Traffic Sign Recognition Project Report

Andrew Megaris

Data Set Summary & Exploration

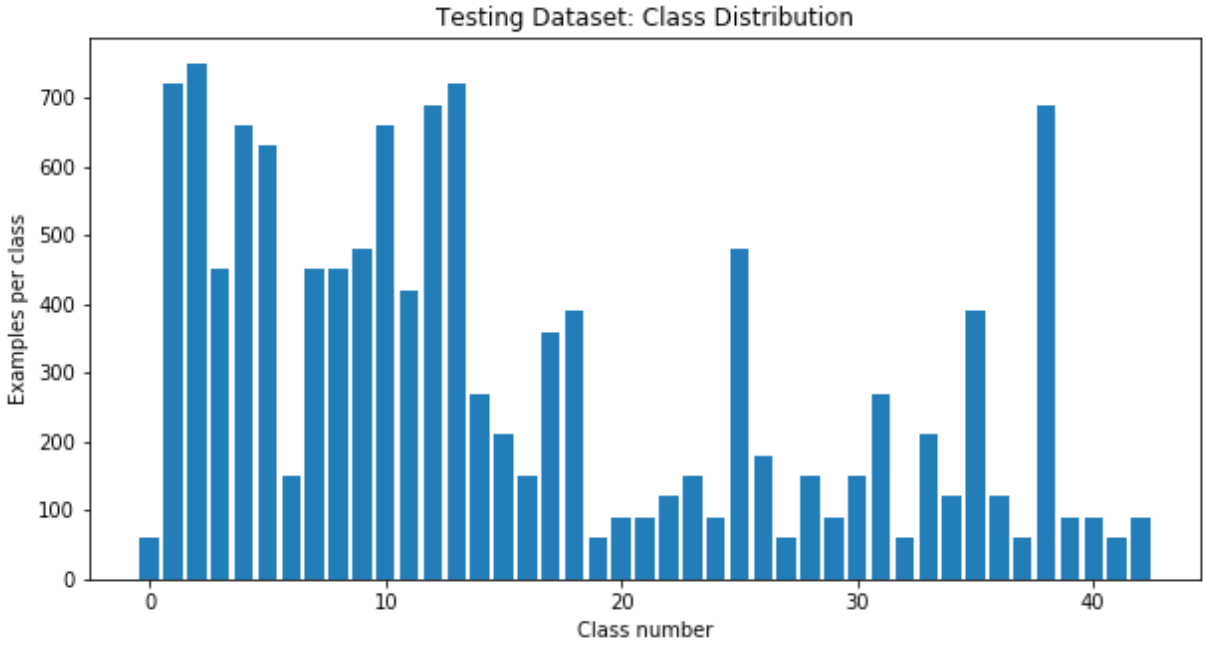
1. Provide a basic summary of the data set.
Here is the result of running code cell #3 showing a basic summary of the data.

