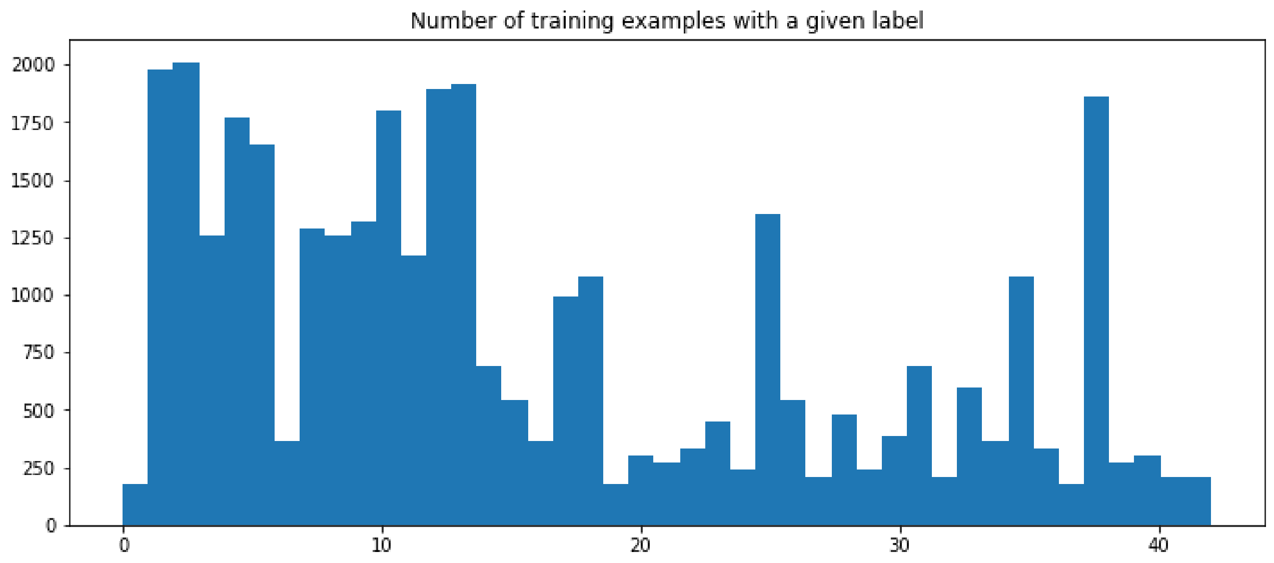
Project 2: Traffic Sign Recognition (June 2017)
Chris Venour

Section 1: Data Set Summary & Exploration

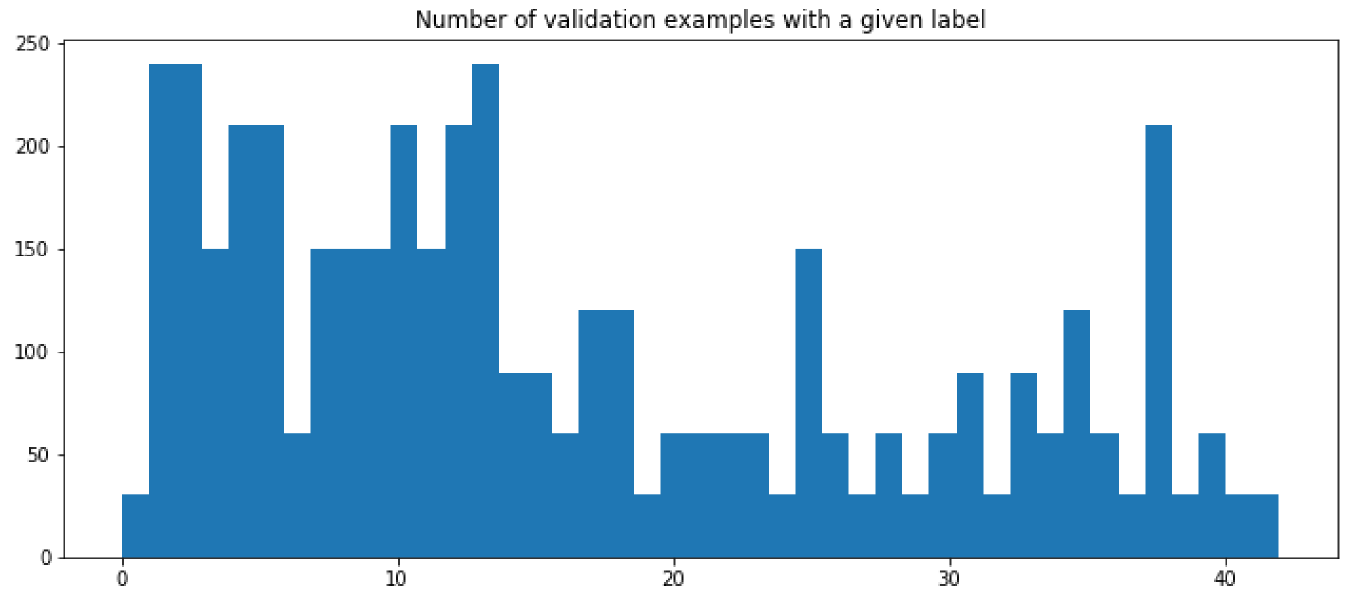
- The training set contains 34,799 traffic sign images
- The validation set contains 4410 images
- The test set contains 12,630 images
- The shape of a traffic sign image is 32x32x3
- The number of unique classes/labels in the data set is 43

