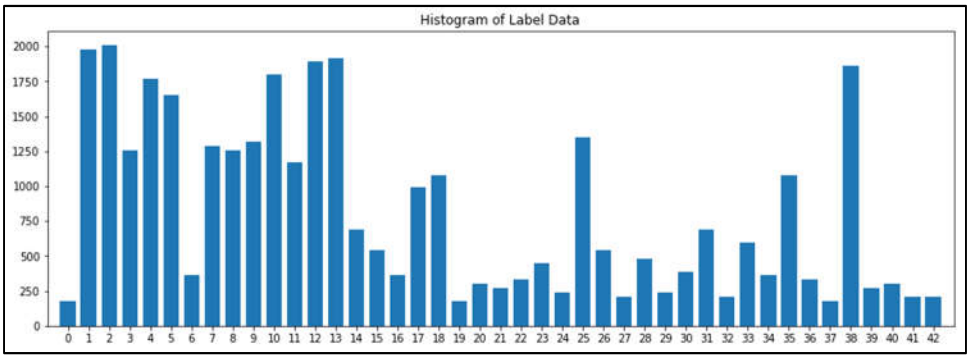
PROJECT WRITEUP: German Traffic Sign Classification

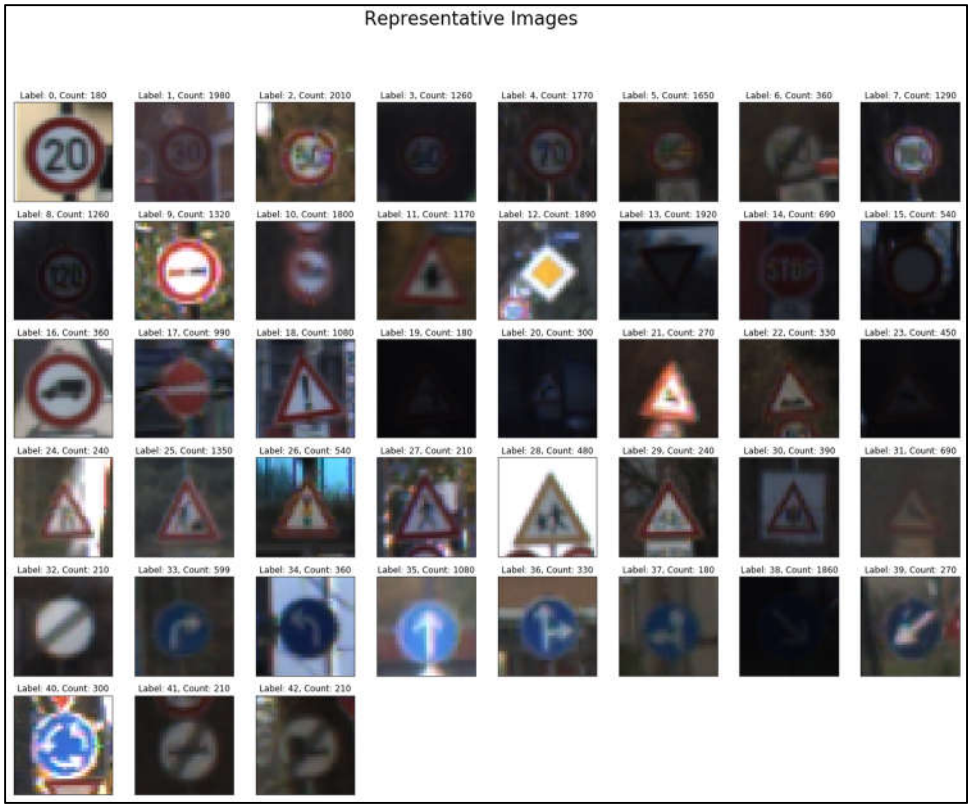
Dataset Exploration

Dataset Summary & Exploratory Visualization

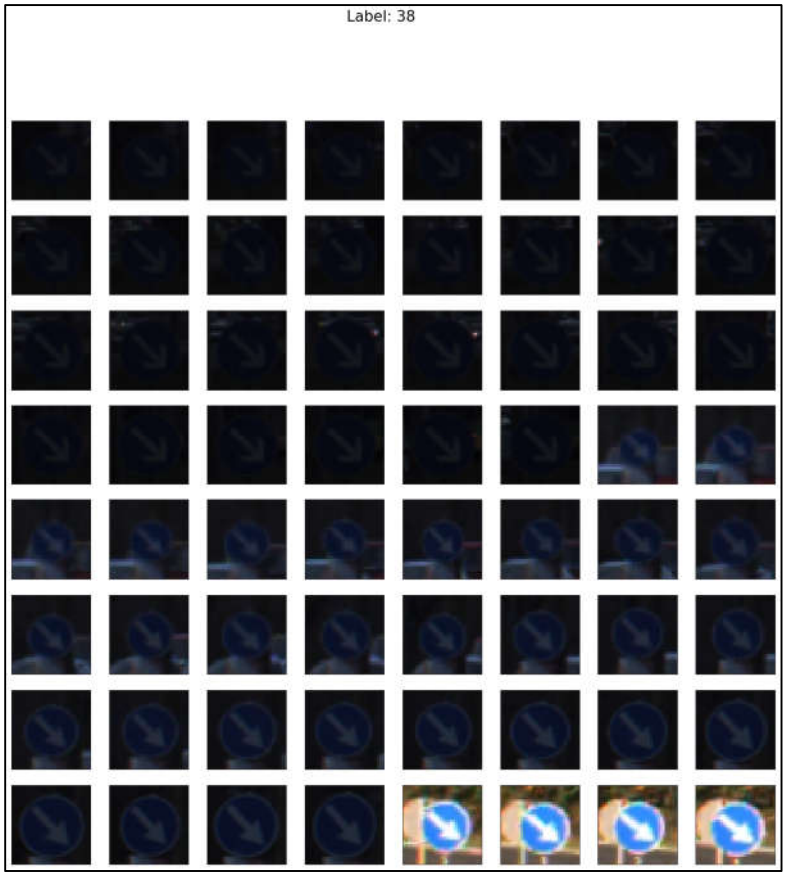
First step was to take a look at the data in order to make a plan for preprocessing and architecture design. On a high-level view, I could see that there was a relatively small sample of images for training, which amounted to roughly 800 images per label, and a very large variance of samples across labels. Plotting a histogram, I could see that there was quite a wide range of samples, which I would address in the preprocessing step.



