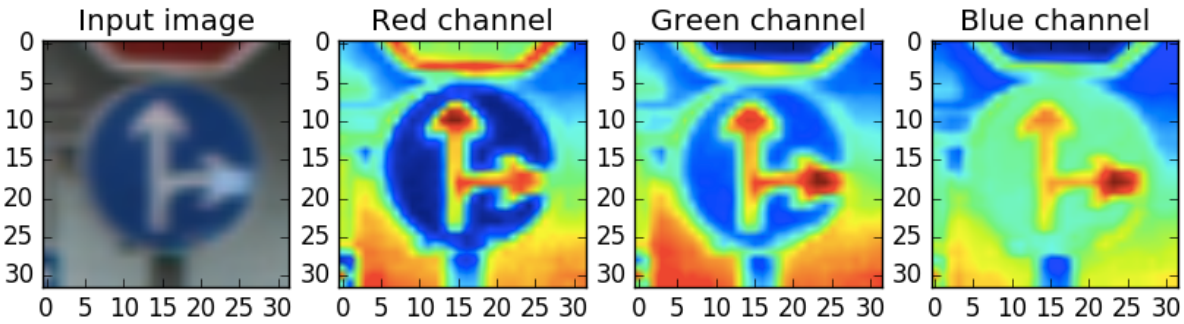
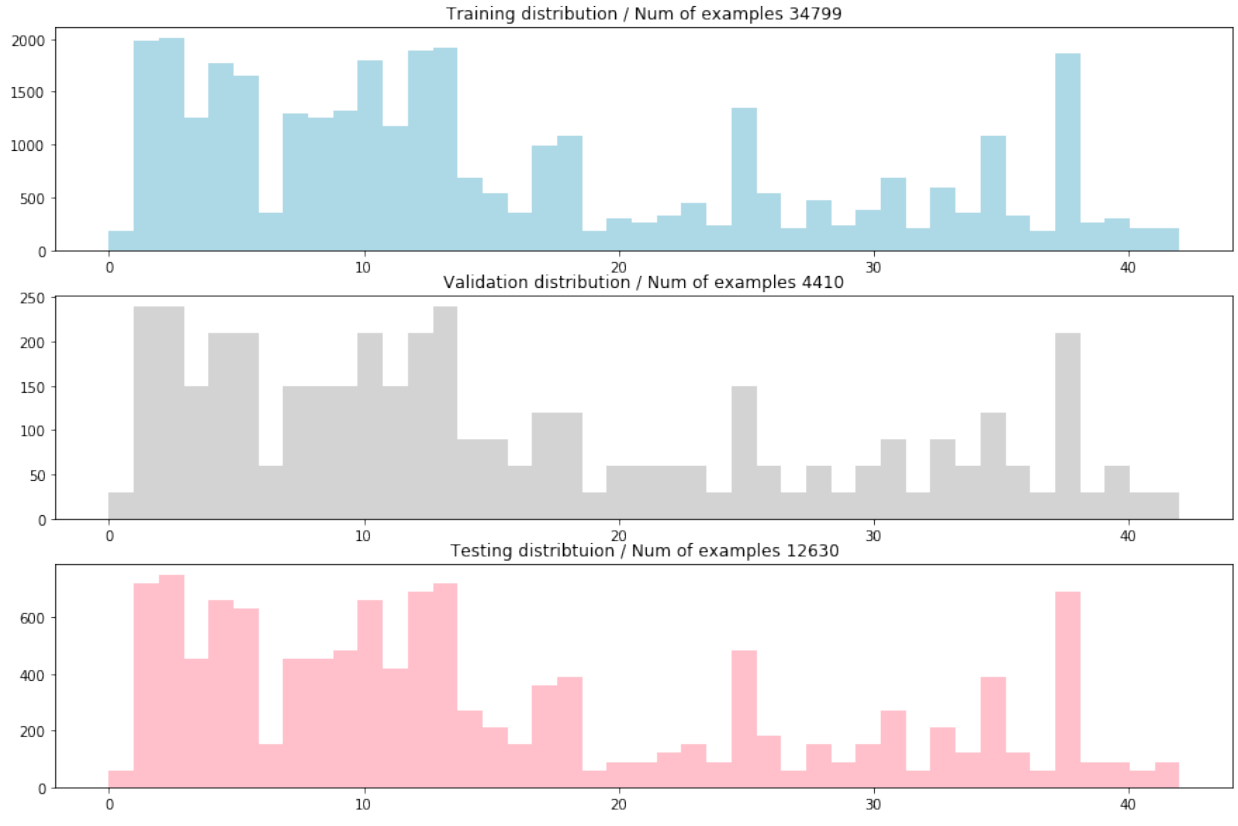
Traffic Sign Recognition Using Deep Learning

