Traffic Sign Classification

Stephanie Buchanan

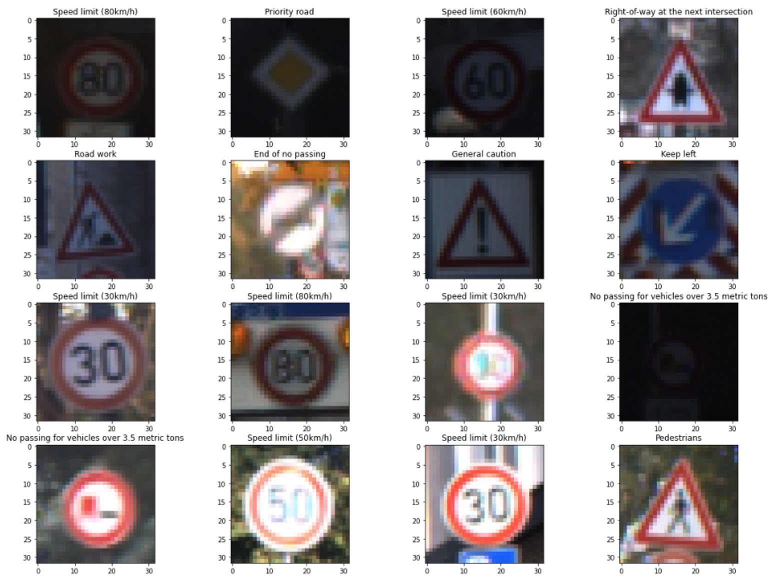
October 13, 2021

1 Executive Summary

The goal of this project was to build a traffic sign classifier using a convolution neural network, CNN. A CNN is a parametric model since it has a fixed number of parameters. The images provided simulate real driving conditions such as low light. The resulting classifier had an overall test accuracy of 97.8%. The classes with the highest F-score were speed limit, stop, priority road, no passing and and yield signs while the classes with the lowest F-score included traffic signals, roundabout mandatory, pedestrians, beware of ice/snow and double curve signs. Suggestions to improve the model are presented in the conclusions section.

2 Data

The data was provided and can be found on the Capstone GitHub site. The dataset was provided in three sets, the train, test and validate datasets. Figure 1 shows a subset of the training dataset along with the labels for the images.



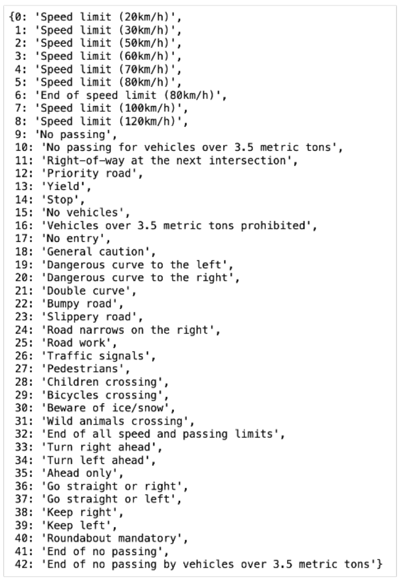

```
{0: 'Speed limit (20km/h)',
 1: 'Speed limit (30km/h)',
 2: 'Speed limit (50km/h)',
 3: 'Speed limit (60km/h)',
 4: 'Speed limit (70km/h)',
 5: 'Speed limit (80km/h)',
 6: 'End of speed limit (80km/h)',
 7: 'Speed limit (100km/h)',
 8: 'Speed limit (120km/h)',
 9: 'No passing',
 10: 'No passing for vehicles over 3.5 metric tons',
 11: 'Right-of-way at the next intersection',
 12: 'Priority road',
 13: 'Yield',
 14: 'Stop',
 15: 'No vehicles',
 16: 'Vehicles over 3.5 metric tons prohibited',
 17: 'No entry',
 18: 'General caution',
 19: 'Dangerous curve to the left',
 20: 'Dangerous curve to the right',
 21: 'Double curve',
 22: 'Bumpy road',
 23: 'Slippery road',
 24: 'Road narrows on the right',
 25: 'Road work',
 26: 'Traffic signals',
 27: 'Pedestrians',
 28: 'Children crossing',
 29: 'Bicycles crossing',
 30: 'Beware of ice/snow',
 31: 'Wild animals crossing',
 32: 'End of all speed and passing limits',
 33: 'Turn right ahead',
 34: 'Turn left ahead',
 35: 'Ahead only',
 36: 'Go straight or right',
 37: 'Go straight or left',
 38: 'Keep right',
 39: 'Keep left',
 40: 'Roundabout mandatory',
 41: 'End of no passing',
 42: 'End of no passing by vehicles over 3.5 metric tons'}
```

Figure 1: Left: Subset of traffic sign images from the training dataset. Right: Labels for the images.

The selected images show the different light conditions included in the dataset. There 43 different types of traffic signs, and a dictionary of the integer key and description value was included. The shape of the images in the dataset is (32, 32, 3) which corresponds to a width and height of 32 pixels and the depth is 3 for the red-green-blue intensity values and indicate that the images are in color.

The goal of the training dataset is for model building. Usually, it is the biggest proportion of the data in order to improve the model as much as possible. The training set can also be used to develop models with different hyperparameter settings. The validation dataset is used primarily for model selection, and to fine-tune parameters of the model. The model does not do any 'learning' with this dataset, but the results of the tuning does update the hyperparameters. The test dataset is used to get an unbiased evaluation of the final model fit on the training dataset. It is only used once the final model is selected to ensure the model is being evaluated on 'unseen' data.

To visualize the number of samples from each class in each of the split datasets, the plots shown in Figure 2 were generated. The numbers looked fairly uniform for each of the different classes across the three datasets.

