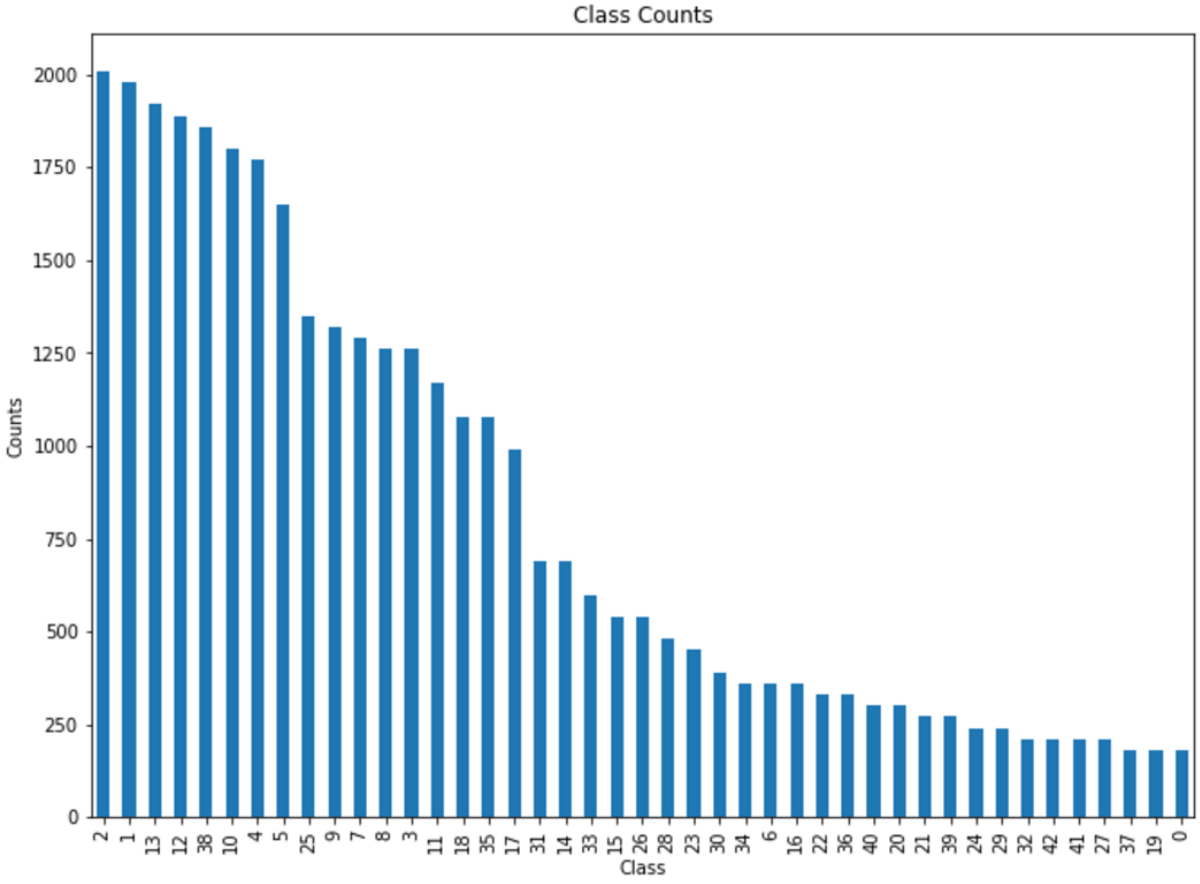
Traffic Sign Classifier

Data Set Summary and Exploration

The data set consist of more than 30,000 training, 4,000 validation and 12,000 testing images. These images were taken from German Traffic Sign Dataset. The dataset had images that represent 43 different types of German traffic signs in 32x32x3 (RGB) format. Numpy and Pandas was use get basic data statistics ([2] – Code block 2).

