Writeup for Traffic Sign Recognition

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](https://review.udacity.com/#!/rubrics/481/view) individually and describe how I addressed each point in my implementation.

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 narray's shape attribute to calculate size of train, validation and test sets. And I used numpy's unique function to find out how many classes in the 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.

Here is an exploratory visualization of the data set. It is a bar chart showing how the data distribute. Image1 is how the training set distribute. Image2 is how the validation set distribute. Image3 is how the testing set distribute. Data in these three dataset seem to distribute similarly.

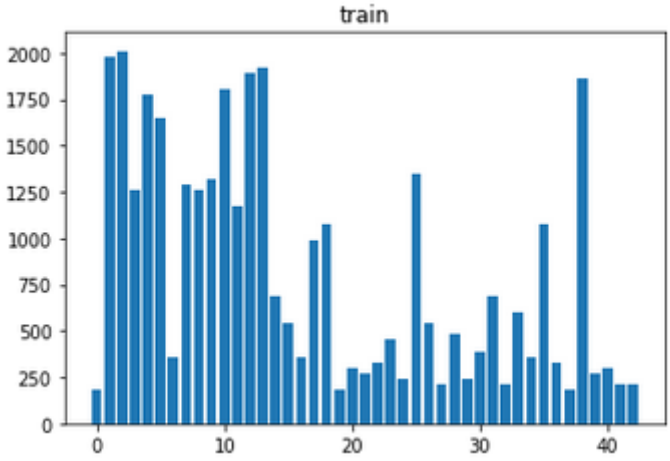

Image1

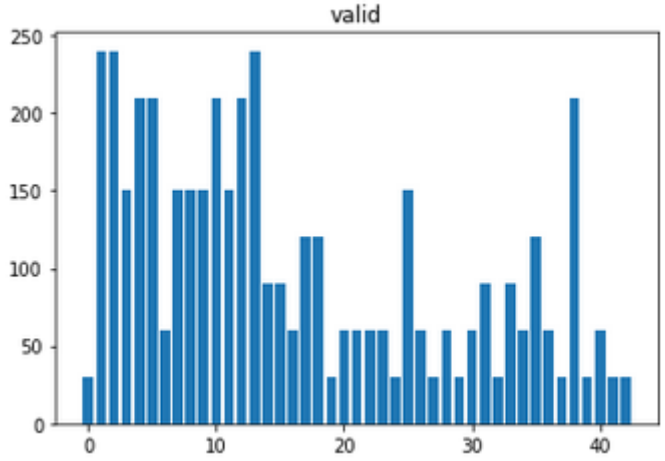

Image2

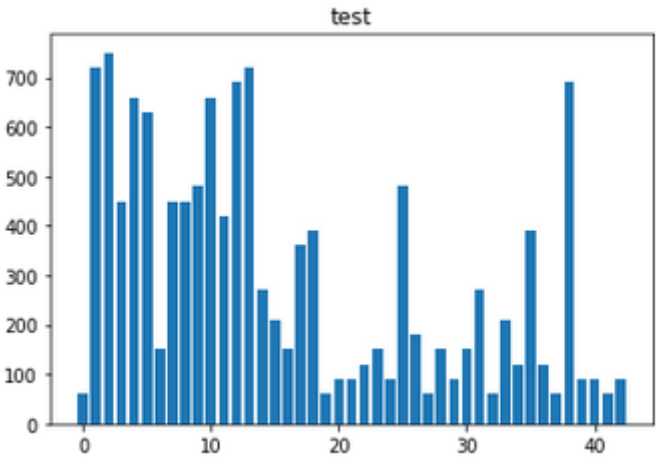

Image3

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.)

As a first step, I decided to convert the images to grayscale because grayscale means we want the gradient in the picture, because big gradient means edges which represents information. And color is likely to change between different sunlight and other condition like shadow, so it varies too much.
Here is an example of a traffic sign image before and after grayscaling.