I plotted a single sample of each label to get an idea of what each sign looked like. Once plotted, I could see that a number of the signs looked extremely dark and in some cases almost totally black.



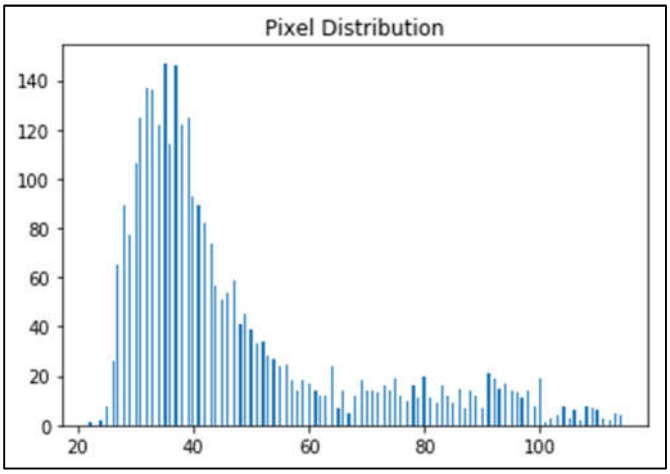
I wrote another function to see a number of samples of a single label to see whether this was the case across the board. Looking at a group of 64 of one of the dark labels, I could see that some were more visible than the first sample. Another characteristic of the training data that I would try to address in preprocessing.