Section 2: Exploratory visualization of the dataset

Below is a visualization of the training data set. It's a bar chart which shows how many example images there are per class and as you can see, the distribution is uneven. For example there are relatively fewer examples of signs that have labels 0,6,19,20,21 etc.



Below is a similar visualization of the validation data set. The distribution here is also uneven and is similar to the distribution of the training set.

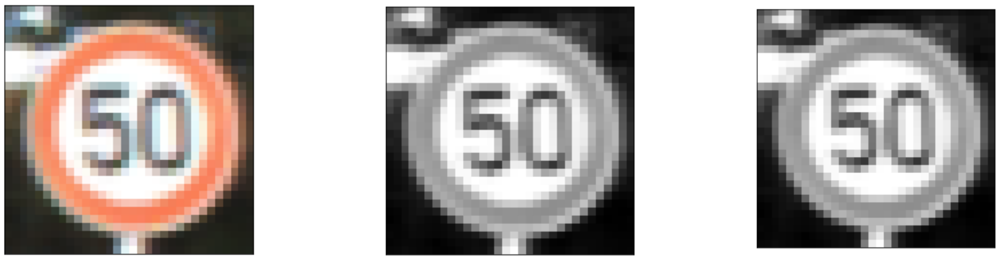


I did not examine any characteristics of the test set. I thought that set should remain untouched and unknown because I wanted to keep the final test of the model as objective and unbiased as possible.

Section 3: Design and Test a Model Architecture

Preprocessing

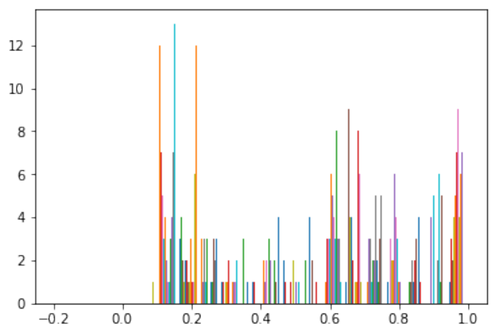
I converted the images to grayscale in order to speed up their processing by the convolutional network. I then normalized the gray-scaled images so that pixel values range between 0 and 1. Below is a sequence of 3 images which shows an image before processing, after grayscaling and then after normalization has been performed.