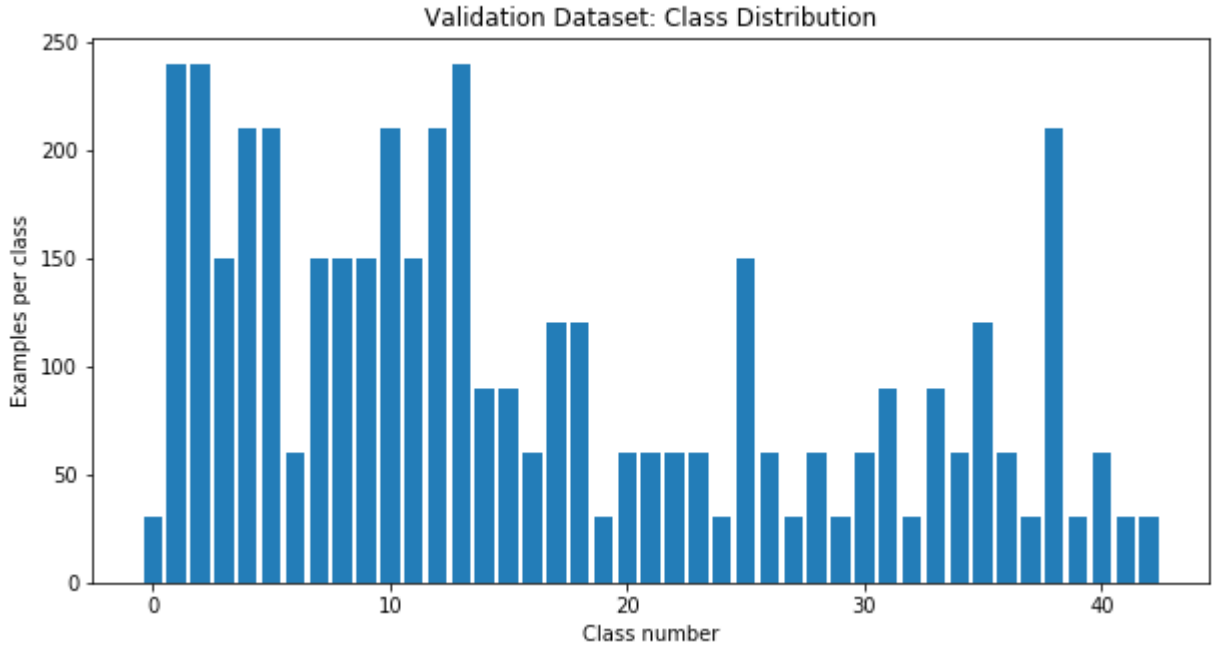
```
Number of training examples = 34799
Number of testing examples = 12630
Number of validation examples = 4410
Image data shape = (32, 32, 3)
Number of classes = 43
```

2. Include an exploratory visualization of the dataset.

Here is the result of running code cell #4 showing a sample of the images from the training set, this is good so we can get a human perspective of what data we are training on. This will allow us to make better choices when chosing preprocessing, learning archetectures..pretty much everything. As well there are graphs of the image class distributions for each of the data sets, I could/should generate additional images to make the distributions more even.


Training Dataset: Class Distribution
Examples per class
Class number
0
250
500
750
1000
1250
1500
1750
2000
0
10
20
30
40


Testing Dataset: Class Distribution
Examples per class
Class number
0
100
200
300
400
500
600
700
0
10
20
30
40



Design and Test a Model Architecture

1. Describe how you preprocessed the image data.

for this project I did not do a whole lot of preprocessing. I think this is because the images are already well cropped and the sign consumes most of the pixels. I did still choose a few performance prepocessing techniques. I used normalization and one-hot encoding, these generally increase the efficency of model training. I as well used Kera's ImageDataGenerator, although this class provides a LOT of preprocessing capabilities, I went easy any took only some minor distortions. It should be noted that you could also use this class to generator aditional data to normalize the class distribution in the sets.

Here is what I did to preprocess, the below code is from cell #5

In [7]:
```
# One-hot encoding
Y_train = np_utils.to_categorical(y_train, n_classes)
Y_valid = np_utils.to_categorical(y_valid, n_classes)
Y_test =  np_utils.to_categorical(y_test, n_classes)

# Normalizeing all of the datasets.
X_train = X_train.astype('float32') / 255
X_valid = X_valid.astype('float32') / 255
X_test = X_test.astype('float32') / 255
datagen = ImageDataGenerator(featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    rotation_range=0,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=False,
    vertical_flip=False)

# Compute quantities required for feature-wise normalization
# (std, mean, and principal components if ZCA whitening is applied).
datagen.fit(X_train)
```

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

My final model consisted of the following layers:

Layer	Description
Convolution	kernel = 3, stride = 3, padding = same
RELU	
Convolution	kernel = 3, stride = 3
RELU	
Max Pooling	pool size = 2x2
Dropout	units = 0.25
Convolution	kernel = 3, stride = 3, padding = same
RELU	
Convolution	kernel = 3, stride = 3
RELU	
Max Pooling	pool size = 2x2
Dropout	units = 0.25
Flatten	
RELU	
Dropout	units = 0.8
Dense	
SOFTMAX	

3. Describe how you trained your model.

I used kera's default Adam Optimizer to train my model, it worked quiet well for this project. The default Adam Optimizer has a learning rate of 0.001, after a

couple of tests I found it to be the best. Finding a good batch size wasnt too hard, I try to keep to powers of two, so I found 64 worked better than 32 or 128. Concerning the number of epochs, I'm still not too certain, I want to be sure I don't overfit my model, but I don't know exactly where that point lies. I settled with 15 epochs, it seemed that this was right about where improvement vs time/possibly overfitting seemed to tip in my mind.

