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

You're reading it! and here is a link to my project

Data Set Summary & Exploration

1. Provide a basic summary of the data set.

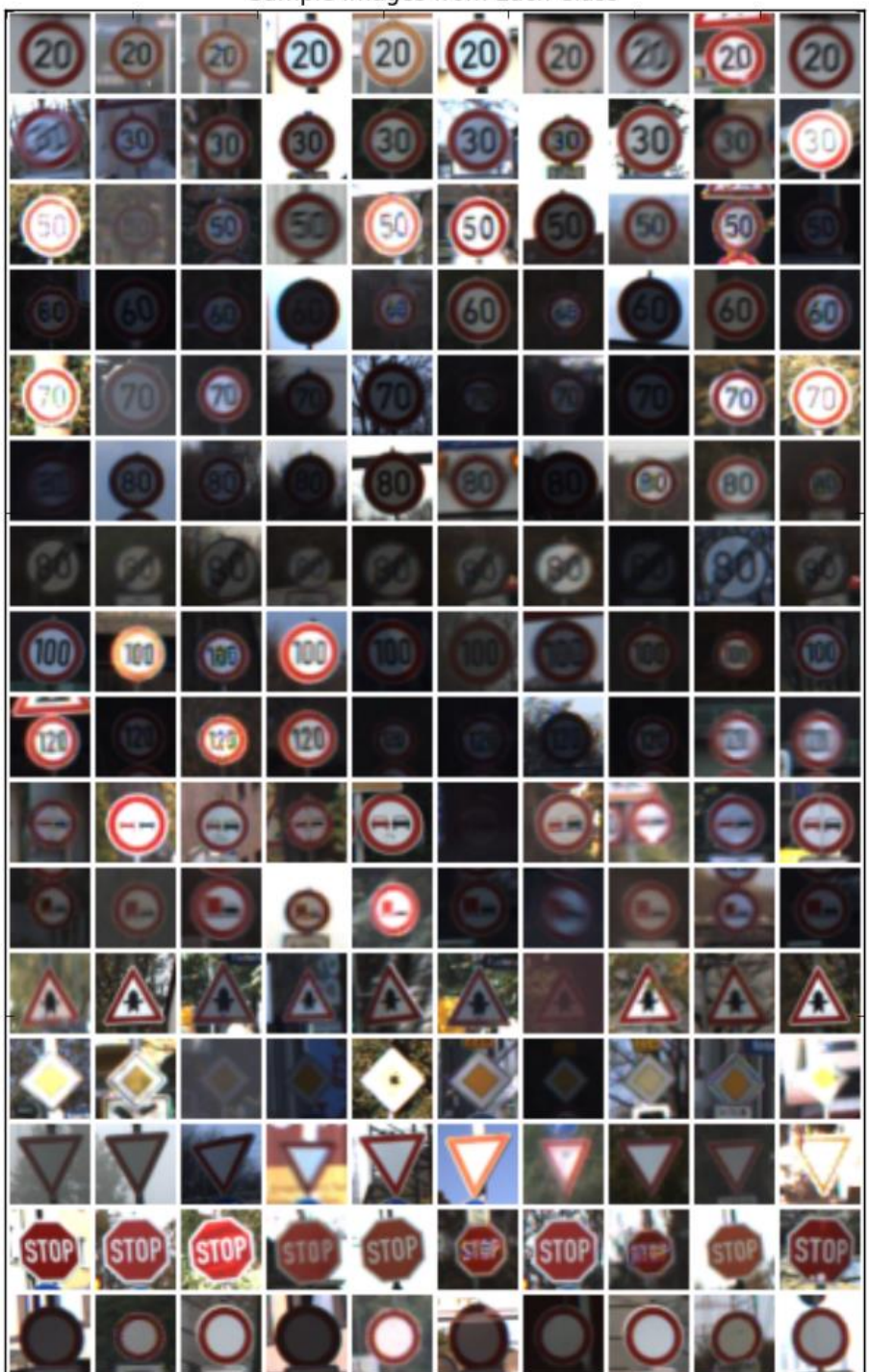
The German Traffic Signs data set is a labeled data set of German traffic sign images. Each image is 32x32x3 pixels (32x32 RGB color) and has an integer class identifier ranging from 0 to 42 (43 classes). There is a .csv file naming each integer class. The data set is provided in three files corresponding to predefined training, validation and test splits with the following sizes:

- Number of training examples = 34799
- Number of validation examples = 4410
- Number of testing examples = 12630
- The shape of a traffic sign image is 32x32
- The number of unique classes / labels in the data set is 43

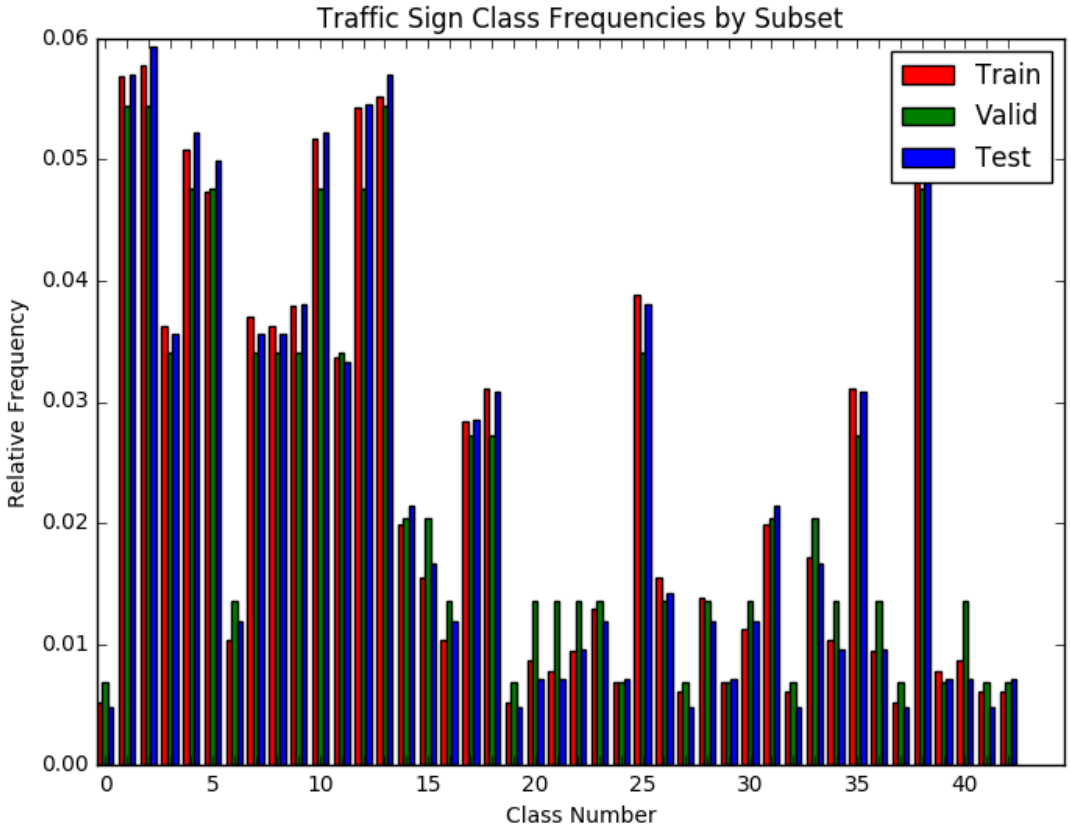
2. Include an exploratory visualization of the dataset.

To visualize the data I constructed a plot in which each of 43 rows contains 10 images of signs from a particular class. This visualization provided a quick sanity check to confirm that the data was loading as expected, and it revealed the relatively poor quality of the images. Many of the images are blurry and poorly lighted so that classifying them would be error prone even for a human.