Visualization of Dataset

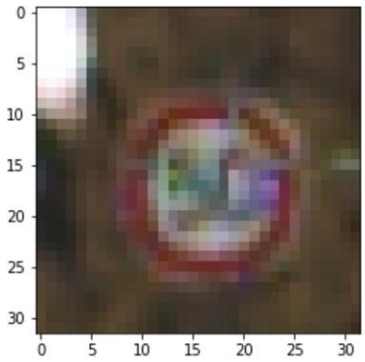
The following graph ([4]) show how frequent images exist in each class.



The largest class, class 2, had the highest image count of 2010. Class 0 only have 180 images. Here are a few examples of image and class representations ([6]-[10]).

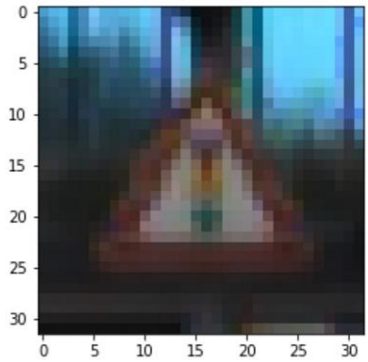
```
#Show a picture from class 2 (Speed limit (50km/h)
plt.imshow(X_train[y_train == 2][0])
```

```
<matplotlib.image.AxesImage at 0x1b90b4bc198>
```



```
#Show a picture from class 26 (Traffic signals)
plt.imshow(X_train[y_train == 26][0])
```

```
<matplotlib.image.AxesImage at 0x1b90b5f39b0>
```

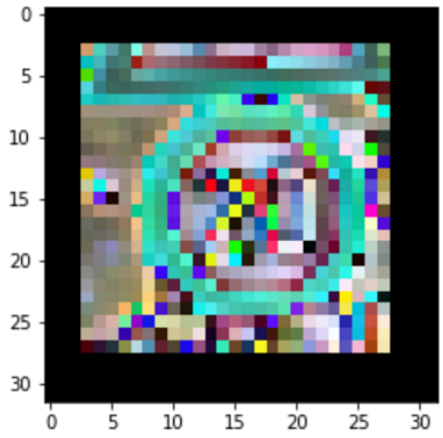


Pre-processing the Training Data Set

I first scale, I called it normalize ([17]), the color values from 0-255 to -1 to 1. This preserve data while reducing overfitting through ReLU activation. I then balance all classes to have 2010 images ([18]) by making either entire set copy and/or random copy from set. Now that all classes have 2010 images, I took 35% from each class to apply random augmentation ([21]). The augmentation consists of either rotate, pixel dropout, shrink, enlarge and shift. Pixel dropout sets 20% of image pixel, at random, to 0 – Inspired by fully connected dropout. Pixel drop should also prevent overfitting and the model to learn more important features of an image. Here is an example of an augmented (shrink) image:

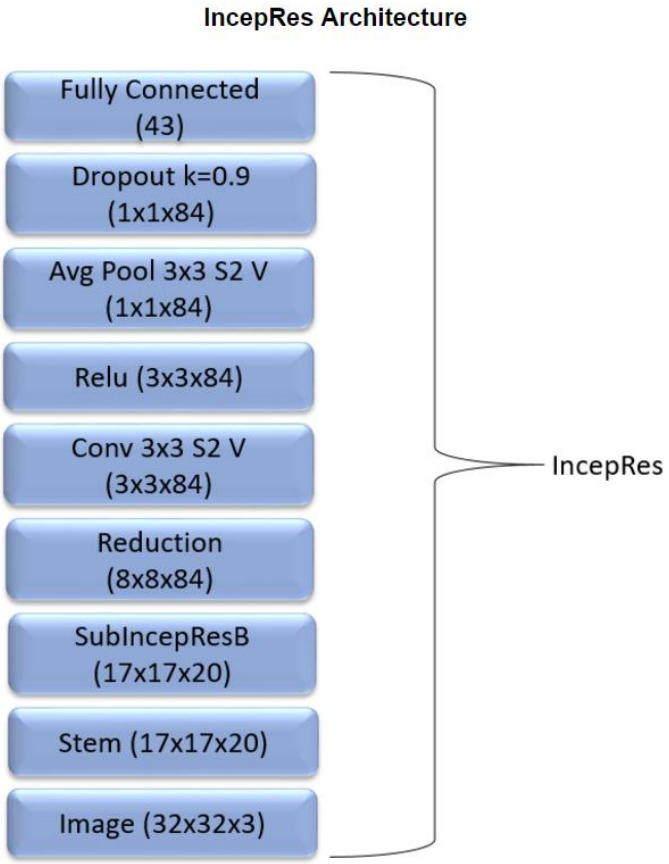
```
#Take a look at an image to make sure class and modification worked
plt.imshow(X_train_mod[y_train_mod==0][500])
```

```
<matplotlib.image.AxesImage at 0x1b918f00048>
```