4. Describe the approach taken for finding a solution and getting the validation set accuracy to be at least 0.93.

I had found Cifar10 while I was looking for images actually. I had implemented Lenet originally and was not super pleased with the results. When I was searching for pre-sized images to test on my Lenet setup, I stumbled across Cifar10, which was designed for classifying small images. I saw that it was quiet simple to set up with Keras and I wanted more experience with the API so I went for it. It turns out that it was super easy to implement and there was a LOT of reference matieral on the internet. Keras provides a sample Cifar 10 example, I built mine modeled after this example from a github.

https://github.com/fchollet/keras/blob/master/examples/cifar10_cnn.py

I liked Cifar10 because it took FAR less epochs to get into the upper 90's for accuracy, this made me feel better about not over fitting my model. As far as tuning paramets, I was more or less happy with all of the default values, I tried changing some of the dropout values, and kernel sizes but it all degregated performance.

My final model results are shown below(code cell 7 and 8):

```
Epoch 14/15
34799/34799 [==============================] - 191s - loss: 0.0941 - acc: 0.9711 - val_loss: 0.0697 - val_acc: 0.9785
Epoch 15/15
34799/34799 [==============================] - 192s - loss: 0.0909 - acc: 0.9732 - val_loss: 0.0767 - val_acc: 0.9834
```

```
In [27]: #evaluate test set
         model.evaluate(X_test, Y_test, batch_size=batch_size, verbose=1)

         12630/12630 [==============================] - 20s

Out[27]: [0.1193455403754731, 0.97165479022929901]
```

Training   Accuracy : 97.32%
Validation Accuracy : 98.34%
Testing   Accuracy : 97.16%

Test a Model on New Images

1. Choose five German traffic signs found on the web and provide them in the report.

Here are five German traffic signs that I found on the web, I think they should all be pretty easy for the model to classify. (code cell 9)

---

2. Discuss the model's predictions on these new traffic signs and compare the results to predicting on the test set.

My model made all 5 predictions correctly. I think these images where pretty easy to classify so I am not supprised. I think the result goes with what was expected. Five is such a small sample size, if you took five predictions from a data set of 40,000 at 98% accuracy your odd are very high that you will get five correct predictions. I think if I wanted to get wrong results, images with lighting changes across the sign, and images that are from more obsure angles. (code cell 10)

---

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.

I think it is safe to say that my model was VERY certain on every prediction, you can see that on images 1,3, and 4 it was predictiing with 100% certainty, on images 2, and 5 it was over 99.9% certainty, which I believe is pretty good.

```
Top 5 softmax probabilities for test image 1:

TopKV2(values=array([[  1.00000000e+00,   4.12936291e-19,   1.53166673e-24,
         1.31118385e-24,   7.06106007e-27]], dtype=float32), indices=array([[18, 26, 24, 19, 27]], dtype=int32))

Top 5 softmax probabilities for test image 2:

TopKV2(values=array([[  9.99998450e-01,   1.55828161e-06,   5.03506785e-11,
         1.32642630e-11,   2.56581475e-12]], dtype=float32), indices=array([[15,  4,  7,  5,  1]], dtype=int32))

Top 5 softmax probabilities for test image 3:

TopKV2(values=array([[  1.00000000e+00,   2.41765538e-13,   1.47972487e-17,
         6.16833066e-21,   1.95697301e-21]], dtype=float32), indices=array([[38,  5,  7, 30, 40]], dtype=int32))

Top 5 softmax probabilities for test image 4:

TopKV2(values=array([[  1.00000000e+00,   5.02580382e-16,   6.65231619e-17,
         6.08207500e-18,   4.85138410e-18]], dtype=float32), indices=array([[25, 30, 11, 28, 26]], dtype=int32))

Top 5 softmax probabilities for test image 5:

TopKV2(values=array([[  9.99412298e-01,   5.87742368e-04,   8.11641065e-09,
         6.43376330e-09,   1.59648780e-10]], dtype=float32), indices=array([[ 3,  5,  0, 16, 14]], dtype=int32))
```