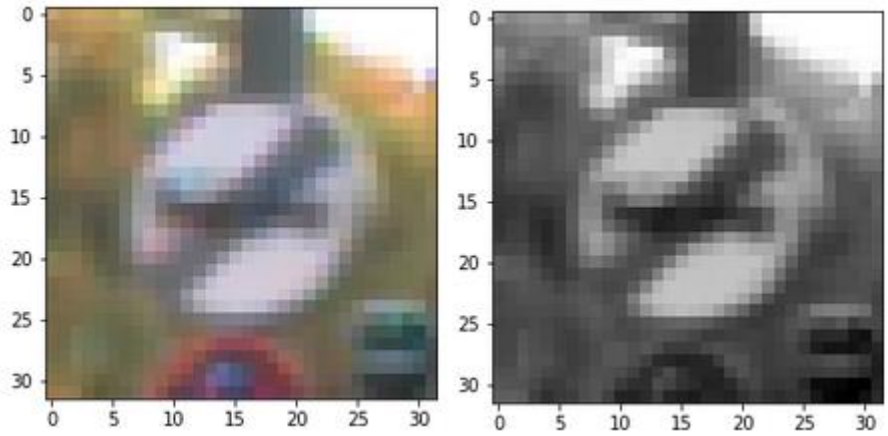

Image4

As a last step, I normalized the image data because we want the error function to never be too big or too little, besides normalized data is also easy for optimization.

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.

My final model consisted of the following layers:

Layer	Description
Input	32x32x1 Gray image
Convolution 5x5	1x1 stride, valid padding, outputs 28x28x32
RELU	
Max pooling	2x2 stride, valid padding, outputs 14x14x32
Convolution 3x3	1x1 stride, valid padding, outputs 12x12x64
RELU	
Max pooling	2x2 stride, valid padding, outputs 6x6x64
Flatten	outputs 2304

Fully connected	Inputs 2304, outputs 800
RELU	
Dropout	Rate 0.6
Fully connected	Inputs 800, outputs 320
RELU	
Dropout	Rate 0.6
Fully connected	Inputs 320, outputs 430

Table1

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.

To train the model, I used this LeNet architecture, the network is shown in table1, the first layer is convolutional layer, I tried a 3*3 filter, but the accuracy is lower than 5*5 filter. A ReLu Layer and a maxpooling layer is followed, maxpooling helps model to reduce width and height dimension. Therefore it may help to reduce overfitting. dropout layers in fully connected layer also help to reduce overfitting.

I have tried 0.5, 0.6, 0.7, 0.8, lower learning rate result in higher accuracy. The difference between 0.5 and 0.6 is little, as shown in image below. Image5 is when learning rate is 0.5, image6 is when learning rate is 0.6. I just pick 0.6 for my model.

```
Dropout p.5
Training...
EPOCH 1 ...
Train Accuracy = 0.572
Validation Accuracy = 0.482
EPOCH 2 ...
Train Accuracy = 0.815
Validation Accuracy = 0.729
EPOCH 3 ...
Train Accuracy = 0.898
Validation Accuracy = 0.813
EPOCH 4 ...
Train Accuracy = 0.936
Validation Accuracy = 0.846
EPOCH 5 ...
Train Accuracy = 0.954
Validation Accuracy = 0.866
EPOCH 6 ...
Train Accuracy = 0.965
Validation Accuracy = 0.882
EPOCH 7 ...
Train Accuracy = 0.972
Validation Accuracy = 0.896
EPOCH 8 ...
Train Accuracy = 0.979
Validation Accuracy = 0.899
EPOCH 9 ...
Train Accuracy = 0.981
Validation Accuracy = 0.907
EPOCH 10 ...
Train Accuracy = 0.985
Validation Accuracy = 0.916
EPOCH 11 ...
Train Accuracy = 0.985
Validation Accuracy = 0.915
EPOCH 12 ...
Train Accuracy = 0.988
Validation Accuracy = 0.923
EPOCH 13 ...
Train Accuracy = 0.990
Validation Accuracy = 0.927
EPOCH 14 ...
Train Accuracy = 0.990
Validation Accuracy = 0.920
EPOCH 15 ...
Train Accuracy = 0.994
Validation Accuracy = 0.929
EPOCH 16 ...
Train Accuracy = 0.993
Validation Accuracy = 0.924
EPOCH 17 ...
Train Accuracy = 0.994
Validation Accuracy = 0.929
EPOCH 18 ...
Train Accuracy = 0.995
Validation Accuracy = 0.937
EPOCH 19 ...
Train Accuracy = 0.996
Validation Accuracy = 0.938
EPOCH 20 ...
Train Accuracy = 0.995
Validation Accuracy = 0.948
EPOCH 21 ...
Train Accuracy = 0.995
Validation Accuracy = 0.939
EPOCH 22 ...
Train Accuracy = 0.996
Validation Accuracy = 0.934
EPOCH 23 ...
Train Accuracy = 0.992
Validation Accuracy = 0.926
EPOCH 24 ...
Train Accuracy = 0.995
Validation Accuracy = 0.928
EPOCH 25 ...
Train Accuracy = 0.997
Validation Accuracy = 0.947

Model saved
```