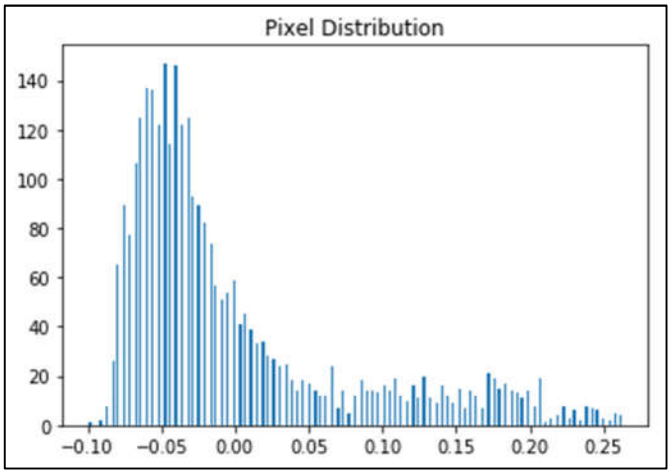


Design and Test a Model Architecture

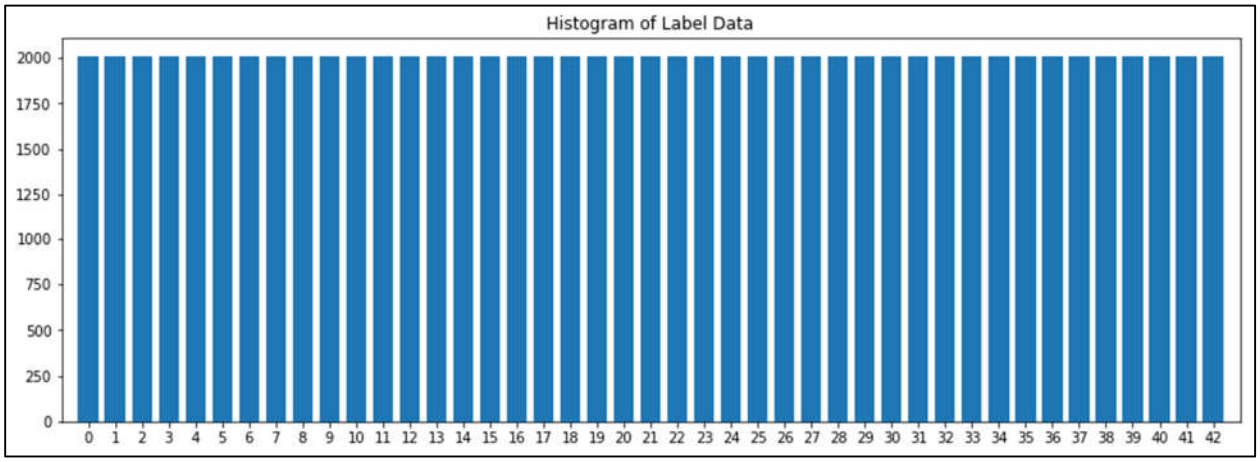
Preprocessing

The first step that I took in preprocessing was to normalize image data to roughly have zero mean and equal variance. Instead of dividing by 255 and shifting over by .5, I divided each image pixel values by 255 then shifted values by the mean of that range. I thought this would help correct for the darkness of some of the images.



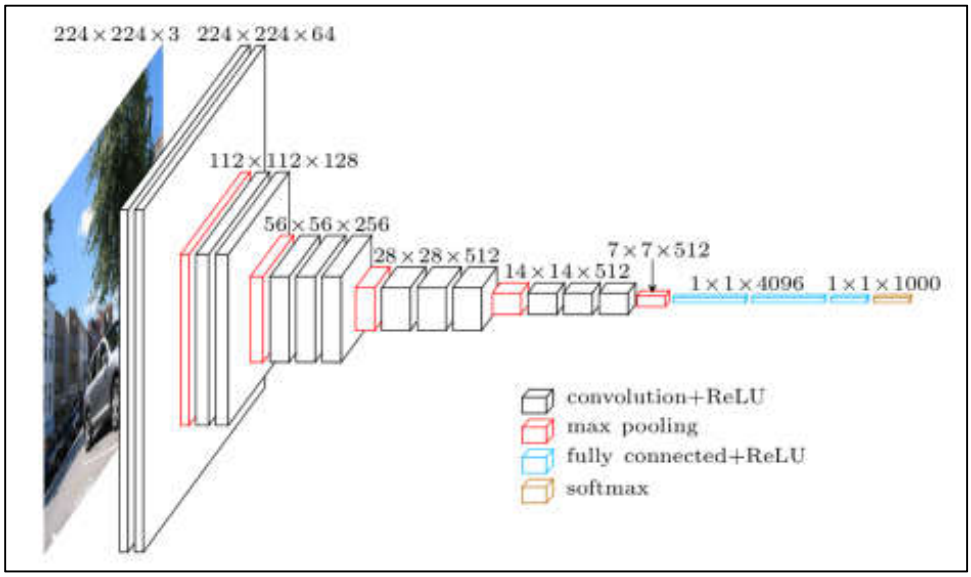


Lastly, I normalized the distribution of images so that each label would have an equal number of samples. I did this by adding duplicate copies of each label set to the point that each bucket matched the number of samples in the max bucket. By doing this, the model wouldn't 'specialize' in classifying images that it had seen most frequently. These images were further processed one last time when being fed into the model by the Keras ImageDataGenerator, which randomly applied rotation, width and height shifts to each image. By doing this, the risk of overfitting images with multiple copies in augmented label buckets was reduced.



Model Architecture

Initially, I wanted to use this project as an opportunity to implement a transfer learning model with Keras, both of which I had learned about but had not implemented from scratch. The plan was to use a pre-trained VGG16 model with fixed weights and train a few fully connected layers on the end. I had read about SELU activations and self-normalizing networks and wanted to experiment with this, but ended up going with a more plain vanilla approach as SELU and associated weight initializations are not available with the Keras version used in the *carnd-term1* environment.



I started with just training the end layers that I added to the model, thinking that the broad range of images in the *imagenet* dataset that yielded the pre-trained VGG16 model may have sufficient images with overlapping structural similarity to allow generalization on the German Traffic Sign Recognition