In order to make sure the pre-processing didn't mangle the data, I printed out the ranges of some randomly chosen normalized images, the means of the pixel values and the standard deviations. The print-out below shows these statistics for the normalized image shown above.

```
The range of the randomly chosen normalized image is 0.098231827112 to 1.00196463654
The mean is 0.555860398248
The std is 0.298733666847
```

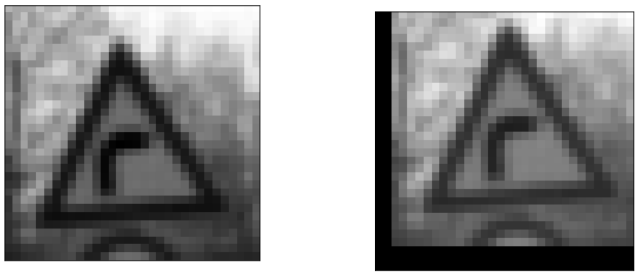
I also constructed histograms of the pixel values of randomly chosen normalized images. The histogram below shows the distribution of pixel values in the grayscaled and normalized picture shown above:

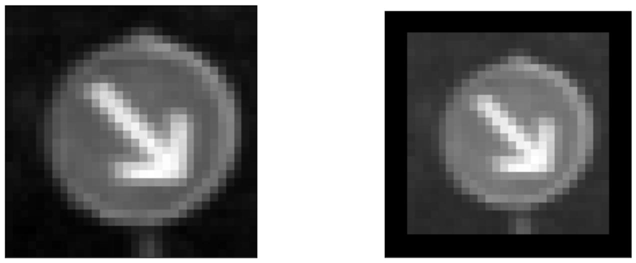


Augmenting the training data

I decided to augment the training data because acquiring more data is often the best way to improve the performance of a model. I generated additional images by translating them in the x and y directions by random amounts ranging from 0 to 3 pixels. I also zoomed out of the images, making the signs smaller, in order to augment the data. (I experimented with zooming **in** on images by a factor of 1.3 as well but including these images in the training data significantly decreased the accuracy of the model. These zoomed in images were therefore not used to augment the training set).

The row of images below shows an image before and then after translation and the next row shows an image before and after zooming outwards:





Initially there were 34,799 training examples but augmenting the data tripled the size of the training dataset to 139,196 examples.

Model architecture

The model is almost identical to the LeNet model except that I have added another fully-connected layer. My model consists of two convolutional layers, 2 max pooling layers, and 4 fully connected layers. Also, I use the ReLU activation function rather than the sigmoid function used in the LeNet model because recent research suggests that the ReLU achieves better results.

Layer	Description
Input	32x32x1 grayscale and normalized images
Convolution 5x5	1x1 stride, padding='valid', outputs 28x28x24
Relu	
Max Pooling	2x2 stride, padding='valid', outputs 14x14x24
Convolution 5x5	1x1 stride, padding='valid', outputs 10x10x64
Relu	
Max Pooling	2x2 stride, padding='valid' outputs 5x5x64
Fully connected	500
Relu	
Fully connected	200
Relu	
Fully connected	100
Relu	
Fully connected	43

Model Training

Accuracy of this model hovered around 91% when the system was trained with the given training set. I decided then to augment the training data set by translating and zooming out of the images. Augmenting the data in this way tripled the training set size from 34,799 to 139,196 examples. I also experimented with zooming **in** on images by a factor of 1.3 but including these images in the training data significantly decreased the accuracy of the model. These zoomed in images were therefore not used to augment the training set.

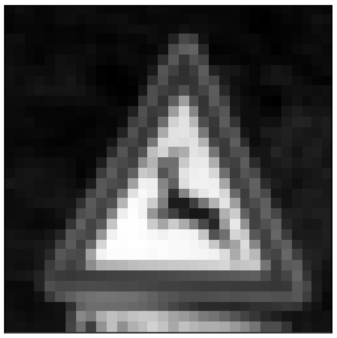
When I used the LeNet architecture with an Adam optimizer, a batch size of 128 and a learning rate of 0.0001 on the augmented data set, validation accuracy only barely exceeded 93% after 10 epochs and started fluctuating up and down at epoch 6. So I experimented by changing the activation function from the sigmoid function to ReLU. (Recent research suggests that the ReLU achieves better results than nearly all other activation functions).