Data Exploration

In order to build our model, we need data. For the purpose of this project we are using images as our dataset and train a network purely on image data to classify traffic sign contained in the image. The type of machine learning problem that we are dealing with here is called multi-class classification. Our input is an image of size 32 x 32 x 3 (H x W x D) and output is a label representing the sign. The figure below shows an example image for a class 'Go straight or right' along with its 3 color channels (red, green, and blue).



The class labels are integer values which are mapped to specific signs. Our dataset contains 43 such unique classes (signs). For accurately testing and verifying our model accuracy, we have split our original data into 3 sets: training, validation, and testing. Each set is made up of (X, y) where X is set of images and y is corresponding set of labels. Training set contains 34799 examples, validation set contains 4410 examples, and the testing set contains 12630 examples. The figure below shows the distribution of labels among the three sets.

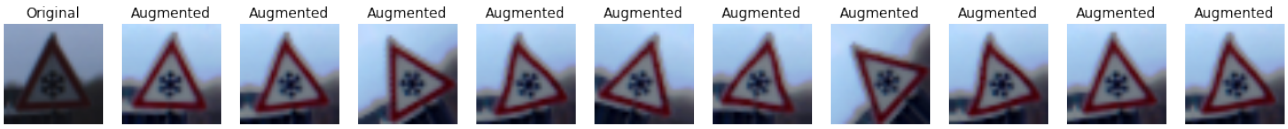


We can notice that each class is not equally distributed. For example, class with label 10 has more examples than class with label 0. This can sometimes lead our model to prefer one class over other. Also, all three sets have a similar distribution which can increase the performance without increasing the accuracy of the underlying model.

In order to visualize the signs and look at the variance for each class, I calculated the mean image for any given class. The figure below shows the mean image for each class/sign.

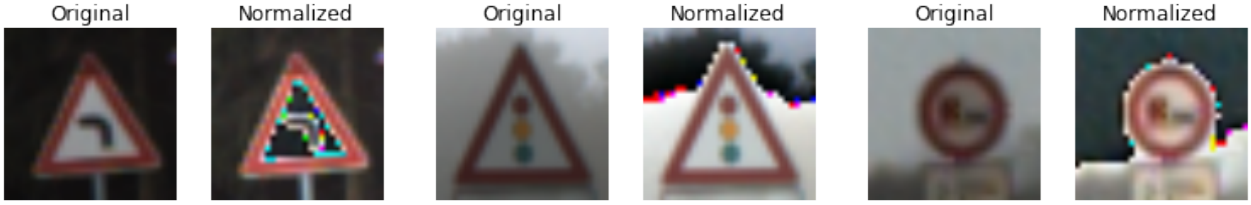
Augmentation and Preprocessing

Since the classes are not evenly distributed, my first approach was to generate fake data and generate until the number of examples for a given class reaches a threshold (I picked 1000). I used 3 image processing techniques to generate new images: rotate, swirl, and equalize histogram. For each image, the augment function would return 2 new images, one with random rotation and another with random swirl, and both followed by equalizing the histogram. Here is an example of newly generated image.