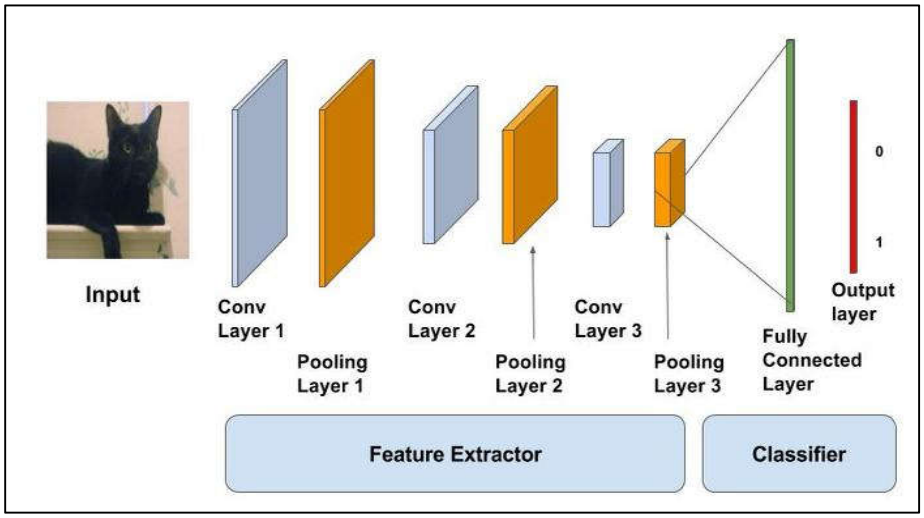
Benchmark (GTSRB). Seeing that this was not the case, I extended the training further and further back up the convolutional layers. In the final architecture, the first 3 convolutional blocks were fixed – as these would likely be highly generalizable features like edges and shapes, and the last 2 convolutional blocks were trained. The output of the VGG16 model was flattened via global average pooling. From there, I applied a dropout layer before the first fully connected layer, batch normalization and 'relu' activation before the last fully connected classification layer with a 'softmax' activation.

```
In [13]: # Model architecture overview
         model.summary()
```

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 32, 32, 3)	0	
block1_conv1 (Convolution2D)	(None, 32, 32, 64)	1792	input_1[0][0]
block1_conv2 (Convolution2D)	(None, 32, 32, 64)	36928	block1_conv1[0][0]
block1_pool (MaxPooling2D)	(None, 16, 16, 64)	0	block1_conv2[0][0]
block2_conv1 (Convolution2D)	(None, 16, 16, 128)	73856	block1_pool[0][0]
block2_conv2 (Convolution2D)	(None, 16, 16, 128)	147584	block2_conv1[0][0]
block2_pool (MaxPooling2D)	(None, 8, 8, 128)	0	block2_conv2[0][0]
block3_conv1 (Convolution2D)	(None, 8, 8, 256)	295168	block2_pool[0][0]
block3_conv2 (Convolution2D)	(None, 8, 8, 256)	590080	block3_conv1[0][0]
block3_conv3 (Convolution2D)	(None, 8, 8, 256)	590080	block3_conv2[0][0]
block3_pool (MaxPooling2D)	(None, 4, 4, 256)	0	block3_conv3[0][0]
block4_conv1 (Convolution2D)	(None, 4, 4, 512)	1180160	block3_pool[0][0]
block4_conv2 (Convolution2D)	(None, 4, 4, 512)	2359808	block4_conv1[0][0]
block4_conv3 (Convolution2D)	(None, 4, 4, 512)	2359808	block4_conv2[0][0]
block4_pool (MaxPooling2D)	(None, 2, 2, 512)	0	block4_conv3[0][0]
block5_conv1 (Convolution2D)	(None, 2, 2, 512)	2359808	block4_pool[0][0]
block5_conv2 (Convolution2D)	(None, 2, 2, 512)	2359808	block5_conv1[0][0]
block5_conv3 (Convolution2D)	(None, 2, 2, 512)	2359808	block5_conv2[0][0]
block5_pool (MaxPooling2D)	(None, 1, 1, 512)	0	block5_conv3[0][0]
globalaveragepooling2d_1 (Global	(None, 512)	0	block5_pool[0][0]
dropout_4 (Dropout)	(None, 512)	0	globalaveragepooling2d_1[0][0]
dense_3 (Dense)	(None, 1024)	525312	dropout_4[0][0]
batchnormalization_1 (BatchNorma	(None, 1024)	4096	dense_3[0][0]
activation_7 (Activation)	(None, 1024)	0	batchnormalization_1[0][0]
dense_4 (Dense)	(None, 43)	44075	activation_7[0][0]

```
Total params: 15,288,171
Trainable params: 13,550,635
Non-trainable params: 1,737,536
```

For some reason, the validation loss was not improving with the pre-trained VGG16 model, so I thought I would run the dataset through a Lenet architecture to see if I would get similar results.



On the initial Lenet model, validation losses did not flatten out as they did with VGG, so I went ahead with adding layers and tweaking Lenet to come to a final architecture. In the model, I made two convolutional blocks with the same structure – convolution/convolution/maxpool/dropout. On the end were two fully connected layers with one more dropout layer.

```
In [11]: # Model architecture overview
         model.summary()
```