Sample Images from Each Class



I also constructed a bar chart comparing the relative rates of occurrences for image class instances in the training, validation, and test sets. The chart reveals that the distribution of individual image instances varies significantly by class but is relatively consistent between the training, validation and test splits. The consistency between splits is desirable as it indicates the training and test data have somewhat similar characteristics.



Design and Test a Model Architecture

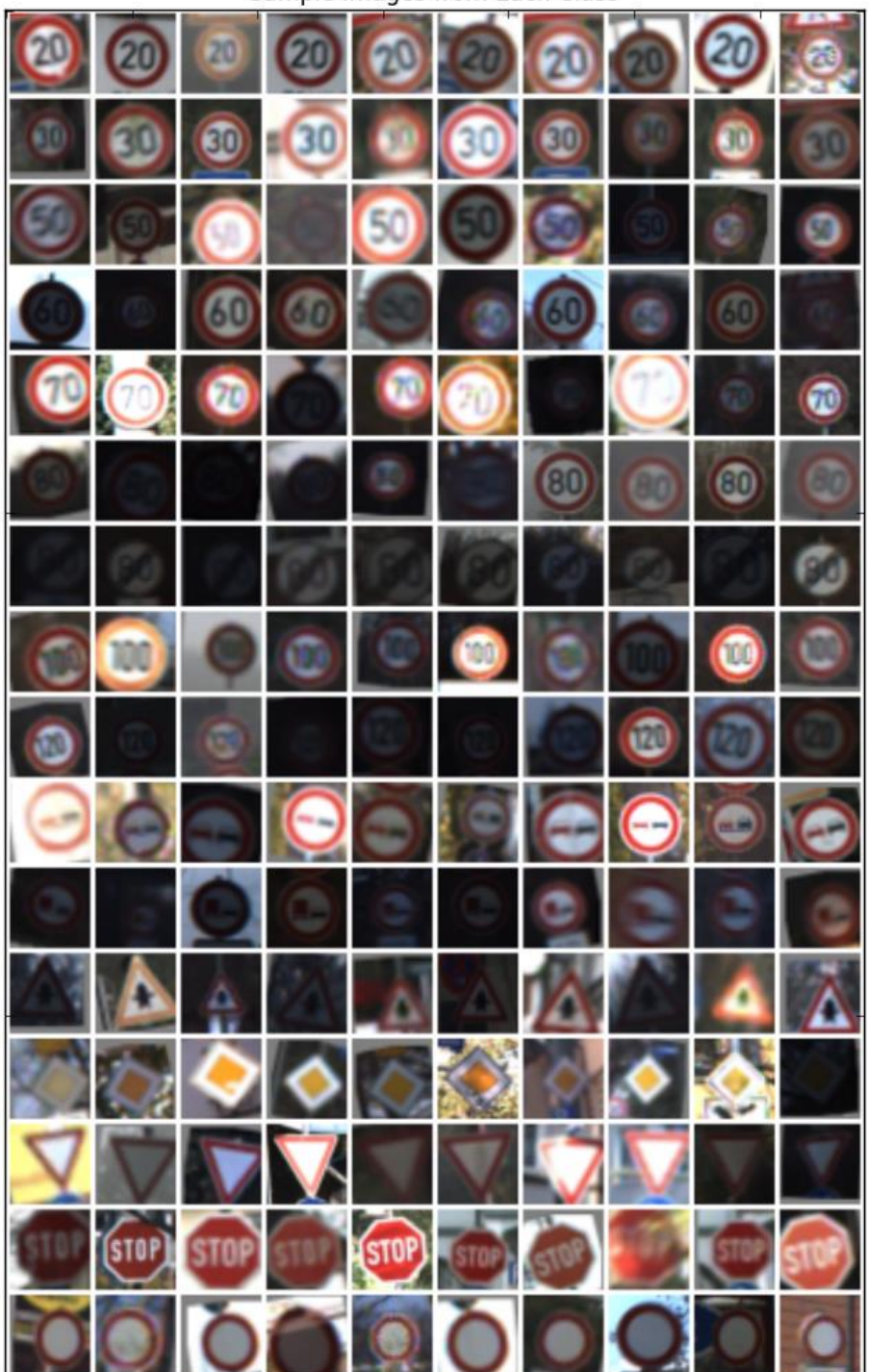
1. Describe how you preprocessed the image data.