I also added another fully connected layer because I thought that going from 1600 nodes (in the final convolution layer) to 43 classes in the space of just 3 fully connected layers was maybe too abrupt and might possibly throw away important information. This was an admittedly intuitive decision but given that there is no rigorous way right now of deciding beforehand on how many levels of architecture there should be for a given problem or what the values of certain hyperparameters should be, I simply tried this out and the accuracy of the system improved. When this layer was added to the model, an accuracy of 94.5% was achieved on the validation set after 6 epochs of training.

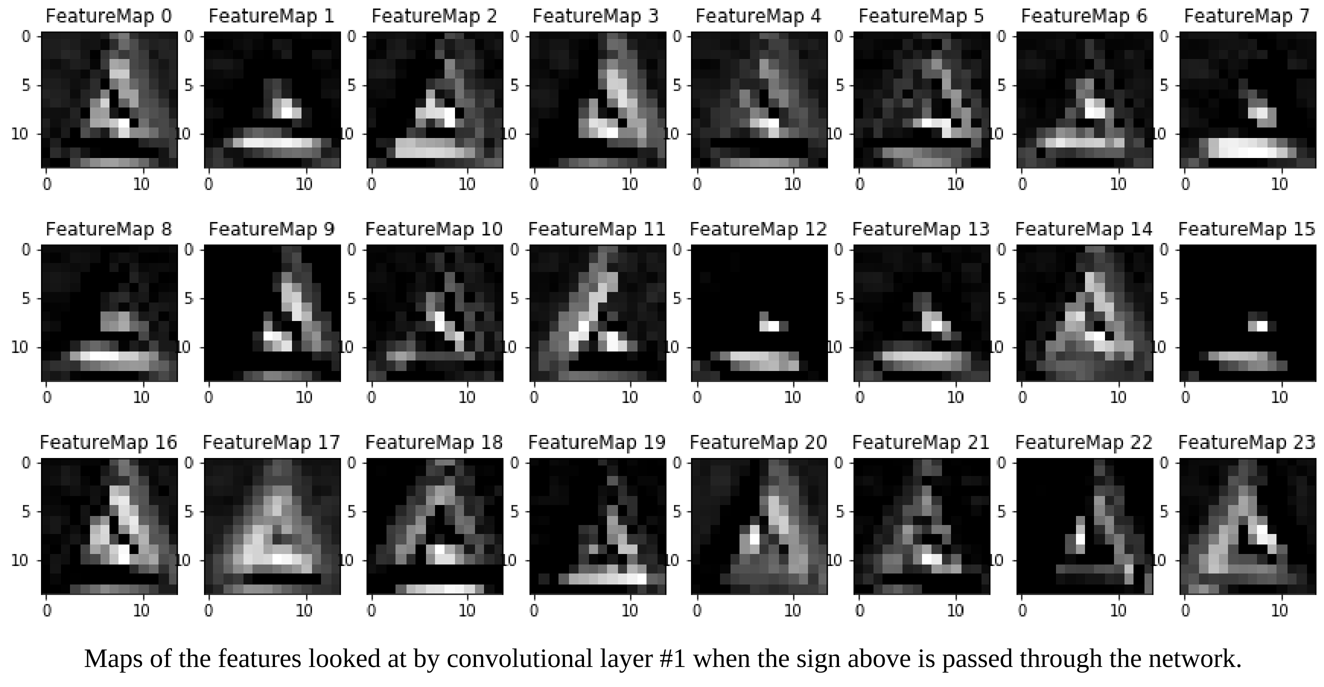
The final results were therefore a training set accuracy of 97%, validation set accuracy of 94.5% and a test set accuracy of 93.6%

Section 4: Visualize layers of the Neural Network

Below is the sign (Wild animals crossing) which I sent through the neural network in order to look at the salient features the network used to perform its classification task.