Image5

Dropout 0.6
Training...
EPOCH 1 ...
Train Accuracy = 0.676
Validation Accuracy = 0.577
EPOCH 2 ...
Train Accuracy = 0.875
Validation Accuracy = 0.776
EPOCH 3 ...
Train Accuracy = 0.927
Validation Accuracy = 0.837
EPOCH 4 ...
Train Accuracy = 0.952
Validation Accuracy = 0.859
EPOCH 5 ...
Train Accuracy = 0.967
Validation Accuracy = 0.886
EPOCH 6 ...
Train Accuracy = 0.976
Validation Accuracy = 0.889
EPOCH 7 ...
Train Accuracy = 0.984
Validation Accuracy = 0.900
EPOCH 8 ...
Train Accuracy = 0.984
Validation Accuracy = 0.904
EPOCH 9 ...
Train Accuracy = 0.988
Validation Accuracy = 0.912
EPOCH 10 ...
Train Accuracy = 0.990
Validation Accuracy = 0.921
EPOCH 11 ...
Train Accuracy = 0.992
Validation Accuracy = 0.920
EPOCH 12 ...
Train Accuracy = 0.992
Validation Accuracy = 0.921
EPOCH 13 ...
Train Accuracy = 0.994
Validation Accuracy = 0.928
EPOCH 14 ...
Train Accuracy = 0.994
Validation Accuracy = 0.924
EPOCH 15 ...
Train Accuracy = 0.995
Validation Accuracy = 0.920
EPOCH 16 ...
Train Accuracy = 0.995
Validation Accuracy = 0.934
EPOCH 17 ...
Train Accuracy = 0.993
Validation Accuracy = 0.921
EPOCH 18 ...
Train Accuracy = 0.997
Validation Accuracy = 0.943
EPOCH 19 ...
Train Accuracy = 0.996
Validation Accuracy = 0.929
EPOCH 20 ...
Train Accuracy = 0.996
Validation Accuracy = 0.939
EPOCH 21 ...
Train Accuracy = 0.997
Validation Accuracy = 0.944
EPOCH 22 ...
Train Accuracy = 0.996
Validation Accuracy = 0.931
EPOCH 23 ...
Train Accuracy = 0.997
Validation Accuracy = 0.941
EPOCH 24 ...
Train Accuracy = 0.998
Validation Accuracy = 0.942
EPOCH 25 ...
Train Accuracy = 0.998
Validation Accuracy = 0.945
Model saved

Image6

I chose 25 as epochs is because at 25 epoch, the validation accuracy is still changing so it is not too much, and the number starts to vibrate, not keeping increasing(as shown in image5 and image6), so it is not too little.

I have tried three optimizer, momentum optimizer seems to work really slow, and the output is worse than adam, as shown in image7, it gets around 0.92accuracy with learning rate of 0.1, 80 epochs, 0.9 momentum. It may achieve 0.93 validation accuracy with more tuning, but it just works so slow. GradientDescentOptimizer works worse than momentum optimizer, as shown in image8, after 80 epochs of training, it only achieve 0.72 validation accuracy, so I didn't choose this one.