I standardized the data using the simple (x-128)/128 strategy suggested in the project description. Normalizing the data to zero mean and unit variance is standard practice for machine learning systems and generally improves their performance by holding the dynamic range of values in a range for which the algorithms are optimized, avoiding ill-conditioning, and other mechanisms. The strategy described above achieves approximate normalization.

I also experimented with data augmentation of the training image split with the intent of improving generalization performance of the model. For each training image, I augmented the data set with three additional images having random rotation over the range -15 to 15 degrees, random translation over the range -2 to 2 pixels, and random scaling by factors between 1.0 and 1.3. The rotation and translation limits were based on the augmentation approach described in the referenced paper by

Yann LeCun. I found that the model generally trained more slowly with the augmented data but that the gap between train set performance and validation set performance was much reduced when training with the augmented data. Overall, my best test scores of ~0.97 were acheived while training with augmented data, and validation scores of ~0.975+ we're routinely observed. The figure below illustrates the augmented training data.

Sample Images from Each Class



2. Describe what your final model architecture looks like including model type, layers, layer sizes, connectivity, etc.

I started with the LeNet convolutional architecture and modified it in the following ways:

- Increased most layer dimensions to increase model capacity
- Changed relu to elu activation functions to guard against dead units
- Added dropout layers to improve regularization and reduce overfitting given greater model capacity
- Changed to Xavier weight initialization to provide improved bidirectional information flow during training
- Added fully connected layers to increase depth and model capacity

My final model consisted of the following layers:

Layer	Description
Convolution 2d	input=[32,32,3], output=[28,28,24], kernel = [5,5,3,24], strides=[1,1,1,1], activation=elu
Maxpool	input=[28,28,24], output=[14,14,24], kernel=[1,2,2,1], strides=[1,2,2,1]
Convolution 2d	input=[14,14,24], output=[10,10,64], kernel = [5,5,24,64], strides=[1,1,1,1], activation=elu
Maxpool	input=[10,10,64], output=[5,5,64], kernel=[1,2,2,1], strides=[1,2,2,1]
Flatten	input=[5,5,64],output=[1600]
Dropout	keep = 0.5 (training only)
Fully Connected	input=[1600], output=[800]
Dropout	keep = 0.5 (training only)
Fully Connected	input=[800], output=[800]
Dropout	keep = 0.5 (training only)
Fully Connected	input=[800], output=[400]
Dropout	keep = 0.5 (training only)

Layer	Description
Fully Connected	input=[400], output=[400]
Dropout	keep = 0.5 (training only)
Fully Connected	input=[400], output=[43]

This model has significantly more parameters and computational complexity than the LeNet model from which I derived it. I used the strategy of making the model capable of overlearning the data set and then using dropout and feature augmentation as regularization strategies to combat overfitting and improve generalization. Based on articles I've read and personal experience, this approach often yields good test performance at the expense of computational and parametric efficiency. My final model does perform reasonably well, but it is not well optimized for efficiency as might be required for a real-time embedded vehicular application.

I experimented with the number and placement of the dropout layers and found the best performance was somewhat related to the model capacity with large model capacity requiring additional dropout layers. Dropout applied in the convolutional layers did not produce good results, and this is likely because the convolutional layers are in general less prone to overfitting owing to their parametric sparsity. When applying dropout to fully connected layers, it seemed that applying dropout to the initial fully connected layers was most effective and that moving toward the output dropout was progressively less crucial.