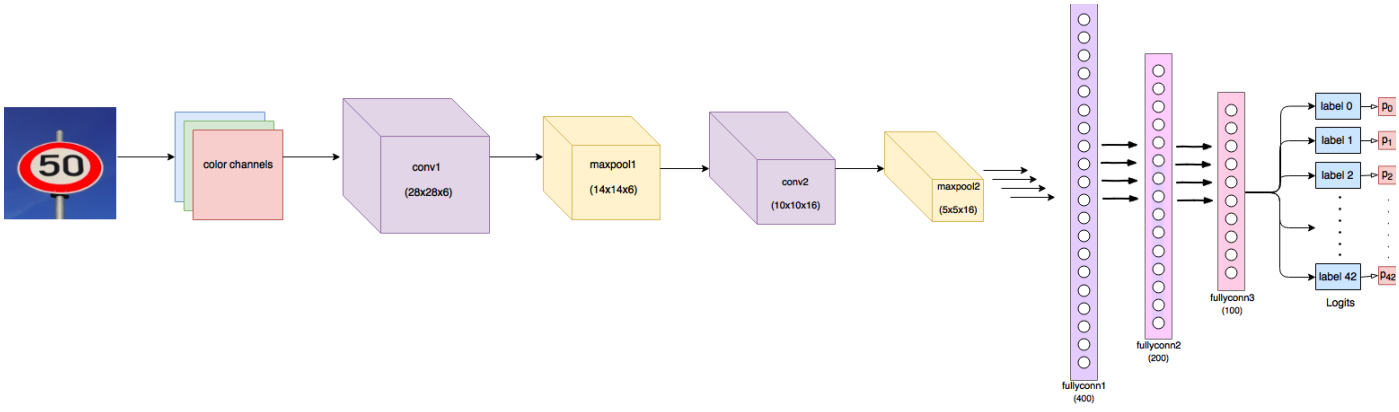
But surprisingly, adding more data to the training set did not improve the accuracy. I tested 2 different networks, described later, and both performed slightly worse with the augmented data compared to the original data. Also, I experimented with the ImageDataGenerator functionality provided by Keras to generate augmented images and reached same conclusion. The original dataset was able to outperform in both the network experiments. One reason for this negative performance gap is that for some of the signs the labels change when we apply some processing technique. For example, flipping the Turn Right sign along its vertical axis changes the sign to Turn Left which would produce incorrect training data.

Also, I normalized the dataset by subtracting 128 from image pixel data and then dividing by 128. This centers our data to have zero mean and equal variance which helps during optimization process. Here is an example of normalized image.



Model Architecture : LeNet

To start with, I used the LeNet architecture to observe how it performed on the dataset. I used the tensorflow implementation from one of the lab assignment and tweaked some of the parameters. Here is the standard architecture for LeNet.



After every maxpooling layer, I used ReLU activation function. To train the model, I choose Adam optimizer to learn the weights because it takes into consideration the momentum (moving average) of the gradient parameters when updating the parameters thereby increasing the step size. Here are the hyperparameters used to train the model after tweaking to find the best accuracy.

```
Optimizer: Adam Optimizer
Learning Rate: 0.005
Epochs: 50
Batch Size: 150
```

Unfortunately, the standard model failed to hit 0.93 accuracy on the validation set. The model performed well on the training set but was having trouble generalizing to the validation set thereby suggesting that the model is overfitting. To overcome this, I added dropout layers after each fully connected layer with probability 0.5. Also a recent technique called batch normalization has produced promising results in reducing overfitting. Batch normalization reduces *internal covariate shift* which refers to the change in the input distribution of our model. Here are the performance results for each dataset of the updated network with dropout and batch normalization layer.

```
Training accuracy: .998
Validation accuracy: .961
Testing accuracy: .942
```