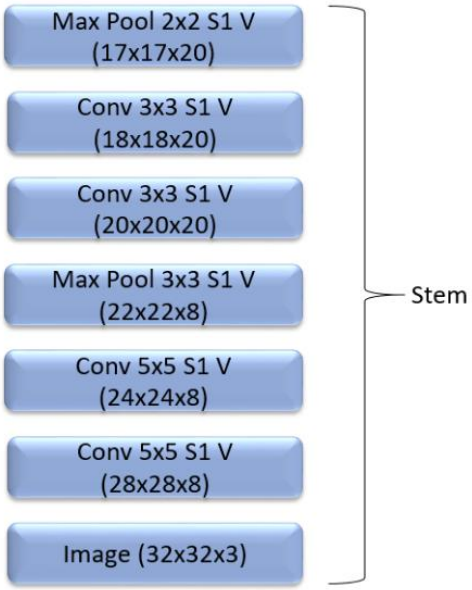
Final Model

Overview: The model started from a quest to learn about inception. IncepRes ([20]) is mainly based on Google's Inception-ResNet-v2 that produced the best result on ILSVRC 2012 in a publication (https://arxiv.org/pdf/1602.07261.pdf). Output shape are specified like (17x17x20)



Stem: Next to layer name; filter size (e.g. 3x3), stride (e.g. S1) and padding (e.g. V) are displayed. The picture below describes the "Stem" of the architecture. The purpose of the stem is to reduce the size of image from 32x32 to 17x17. I arbitrarily choose a depth of 20 layers (4 more than LeNet). I used convolution and max pooling layers without ReLU activation to reduce the size.

Stem Sub-Architecture



Next is Inception-ResNet-B from Google's Inception-ResNet-v2. Every layer has same padding to output the same shape as input (17x17x20). The graph does not explicit show filter concatenation when convolution branch merge, however, that is what I did. The paper also explains that residual (convolution layers) should be scaled between 0.1 and 0.3 before adding back to identity. I scaled the last 1x1 convolution layer with Batch Normalization with scaling of 0.3.

Inception-ResNet-B Sub-Architecture

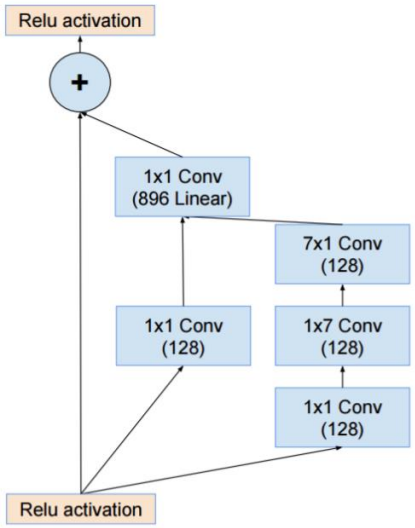


Figure 11. The schema for 17 × 17 grid (Inception-ResNet-B) module of Inception-ResNet-v1 network.

Then I modified Reduction-B to reduce 17x17x20 down to 8x8x84. The depth of convolution consists of 3 layers of 20 deep and 1 layer of 24 deep.