Momentumoptimizer
Rate0.001 epoch80
.....
EPOCH 60 ...
Train Accuracy = 0.967
Validation Accuracy = 0.901
EPOCH 61 ...
Train Accuracy = 0.970
Validation Accuracy = 0.907
EPOCH 62 ...
Train Accuracy = 0.970
Validation Accuracy = 0.907
EPOCH 63 ...
Train Accuracy = 0.971
Validation Accuracy = 0.905
EPOCH 64 ...
Train Accuracy = 0.974
Validation Accuracy = 0.912
EPOCH 65 ...
Train Accuracy = 0.973
Validation Accuracy = 0.905
EPOCH 66 ...
Train Accuracy = 0.973
Validation Accuracy = 0.906
EPOCH 67 ...
Train Accuracy = 0.974
Validation Accuracy = 0.906
EPOCH 68 ...
Train Accuracy = 0.975
Validation Accuracy = 0.908
EPOCH 69 ...
Train Accuracy = 0.975
Validation Accuracy = 0.909
EPOCH 70 ...
Train Accuracy = 0.976
Validation Accuracy = 0.915
EPOCH 71 ...
Train Accuracy = 0.976
Validation Accuracy = 0.912
EPOCH 72 ...
Train Accuracy = 0.976
Validation Accuracy = 0.914
EPOCH 73 ...
Train Accuracy = 0.976
Validation Accuracy = 0.913
EPOCH 74 ...
Train Accuracy = 0.978
Validation Accuracy = 0.914
EPOCH 75 ...
Train Accuracy = 0.978
Validation Accuracy = 0.918
EPOCH 76 ...
Train Accuracy = 0.978
Validation Accuracy = 0.914
EPOCH 77 ...
Train Accuracy = 0.977
Validation Accuracy = 0.911
EPOCH 78 ...
Train Accuracy = 0.978
Validation Accuracy = 0.909
EPOCH 79 ...
Train Accuracy = 0.979
Validation Accuracy = 0.910
EPOCH 80 ...
Train Accuracy = 0.980
Validation Accuracy = 0.915
Model saved

Image7

GradientDescentOptimizer
Rate 0.001, epoch 80
.....
EPOCH 57 ...
Train Accuracy = 0.604
Validation Accuracy = 0.525
EPOCH 58 ...
Train Accuracy = 0.617
Validation Accuracy = 0.544
EPOCH 59 ...
Train Accuracy = 0.619
Validation Accuracy = 0.548
EPOCH 60 ...
Train Accuracy = 0.622
Validation Accuracy = 0.549
EPOCH 61 ...
Train Accuracy = 0.631
Validation Accuracy = 0.556
EPOCH 62 ...
Train Accuracy = 0.637
Validation Accuracy = 0.559
EPOCH 63 ...
Train Accuracy = 0.643
Validation Accuracy = 0.568
EPOCH 64 ...
Train Accuracy = 0.646
Validation Accuracy = 0.560
EPOCH 65 ...
Train Accuracy = 0.650
Validation Accuracy = 0.578
EPOCH 66 ...
Train Accuracy = 0.659
Validation Accuracy = 0.576
EPOCH 67 ...
Train Accuracy = 0.663
Validation Accuracy = 0.585
EPOCH 68 ...
Train Accuracy = 0.662
Validation Accuracy = 0.579
EPOCH 69 ...
Train Accuracy = 0.670
Validation Accuracy = 0.599
EPOCH 70 ...
Train Accuracy = 0.678
Validation Accuracy = 0.592
EPOCH 71 ...
Train Accuracy = 0.677
Validation Accuracy = 0.586
EPOCH 72 ...
Train Accuracy = 0.684
Validation Accuracy = 0.609
EPOCH 73 ...
Train Accuracy = 0.689
Validation Accuracy = 0.612
EPOCH 74 ...
Train Accuracy = 0.698
Validation Accuracy = 0.609
EPOCH 75 ...
Train Accuracy = 0.698
Validation Accuracy = 0.622
EPzCH 76 ...
Train Accuracy = 0.697
Validation Accuracy = 0.614
EPOCH 77 ...
Train Accuracy = 0.707
Validation Accuracy = 0.629
EPOCH 78 ...
Train Accuracy = 0.711
Validation Accuracy = 0.625
EPOCH 79 ...
Train Accuracy = 0.716
Validation Accuracy = 0.635
EPOCH 80 ...
Train Accuracy = 0.721
Validation Accuracy = 0.629
Model saved

Image8

Adamoptimizer works really fast, and with some tuning it can achieve 0.93 accuracy on validation set. So I choose Adamoptimizer.
At first I set learning rate at 0.001, then I discover lower learning rate can increase accuracy, so I set learning rate at 0.0004 after some trial. 0.0004 seems to be okay.

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 ? 0.997
* validation set accuracy of ? 0.940
* test set accuracy of ? 0.930

If an iterative approach was chosen:
* What was the first architecture that was tried and why was it chosen?
* What were some problems with the initial architecture?
* 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.
* Which parameters were tuned? How were they adjusted and why?
* 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?