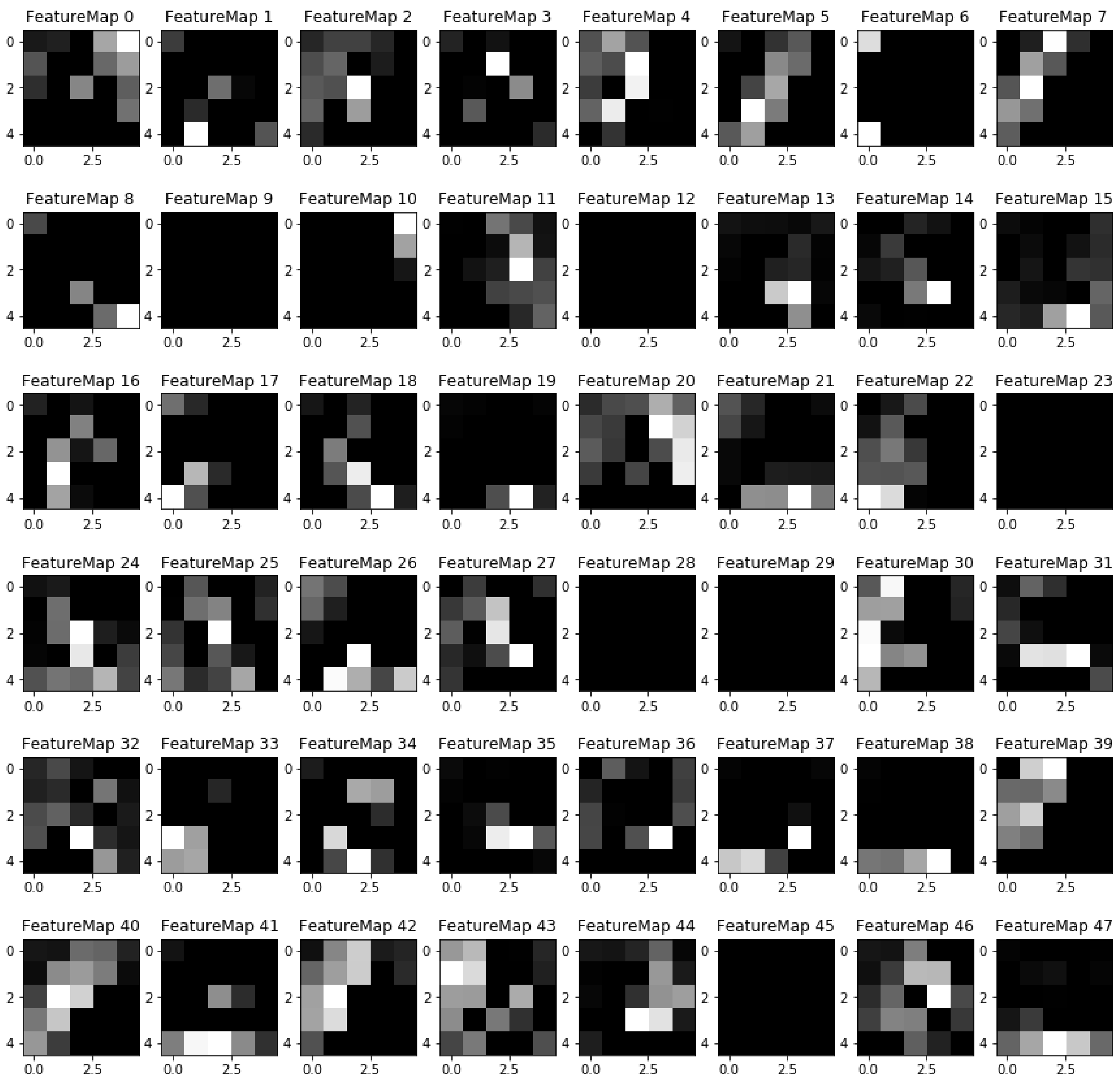


The following images show the features looked for by convolutional layer #1. For the most part it noticed large sections of triangles because the sign has a triangular shape.



Maps of the features looked at by convolutional layer #1 when the sign above is passed through the network.

The following images show the features looked for by convolutional layer #2 and these features appear to be more primitive kinds of shapes that the network looked for in order to classify the sign: horizontal lines in map 38, 41, 47 and diagonal lines in map 5, 7, 25 and 40.



Maps of the features looked at by convolutional layer #2 when a sign is passed through the network.