Reduction-B Sub-Architecture

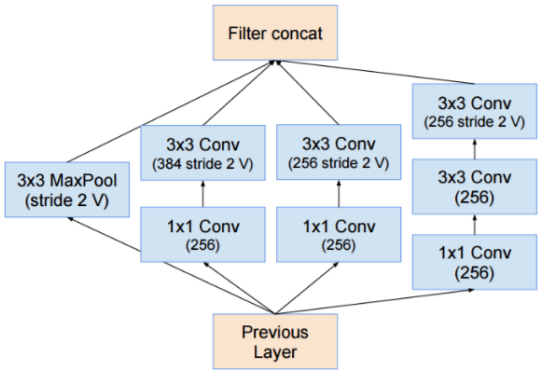


Figure 12. "Reduction-B" 17×17 to 8×8 grid-reduction module. This module used by the smaller Inception-ResNet-v1 network in Figure 15.

Since 84 was like the number of Fully Connected nodes at the end of LeNet, the last part of the architecture further reduces 8x8x84 to 1x1x84. I tested with dropout and keep probability of 0.8, 0.88, 0.9, and 1.0.

I also test with adding 3 more Inception-ResNet-C (IncepRes_V1), with size 3x3x84, after reducing 8x8x84 to 3x3x84. The resulting validation accuracy was much less (90% best validation) compared to final model. LeNet was also much less (88% best validation) compared to final model.

Train, Validate and Test Final Model

First I apply the same scale (normalization) to the validation and test set ([21]). Then I built the training pipeline ([24]):

1. Build IncepRes and store the logits as a vector (43 classes).
2. Calculate Softmax Cross Entropy using logits.
3. Use Adam Optimizer to reduce mean of Cross Entropy.

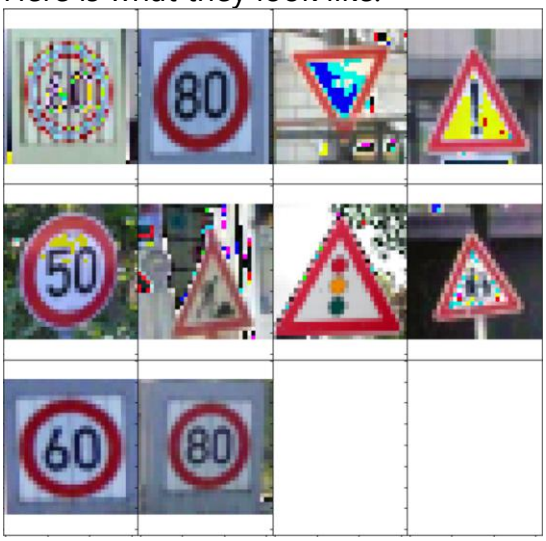
I created a script ([60]) to try multiple sets of hyperparameter to find the best model hyperparameters ([89]):

- Best Validation Accuracy: 0.981405895178
- Best Dropout Keep Probability: 0.88
- Best learning rate: 0.001