Layer (type)	Output Shape	Param #	Connected to
convolution2d_1 (Convolution2D)	(None, 30, 30, 32)	896	convolution2d_input_1[0][0]
activation_1 (Activation)	(None, 30, 30, 32)	0	convolution2d_1[0][0]
convolution2d_2 (Convolution2D)	(None, 28, 28, 32)	9248	activation_1[0][0]
activation_2 (Activation)	(None, 28, 28, 32)	0	convolution2d_2[0][0]
maxpooling2d_1 (MaxPooling2D)	(None, 14, 14, 32)	0	activation_2[0][0]
dropout_1 (Dropout)	(None, 14, 14, 32)	0	maxpooling2d_1[0][0]
convolution2d_3 (Convolution2D)	(None, 12, 12, 64)	18496	dropout_1[0][0]
activation_3 (Activation)	(None, 12, 12, 64)	0	convolution2d_3[0][0]
convolution2d_4 (Convolution2D)	(None, 10, 10, 64)	36928	activation_3[0][0]
activation_4 (Activation)	(None, 10, 10, 64)	0	convolution2d_4[0][0]
maxpooling2d_2 (MaxPooling2D)	(None, 5, 5, 64)	0	activation_4[0][0]
dropout_2 (Dropout)	(None, 5, 5, 64)	0	maxpooling2d_2[0][0]
flatten_1 (Flatten)	(None, 1600)	0	dropout_2[0][0]
dense_1 (Dense)	(None, 1024)	1639424	flatten_1[0][0]
activation_5 (Activation)	(None, 1024)	0	dense_1[0][0]
dropout_3 (Dropout)	(None, 1024)	0	activation_5[0][0]
dense_2 (Dense)	(None, 43)	44075	dropout_3[0][0]
activation_6 (Activation)	(None, 43)	0	dense_2[0][0]

```
Total params: 1,749,067
Trainable params: 1,749,067
Non-trainable params: 0
```

Model Training

I ran both models for 100 epochs, and saved the best model of each using the Keras 'ModelCheckpoint' callback. I used a batch size of 256, as I have read that its good to use the largest batch size memory will allow, and to use a binary sequence number which better optimizes memory usage.

Solution Approach

Running the final architecture of each model for 100 epochs resulted in acceptable test results for the Lenet model (95.8%), while the VGG16 model did not (82.8%). I adjusted hyperparameters and architectures of each over many iterations – discussed briefly above – to arrive at these results.

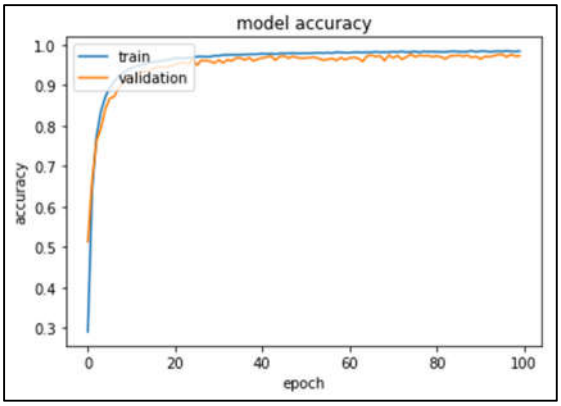
```
# Model performance on test set
score = lenet.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

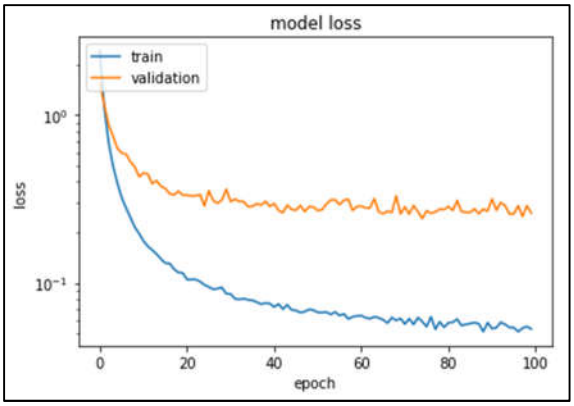
```
Test loss: 0.235536605868
Test accuracy: 0.957957244703
```

```
# Model performance on test set
score = vgg16.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

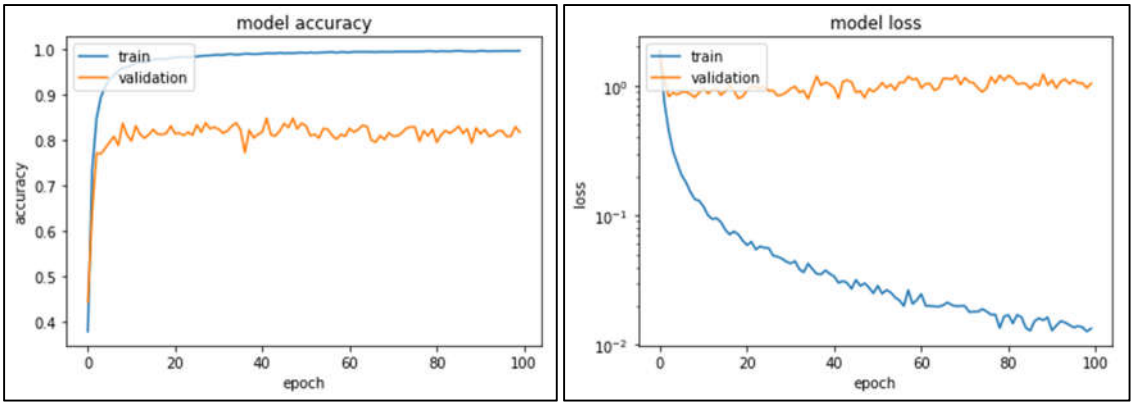
```
Test loss: 0.838180618145
Test accuracy: 0.828028503572
```