Model Training

I used the Adam optimizer for training the model with a learning rate parameter of 0.0005. Training at learning rates above 0.0005 was sometimes unstable and would cause the model to diverge. I trained the model for 20 epochs, where each epoch included the original training data and three augmented variants for a total of 4 times the original training data per epoch. I used a batch size of 128 samples for the first 10 epochs and 2048 samples thereafter. I used this strategy because the small batch size allowed for very rapid initial convergence, but I found that the small batch size seemed to make validation performance volatile at the later stages of training. This may be due to the somewhat aggressive nature of the Adam optimizer with respect to learning rate. By increasing the batch size during the later training epochs I found the validation performance to be more stable and predictable.

Solution Approach

I started with the LeNet architecture which is a well-known architecture for image classification. Its performance is somewhat short of the required performance for this project, so I decided to experiment with various changes to see if I could improve it. I began by increasing the dimensions for many of the internal structures to give the model more capacity. This improved performance to a degree, but the model started to exhibit signs of overfitting, so I added dropout layers to improve regularization. I changed the relu units to elu units because I've read that they provide some resistance to the "dead neuron" problem since their gradient is always nonzero. I changed the weight initialization standard deviation and found that this had a large effect on model performance, so I

changed to Xavier initialization of the weights because I've read that it is effective in establishing a network through which both forward and backward propagation of information can occur leading to effective training of the network. The Xavier initialization provided layer-specific standard deviation values that worked as well as or better than my hand selected standard deviations, so I retained this modification as well. Based on the recent success of much deeper neural networks, I added layers and found that additional convolutional layers were not helpful but additional fully connected layers improved performance somewhat. I did not see a significant benefit to additional layers beyond the fully connected layers I added. I experimented with the learning rate parameter and found that a value of 0.0005 led to reliable convergence and good performance, while values above 0.0007 sometimes led to training failure. I observed that small batch sizes were much better for initial convergence but led to volatile model behavior in later training stages, so I changed to smaller batch sizes after the first few epochs and found that this made performance of the trained network less variable. One of the last changes I made was to augment the training data with random rotations, translations, and scaling in order to expose the network to a wider variety of valid training data. The extra data slowed convergence of the network, but it resulted in a somewhat better validation set performance. Networks trained with augmented data also had test set performance that was very close to their validation set performance and performed best overall. I kept the Adam optimizer throughout as it is generally regarded to yield consistently good performance.

I experimented with more aggressive data augmentation including large amounts of translation and scaling, and this improved performance on classifying images from the web (which had scaling and translation outside the typical examples from the dataset). Unfortunately this improved generalization cam at the cost of decreased accuracy on the actual dataset, so I used more conservative values in my final network.

My final model results were:

- Training set accuracy of 0.998
- **Validation set accuracy of 0.977**
- Test set accuracy of 0.967

The high training accuracy might suggest that the model is over fit, but additional interventions to reduce overfitting (reduction of model capacity, early stopping, and an additional dropout layer) did not materially improve validation or test accuracy. The validation accuracy of 0.977 is reasonably good compared to published results, so I think it might take a fairly significant architecture change to significantly improve the results. The test set accuracy of 0.967 is somewhat short of the 0.99+ results published by Yann LeCun and others (Traffic Sign Recognition with Multi-Scale Convolutional Networks), so there is certainly at least some room for improvement.

Next steps might be to implement batch normalization between layers and to incorporate deeper additional layers with skip layer connections as these have been demonstrated to yield excellent performance in some recent image classification competitions. It might also be beneficial to include additional forms of data augmentation and to evaluate color space transformations (for example YUV) as implemented in Yann LeCun's referenced paper. In the convolutional area, 1x1 convolutions might be worth trying.

Test a Model on New Images

Signs from the Web

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

The signs might be difficult to classify because:

- The “speed limit 20” sign is somewhat off-center.
- The “stop” sign is far off-center and heavily cropped.
- The “no entry” sign doesn't have any obvious difficulties.
- The “children crossing” sign is somewhat off-center, rotated and not a full frontal view. It is also very similar to other warning signs.
- The “yield” sign is far off-center.
- The “roundabout mandatory” sign is large, off-center, and partially cropped.
- The “beware of ice” sign might be a challenge because the snowflake in the center is the only differentiator between it and other signs like children crossing.
- The “no passing” sign is very large and severely cropped and has an extra heart painted on it.