Model Architecture : Modified Vgg

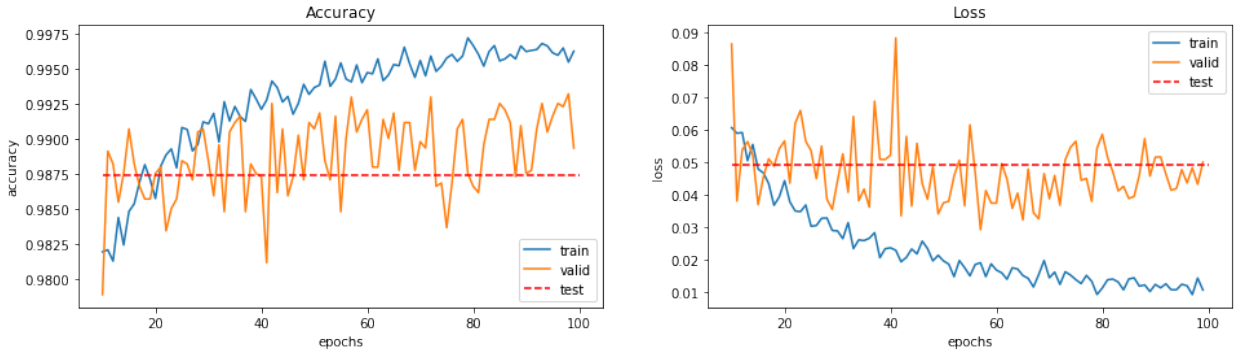
A lot of the modern networks consists of many more deep layers compared to LeNet and have achieved state of the art results. One such network was the Vgg (Visual geometry group) network which was one of the submission for 2014 ImageNet competition. The idea behind Vgg was to build simple but deep network consisting of convolutional layers followed by max-pool layers. The key idea was to use 3x3 sized filters rather than larger sized filters and reasoning is that combination of several small filters has similar results of using large receptive field (eg: 5x5, 11x11). Another key idea of this network is stacking convolutional networks back to back which also related to the idea of using small filters with increasing depth.

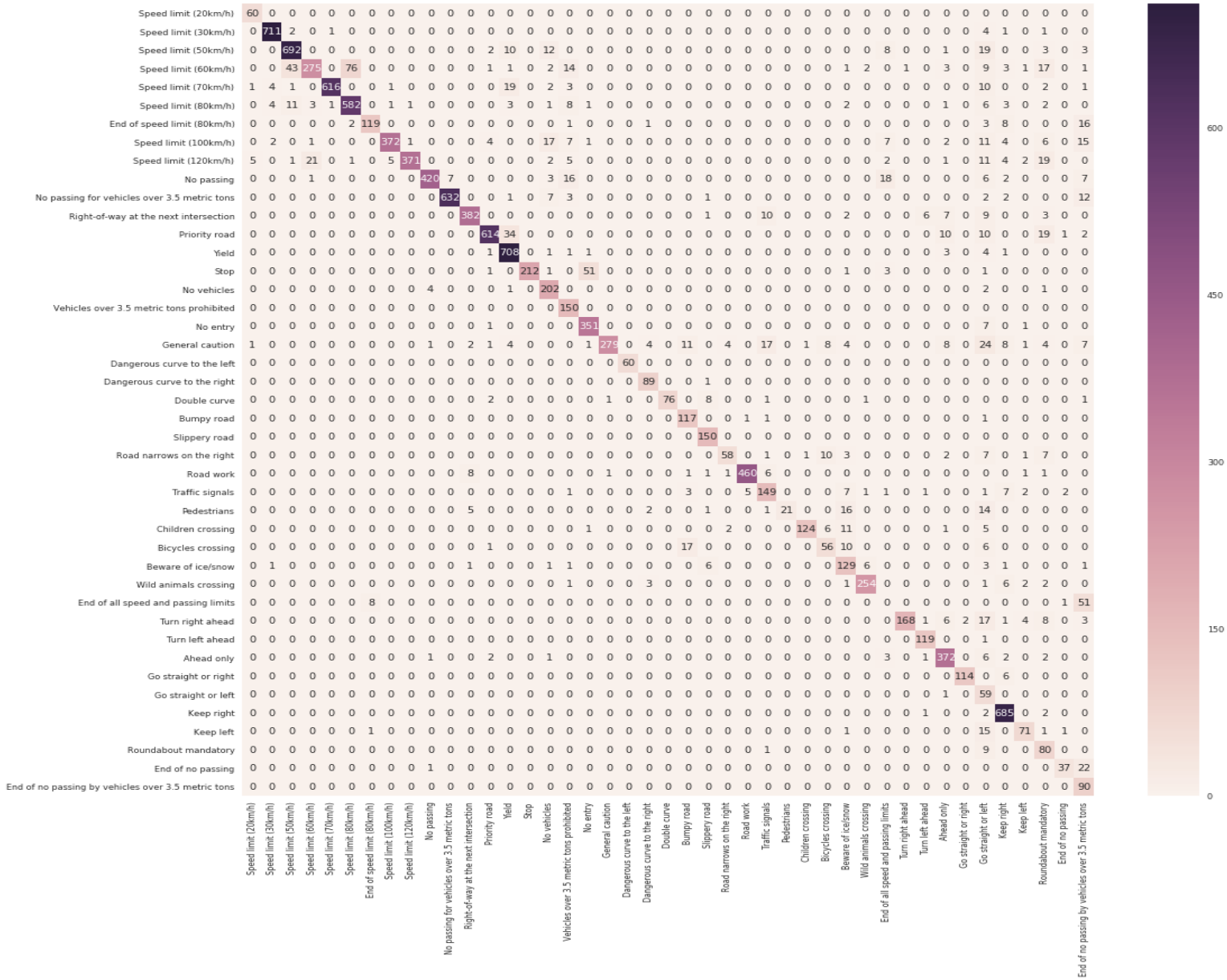
For the purpose of this assignment, I used Vgg's skeleton to create personalized Vgg network. Below is the architectural diagram of the network. I also used batch normalization layer along with dropout layers to reduce overfitting.