Lenet Accuracy & Loss





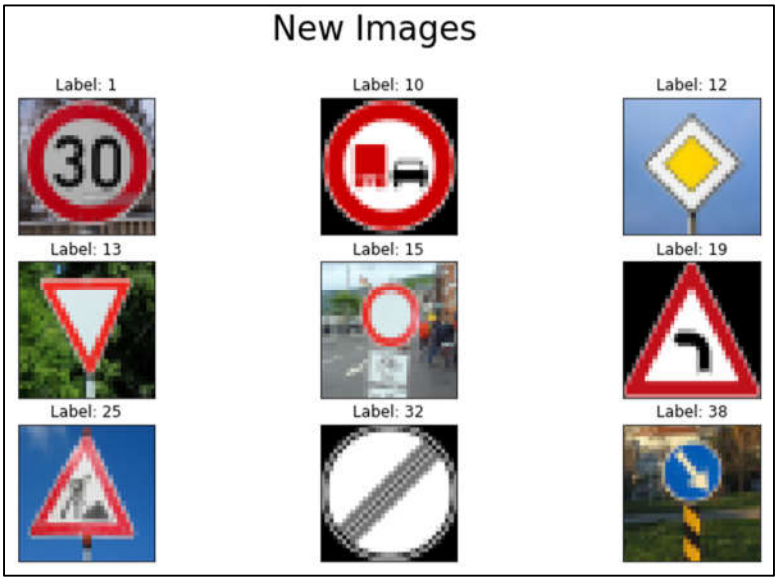
VGG16 Accuracy & Loss



Test a Model on New Images

Acquiring New Images

To test the model on new images, I chose 9 german street signs from the internet. Of the 9 images, 5 had a high number of images in the original dataset (1800-1980), one had slightly less (1350), and three had a small number (180-540). I wanted to choose a group from the underrepresented group to see if there was a difference in performance between them, or whether preprocessing and image augmentation may have addressed this.



Performance on New Images

On the new images, both images performed the same (78% accurate) – misclassifying the same two images. Although both produced the same classification outputs, the Lenet model had a much lower loss.

Comparative Performance: LeNet vs. VGG16

In [388]:
```
# LeNet vs. VGG16
lenet_new = lenet.evaluate(new_imgs_p, new_labels, verbose=0)
vgg16_new = vgg16.evaluate(new_imgs_p, new_labels, verbose=0)
print('LeNet Accuracy: %.3f%%'% (lenet_new[1]*100))
print('VGG16 Accuracy: %.3f%%'% (vgg16_new[1]*100))
print("")
print('LeNet Loss: %.5f'% (lenet_new[0]))
print('VGG16 Loss: %.5f'% (vgg16_new[0]))
```

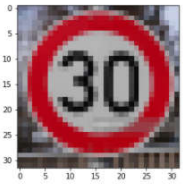
```
LeNet Accuracy: 77.778%
VGG16 Accuracy: 77.778%

LeNet Loss: 0.52071
VGG16 Loss: 1.22412
```

Model Certainty - Softmax Probs

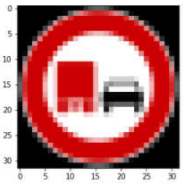
While most classifications were close to 100%, in cases of doubtful or incorrect classification, Lenet was more certain about the correct classification and less certain about incorrect classifications (see below).

LeNet Results

```
In [41]: show_preds(lenet_idx, lenet_prob)
```



```
Predictions for Sample 1: Speed limit (30km/h)

Prediction 1: Speed limit (30km/h)                    ==>     99.9998%
Prediction 2: Speed limit (20km/h)                    ==>      0.0002%
Prediction 3: Speed limit (50km/h)                    ==>      0.0000%
Prediction 4: Speed limit (80km/h)                    ==>      0.0000%
Prediction 5: Speed limit (70km/h)                    ==>      0.0000%

***Prediction is Correct***
```

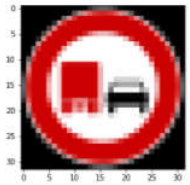


```
Predictions for Sample 2: No passing for vehicles over 3.5 metric tons

Prediction 1: No passing                              ==>     70.5987%
Prediction 2: No passing for vehicles over 3.5 metric tons ==>     29.4013%
Prediction 3: No entry                                ==>      0.0000%
Prediction 4: Yield                                   ==>      0.0000%
Prediction 5: Road work                               ==>      0.0000%

***Prediction is Incorrect***
```

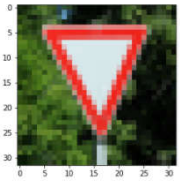
VGG16 Results

```
In [24]: show_preds(vgg_idx, vgg_prob)
```

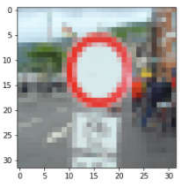
```
Predictions for Sample 1: Speed limit (30km/h)

Prediction 1: Speed limit (30km/h)                    ==>     99.9940%
Prediction 2: Speed limit (50km/h)                    ==>      0.0053%
Prediction 3: Speed limit (20km/h)                    ==>      0.0007%
Prediction 4: Speed limit (70km/h)                    ==>      0.0000%
Prediction 5: Road work                               ==>      0.0000%

***Prediction is Correct***
```