Performance on New Images

The model classified 5 of the 8 signs correctly for a (lackluster) accuracy of 62.5%. It misclassified the speed limit 20, beware of ice, and no passing signs.

Model Certainty

For the speed limit 20km/h sign, the model was 87% sure of its incorrect prediction although the correct class was its third choice. All of the top alternatives were speed limit signs.

Actual Class = Speed limit (20km/h)

1. Probability = 0.8760, Classname = Speed limit (60km/h)
2. Probability = 0.0736, Classname = Speed limit (30km/h)
3. Probability = 0.0303, Classname = Speed limit (20km/h)
4. Probability = 0.0153, Classname = Speed limit (80km/h)
5. Probability = 0.0023, Classname = Speed limit (120km/h)

For the stop sign, the model was 75% sure of its correct prediction, though it considered prioririty road to be a contender at 25%.

Actual Class = Stop

1. Probability = 0.7502, Classname = Stop
2. Probability = 0.2464, Classname = Priority road
3. Probability = 0.0029, Classname = No entry
4. Probability = 0.0003, Classname = No passing for vehicles over 3.5 metric tons
5. Probability = 0.0001, Classname = Speed limit (80km/h)

For the no entry sign, the model was 100% sure of its correct choice.

Actual Class = No entry

1. Probability = 1.0000, Classname = No entry
2. Probability = 0.0000, Classname = Speed limit (20km/h)
3. Probability = 0.0000, Classname = Speed limit (30km/h)

4. Probability = 0.0000, Classname = Speed limit (50km/h)
5. Probability = 0.0000, Classname = Speed limit (60km/h)

For the children crossing sign, the model was 100% sure of its correct choice.

Actual Class = Children crossing

1. Probability = 1.0000, Classname = Children crossing
2. Probability = 0.0000, Classname = Bicycles crossing
3. Probability = 0.0000, Classname = Slippery road
4. Probability = 0.0000, Classname = Right-of-way at the next intersection
5. Probability = 0.0000, Classname = Dangerous curve to the right

For the yield sign, the model was 100% sure of its correct choice.

Actual Class = Yield

1. Probability = 1.0000, Classname = Yield
2. Probability = 0.0000, Classname = Priority road
3. Probability = 0.0000, Classname = Children crossing
4. Probability = 0.0000, Classname = Ahead only
5. Probability = 0.0000, Classname = Speed limit (30km/h)

For the roundabout mandatory sign, the model was 100% sure of its correct choice.

Actual Class = Roundabout mandatory

1. Probability = 1.0000, Classname = Roundabout mandatory
2. Probability = 0.0000, Classname = Keep right
3. Probability = 0.0000, Classname = Turn right ahead
4. Probability = 0.0000, Classname = Ahead only
5. Probability = 0.0000, Classname = Go straight or right

For the beware of ice/snow sign, the model was more conflicted with all five of the alternatives having nontrivial probability weight. The correct choice was not among the top 5 options.

Actual Class = Beware of ice/snow

1. Probability = 0.5819, Classname = Slippery road
2. Probability = 0.1893, Classname = Beware of ice/snow
3. Probability = 0.1554, Classname = Pedestrians
4. Probability = 0.0589, Classname = Right-of-way at the next intersection
5. Probability = 0.0102, Classname = Children crossing

For the no passing sign, the model was fairly confident in its choice of general caution at 91%. Again, it failed to include the correct choice in the top 5 options.

Actual Class = No passing

1. Probability = 0.9145, Classname = General caution
2. Probability = 0.0452, Classname = Right-of-way at the next intersection
3. Probability = 0.0105, Classname = Speed limit (30km/h)

4. Probability = 0.0074, Classname = Slippery road
5. Probability = 0.0060, Classname = Traffic signals

(Optional) Visualizing the Neural Network

I did not complete the optional exercise.