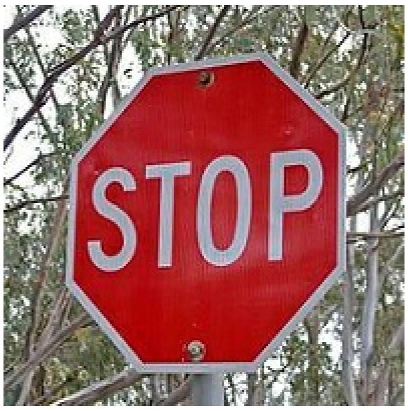
Test a Model on New Images

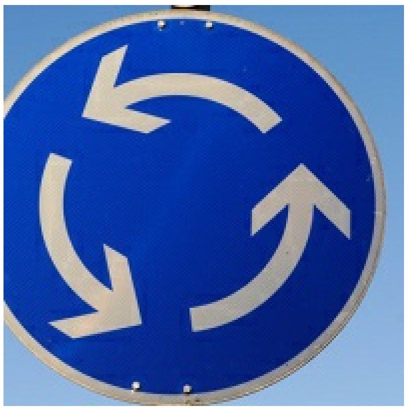
1. Choose five German traffic signs found on the web and provide them in the report. For each image, discuss what quality or qualities might be difficult to classify.

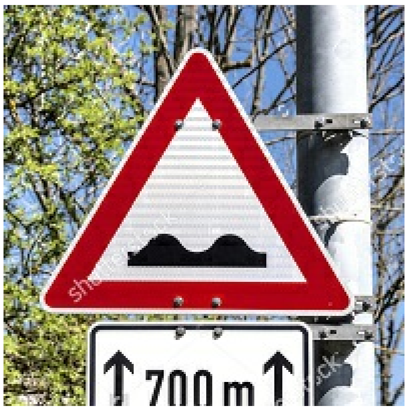
Here are five German traffic signs that I found on the web:











The first image might be difficult to classify because it pictured from buttom, the second the last one contains another textual sign bellow the main sign, the third sign has no difficulty to classify and the fourth sign is croppted.

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
Right-of-way at the next intersection	Priority road
Children crossing	Children crossing
Stop	Stop
Roundabout mandatory	Roundabout mandatory
Bumpy road	Bumpy Road

The model was able to correctly guess 4 of the 5 traffic signs, which gives an accuracy of 80%.

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 11th cell of the Ipython notebook.

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

Probability	Prediction
1.0	Priority road
.00	End of no passing
.00	Turn left ahead
.00	Right-of-way at the next intersection
.00	Go straight or left

For the second image ...

Probability	Prediction
1.0	Children crossing

.00	Speed limit (20km/h)
.00	Speed limit (30km/h)
.00	Speed limit (50km/h)
.00	Speed limit (60km/h)

For the third image ...

Probability	Prediction
1.0	Stop
.00	Speed limit (20km/h)
.00	Speed limit (30km/h)
.00	Speed limit (50km/h)
.00	Speed limit (60km/h)

For the fourth image ...

Probability	Prediction
1.0	Roundabout mandatory
.00	Speed limit (20km/h)
.00	Speed limit (30km/h)
.00	Speed limit (50km/h)
.00	Speed limit (60km/h)

For the fifth image ...

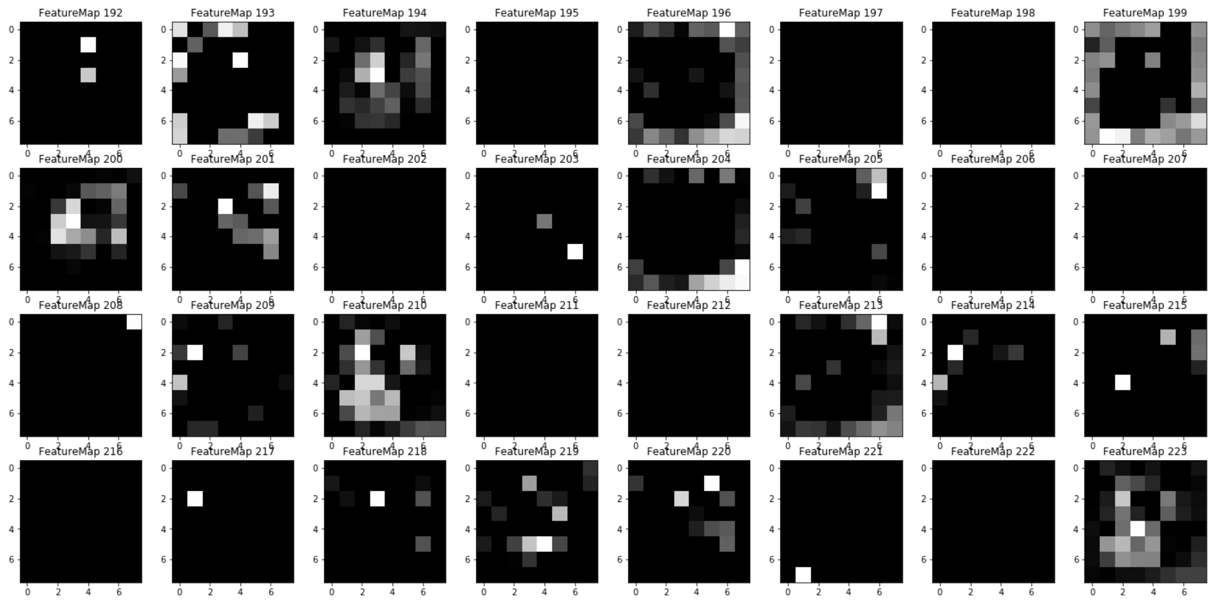
Probability	Prediction
1.0	Bumpy road
.00	Speed limit (20km/h)
.00	Speed limit (30km/h)
.00	Speed limit (50km/h)
.00	Speed limit (60km/h)

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

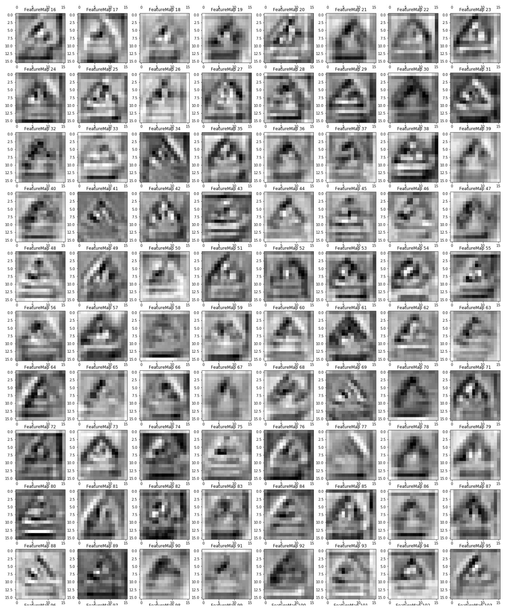
1. Discuss the visual output of your trained network's feature maps. What characteristics did the neural network use to make classifications?

For visualizing the model I used two methods, first method is the one that includes in IPython notebook and second one is guided backpropagation (ref link)

First I visualize output of the last layer in second inception but it give me nothing.



so I decided to use the last convultion before inception modules. The visualization of that convultion shows that it is sensitive both in shape of the sign and the textual part of sign as well.



Visualizing the network with guided backpropagation shows me the pixels on image where network are used to decide the class. this visualization is more helpfull when an image classified incorrectly and with this method we can see that which part of the image is used to achieve the incorrect classification.