Layer (type)	Output Shape	Param #	Connected to
convolution2d_13 (Convolution2D)	(None, 32, 32, 32)	896	convolution2d_input_4[0][0]
activation_21 (Activation)	(None, 32, 32, 32)	0	convolution2d_13[0][0]
convolution2d_14 (Convolution2D)	(None, 30, 30, 32)	9248	activation_21[0][0]
activation_22 (Activation)	(None, 30, 30, 32)	0	convolution2d_14[0][0]
maxpooling2d_7 (MaxPooling2D)	(None, 15, 15, 32)	0	activation_22[0][0]
dropout_10 (Dropout)	(None, 15, 15, 32)	0	maxpooling2d_7[0][0]
convolution2d_15 (Convolution2D)	(None, 15, 15, 64)	18496	dropout_10[0][0]
activation_23 (Activation)	(None, 15, 15, 64)	0	convolution2d_15[0][0]
convolution2d_16 (Convolution2D)	(None, 13, 13, 64)	36928	activation_23[0][0]
activation_24 (Activation)	(None, 13, 13, 64)	0	convolution2d_16[0][0]
maxpooling2d_8 (MaxPooling2D)	(None, 6, 6, 64)	0	activation_24[0][0]
dropout_11 (Dropout)	(None, 6, 6, 64)	0	maxpooling2d_8[0][0]
flatten_4 (Flatten)	(None, 2304)	0	dropout_11[0][0]
dense_9 (Dense)	(None, 128)	295040	flatten_4[0][0]
activation_25 (Activation)	(None, 128)	0	dense_9[0][0]
batchnormalization_4 (BatchNorma	(None, 128)	512	activation_25[0][0]
dropout_12 (Dropout)	(None, 128)	0	batchnormalization_4[0][0]
dense_10 (Dense)	(None, 43)	5547	dropout_12[0][0]
activation_26 (Activation)	(None, 43)	0	dense_10[0][0]

VGG architecture

The code was implemented using Keras which is a wrapper library on top of Tensorflow. It provides a simple high level framework to build network architectures without the need to explicitly instantiate variables for inputs, parameters, outputs, etc. Here as well, I used the Adam optimizer with learning rate 0.001. Also for curiosity purpose, I trained the network for 100 epochs to see if there were any significant improvements. With the modified Vgg network, I was able to achieve 99% accuracy on validation set and 98.7% accuracy on testing set which is a significant improved over the LeNet. Figure below shows the training, validation, and testing accuracy and loss with each epoch