If a well known architecture was chosen:
* What architecture was chosen?
* Why did you believe it would be relevant to the traffic sign application?
* How does the final model's accuracy on the training, validation and test set provide evidence that the model is working well?

LeNet was chosen, because this architecture works well on small images like recognizing handwriting image as we discussed in class. Training accuracy is larger than validation accuracy, which means the model has some overfitting. But validation accuracy is greater than 0.93, test accuracy is 0.926, so I guess it works okay.

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:

Image9

Image10

Image11

Image12

Image13

The fifth image13 might be difficult to classify because all images from data seem to be taken right in front of the traffic sign, the fifth image was taken in a strange angle from below, which means this image has a different perspective, and it may be hard to be classified. Other images seems to be classified fine.

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
General caution	General caution
No passing	No passing
Road work	Road work
Turn right ahead	Turn right ahead
Road narrows on the right	Keep left

Accuracy is 80%, which is lower than test accuracy. I noticed in German Traffic Signs Dataset, the images are taken right in front of them and without background. At first the image9 has some background as shown in image14, and it got a wrong prediction,

so is image4 and image 5. So at first I only got a accuracy of 0.4. Then I cut out only the traffic sign part of the image, and I got a 0.8 accuracy. So wrongful prediction may because my images are not quite identical with German Traffic Signs Dataset.

Image14

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)

For the first image, the model is very sure that this is a General caution sign (probability nearly 1.0), and the image does contain a General caution sign. The top three soft max probabilities were

Probability	Prediction
1.00000000e+00	General caution
2.18791389e-13	Pedestrians
5.97816203e-14	Traffic signals
2.93775158e-22	Right-of-way at the next intersection
4.22379763e-23	Double curve

For the second image, the model is very sure that this is a No passing sign (probability nearly 1.0), and the image does contain a No passing sign. The top three soft max probabilities were

Probability	Prediction
1.00000000e+00	No passing
1.16238067e-16	End of no passing
6.74316343e-19	Go straight or left
6.40872397e-19	Vehicles over 3.5 metric tons prohibited
3.71980738e-21	Dangerous curve to the left

For the third image, the model is very sure that this is a Road work sign (probability

nearly 0.99), and the image does contain a Road work sign. The top three soft max probabilities were

Probability	Prediction
9.94571984e-01	Road work
5.15016960e-03	Keep left
1.53788948e-04	Wild animals crossing
6.67598506e-05	Turn right ahead
4.19846583e-05	Pedestrians

For the fourth image, the model is very sure that this is a Turn right ahead sign (probability nearly 1.0), and the image does contain a Turn right ahead. The top three soft max probabilities were

Probability	Prediction
1.00000000e+00	Turn right ahead
3.62993663e-11	Ahead only
1.93461067e-12	Speed limit (70km/h)
5.29258386e-13	Road narrows on the right
5.14687956e-13	Go straight or left

For the fifth image, the model is pretty sure that this is a Turn right ahead sign (probability is 1.00), and the image is not a Go straight or left sign. The top three soft max probabilities were

Probability	Prediction
9.99965310e-01	Keep left
2.98445775e-05	Go straight or left
4.11156770e-06	Speed limit (50km/h)
5.00264150e-07	Stop
1.01519745e-07	Roundabout mandatory

At last I have some question I would like to discuss. The first one is that I noticed images in German Traffic Signs Dataset do not distribute evenly, is it because in real life traffic signs don't distribute evenly? I mean some traffic signs are more common to see on the street. So if we generalize more images to make the distribution more even, does it conflict what is real in reality? If some traffic signs are more than others, the model should be more 'biased' to them, is it right?

The second one is how to obtain this tiny images in real life? How do we tell traffic signs from background? And even if we know there is a traffic sign due to the map, how do we know where to zoom in to get these 32*32 tiny little images?

The third one is in the last project, we used some cv2 function to process images to detect lane lines. I tried some tricks in this project, for example, I tried canny algorithm to detect edges, and it does not work well. My guess is the image is too small, and the cv2 transformation may lose information in the process. So for tiny and vague images like I encountered in this project, is there any way to process them other than grayscale?

The last one is I noticed the images in the dataset are taken right in front of the traffic sign, in real life, the images taken from cameras mounted on a car has a different perspective, so do we need to change perspective before we feed the

images to the model? If so I suppose it won’t be easy to find source points for cv2.getPerspectiveTransform function. So how do we solve this?