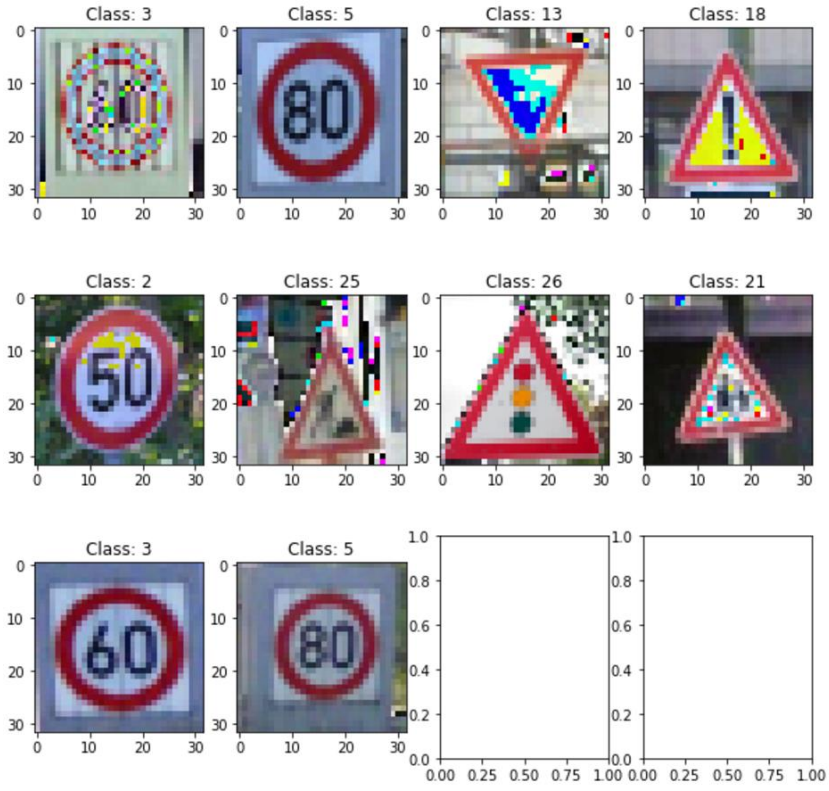
- Best Epoch: 33
- Best Batch Size: 256

Test Model on New Images

I first wrote a script ([30]) to build a new test data set based on 10 pictures found on Google Street View. I then apply the same scaling (normalizing) to the new test data set. Here is what they look like:



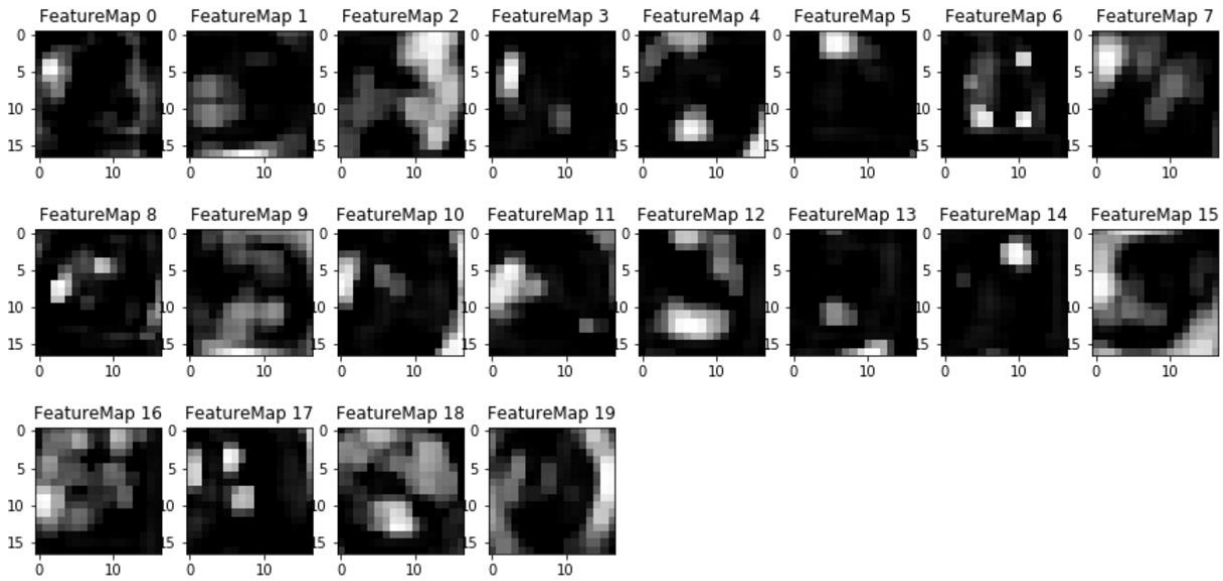
I then feed new test set through the model to show what class was predicted:



The only image that was incorrectly predicted was the image with “Class: 21”. The sign of children crossing had more shrinking and blurring compared to what the training set had seen. The model had more than 99% confidence for the other signs ([96]).

Visualize Deep Learning

I used the first image of the new test set to visualize how the weight had learn how to classify an image:



Feature map 19 Looks like it can be a 0 as part of "60". The size of "0" in the image is about 17 pixels. Feature map 16 basically looks like a "6". The other feature looks like parts of "6". Combining it together the classifier was basically 100% confident that it was 60 kph speed limit sign.