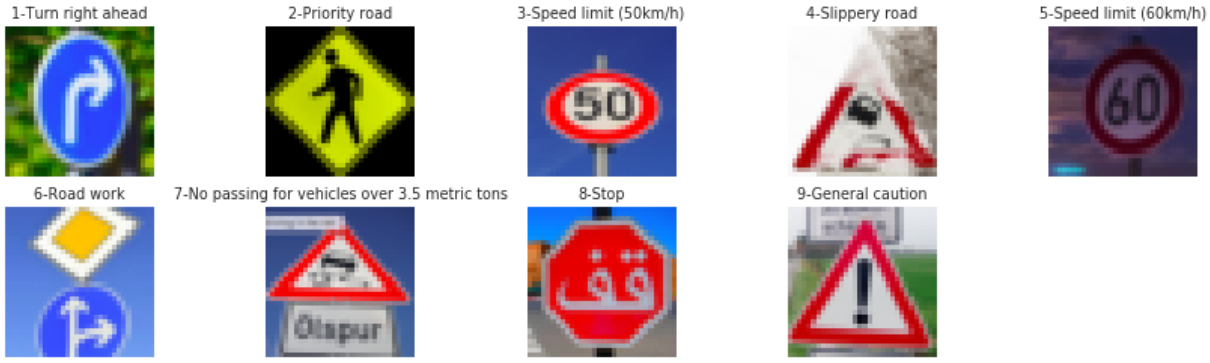
To understand how our model performed for each of the class and to visualize number of correct and incorrect examples per class, I calculated the confusion matrix. Figure above shows the confusion matrix. The y-axis represents the true class label and the x-axis represents the predicted output from our trained model. The class with the lowest classification accuracy was ‘End of all speed and passing limits’, Our model failed to classify any such images corrects and misclassified them as ‘End of no passing by vehicle over 3.5 meters’ since both signs are very much similar.

New Images

I collected the above image sample from internet to test our model on different set of examples. These images present different settings and will challenge our model. Here is a simple description of the images along with how difficult I think it would be for our model to classify them correctly.

image 1: similar to our training set but slightly tilted (class: turn right ahead) (easy)
image 2: new example not present in our dataset (class: pedestrians) (difficult)
image 3: similar to our training set (class: speed limit [50km/h]) (easy)
image 4: contains some noise which obscures the image (class: slippery road) (medium)
image 5: image with low contrast (class: speed limit [60km/h]) (medium)
image 6: contains 2 different signs (class: priority road, go straight or right) (difficult)
image 7: contains some noise (class: slippery road) (medium)
image 8: new example not present in our dataset (class: stop) (difficult)
image 9: similar to our training set (class: general caution) (easy)

Here are the results of what the model predicted



The model correctly predicted most of the images. But it misclassified three images: image 2, image 6 and image 7. For image 2, the correct label is "Pedestrians" but our model classified it as "Priority Road". This is because it has similar diamond shape structure and color which confuses our model. For image 6, the correct label would be either "Priority Road" or "Go straight or right" because it contains two signs but our model outputed "Road work". This is a tricky example for our model because our training examples contains only a single sign image.

Also, our model failed to output correct label for image 7. For image 4, our model was able to output correct label which suggests that our model was able to learn true representation for that specific class, and by looking at the confusion matrix, we can see that our model successfully classified all the test examples from that class. What is interesting is that the model successfully predict the "Stop Sign" despite being in a different language. The model relied on different features such as color and shape to predict the output rather than the actual text.

Total Examples = 9
Correct Examples = 6
Incorrect Examples = 3
Accuracy = 66.7%

To look into in depth result of the model predictions on the new images, I plotted the top 5 softmax scores which gives us more insight on how the model performs overall.