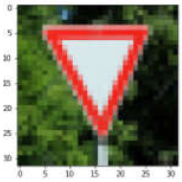
```
Predictions for Sample 2: No passing for vehicles over 3.5 metric tons

Prediction 1: No passing                              ==>     97.3221%
Prediction 2: No passing for vehicles over 3.5 metric tons ==>      1.2852%
Prediction 3: Keep right                              ==>      0.8252%
Prediction 4: Speed limit (80km/h)                    ==>      0.0911%
Prediction 5: Speed limit (30km/h)                    ==>      0.0888%

***Prediction is Incorrect***
```



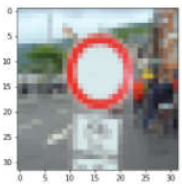
```
Predictions for Sample 4: Yield

Prediction 1: Yield                                   ==>    100.0000%
Prediction 2: Traffic signals                         ==>      0.0000%
Prediction 3: No entry                                ==>      0.0000%
Prediction 4: Speed limit (70km/h)                    ==>      0.0000%
Prediction 5: No passing                              ==>      0.0000%

***Prediction is Correct***
```



```
Predictions for Sample 5: No vehicles

Prediction 1: Yield                                   ==>     96.6302%
Prediction 2: No vehicles                             ==>      3.3691%
Prediction 3: No passing                              ==>      0.0006%
Prediction 4: Traffic signals                         ==>      0.0001%
Prediction 5: Speed limit (70km/h)                    ==>      0.0000%

***Prediction is Incorrect***
```



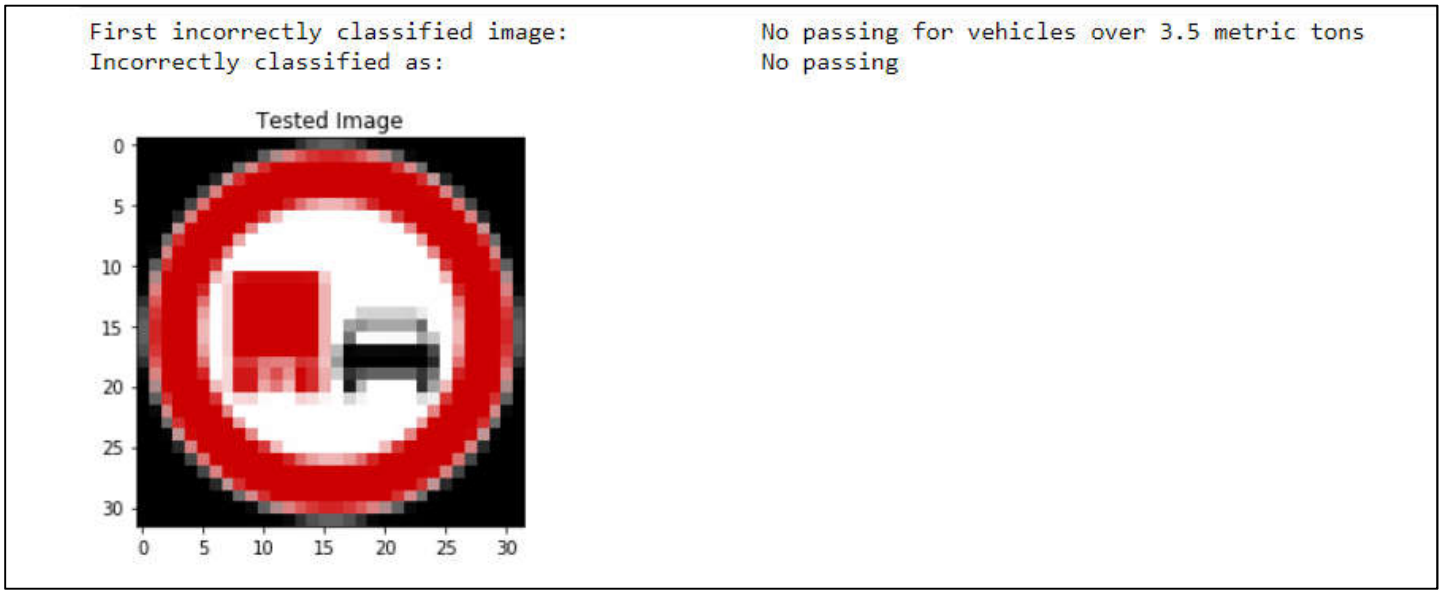
```
Predictions for Sample 4: Yield

Prediction 1: Yield                                   ==>    100.0000%
Prediction 2: Bumpy road                              ==>      0.0000%
Prediction 3: No vehicles                             ==>      0.0000%
Prediction 4: General caution                         ==>      0.0000%
Prediction 5: Traffic signals                         ==>      0.0000%

***Prediction is Correct***
```

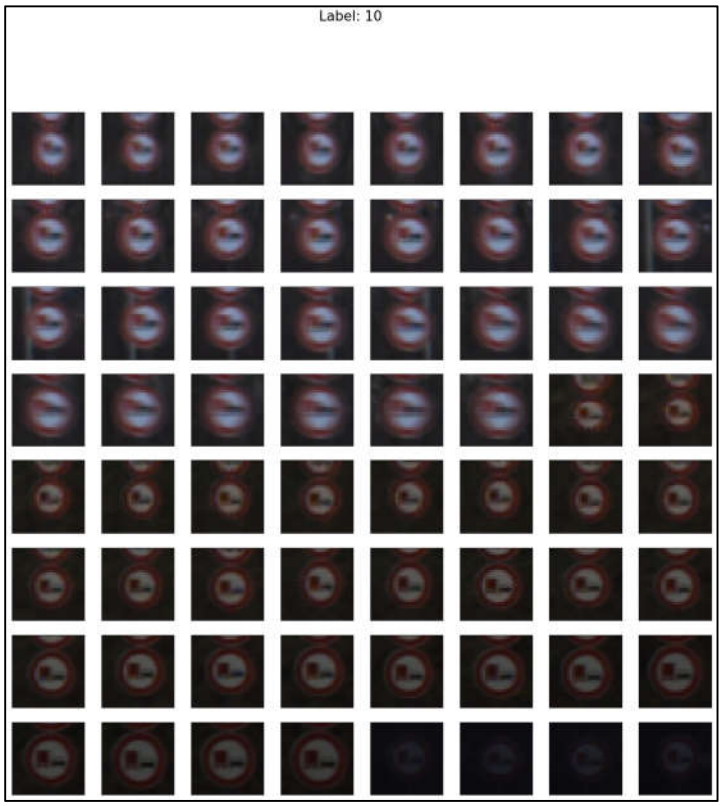


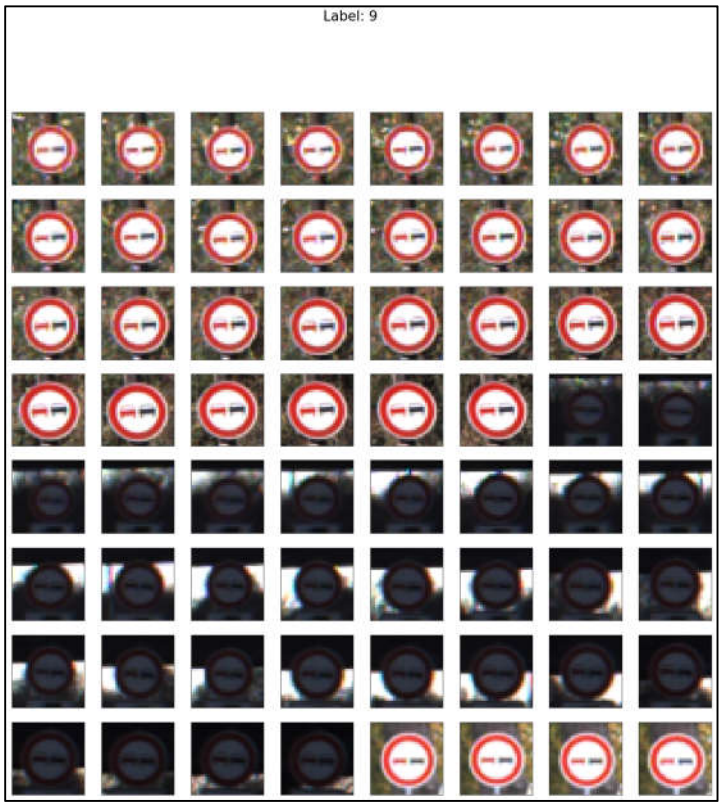
```
Predictions for Sample 5: No vehicles

Prediction 1: Yield                                   ==>     97.3284%
Prediction 2: Priority road                           ==>      1.6932%
Prediction 3: No passing                              ==>      0.4839%
Prediction 4: No vehicles                             ==>      0.1771%
Prediction 5: No passing for vehicles over 3.5 metric tons ==>      0.1340%

***Prediction is Incorrect***
```

Error Analysis – Misclassification #1

Considering the first of two misclassified images, the correct and incorrect prediction images look very similar. The only difference between the two is the red vehicle on the left is taller/larger in the 'No Passing: Vehicles over 3.5 tonnes' versus the normal 'No Passing' sign. It is easy to see how the models could have made this error. In both models, the correct prediction was second highest probability, with LeNet outperforming VGG16 in terms of certainty of correct class.







Error Analysis – Misclassification #2

In the second instance, the 'Yield' sign was misclassified as 'No Vehicles' with a high level of confidence in both cases. Based on the triangle/circle shape difference, it seems like it should be a very easy classification to discern. The only similarity is the coloration of each, but it doesn't seem like this should be sufficient to outweigh the shape difference. Maybe if the models were trained with grayscale images, they would have relied more heavily on edges versus colors, which appears to be the issue in this case. In both models the correct prediction was second highest probability, with LeNet outperforming VGG16 in terms of certainty of correct class.