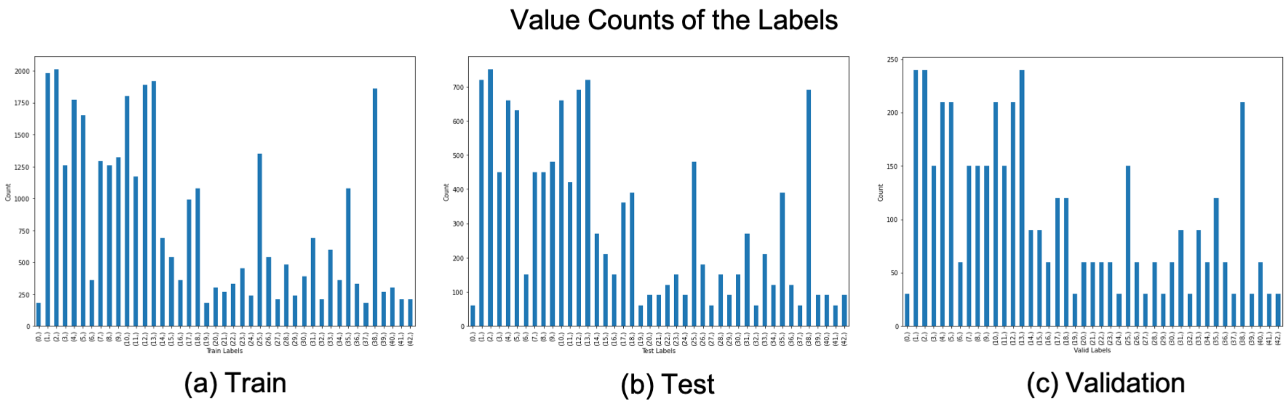


Figure 2: Value counts of each class from each of the train, validate and test datasets.

3 Research Question

Is it possible to build a model capable of determining the type of traffic sign that is displayed in an image captured under different real-life conditions and showing obstructions, poor lighting, or even the sign being far away from the camera?

4 Methods

A CNN is a network architecture for deep learning that learns directly from images. A typical neural network is made up of neurons or nodes in layers and weights or connections between all of the nodes. It is important to note that all attribute values of the data must be standardized resulting in values between 0 and 1 in a neural network. A typical neural network is usually fully connected, which means every node in a layer is connected to every node in the next layer.

The main way in which a CNN differs from a typical neural network is that, instead of being fully connected, only a small region of input layer neurons connects to the neurons in the hidden layer.[1] Figure 3 illustrates this difference and these regions are defined as local receptive fields.

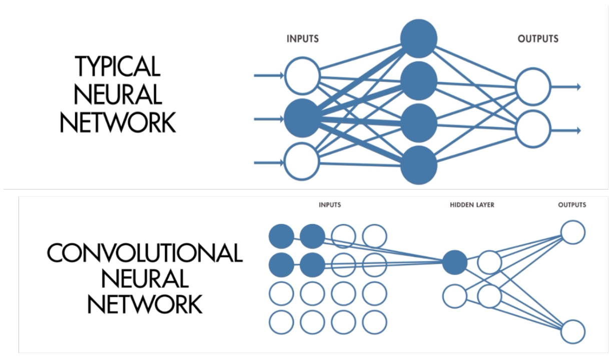


Figure 3: Typical vs. Convolutional Neural Networks.

There are three properties to the CNN architecture (a) local receptive fields and filters, (b) weights and biases, and (c) activation and pooling. Figure 4 shows a visualization of a local receptive field. A local receptive field is defined as an area, size 5x5 in the figure, of the input image that a particular convolutional layer is looking at. The output of convolutional layers is a feature map.

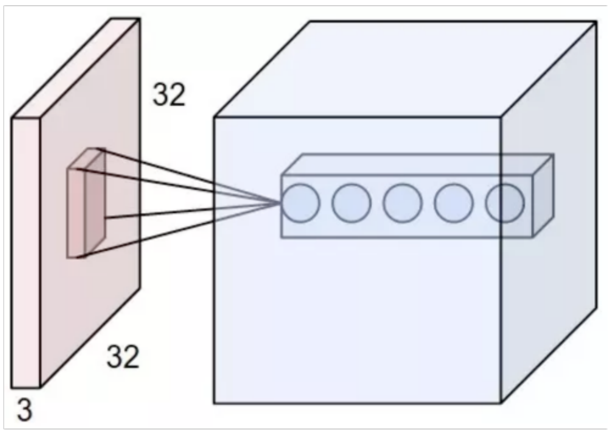


Figure 4: Illustration of local receptive fields in the input (red) and convolutional layer (blue).

The second property are the weights and biases. Figure 5 shows a neuron in a typical neural network.The weights are associated with each feature and bias is used for shifting the activation function left or right. In a CNN, the weights and bias vales are the same for all hidden neurons in a given convolutional layer.

[1] https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks–1489512765771.html

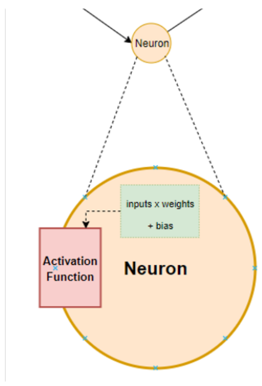


Figure 5: Neuron in a neural network.

The third property is activation and pooling and an illustration of pooling is shown in Figure 6. Activation and pooling transform the output of a neuron. Pooling reduces the dimensionality of the feature map by condensing the output of small regions of neurons into a single output. The pooling function typically applied is either the maximum or average value across the pooled area.

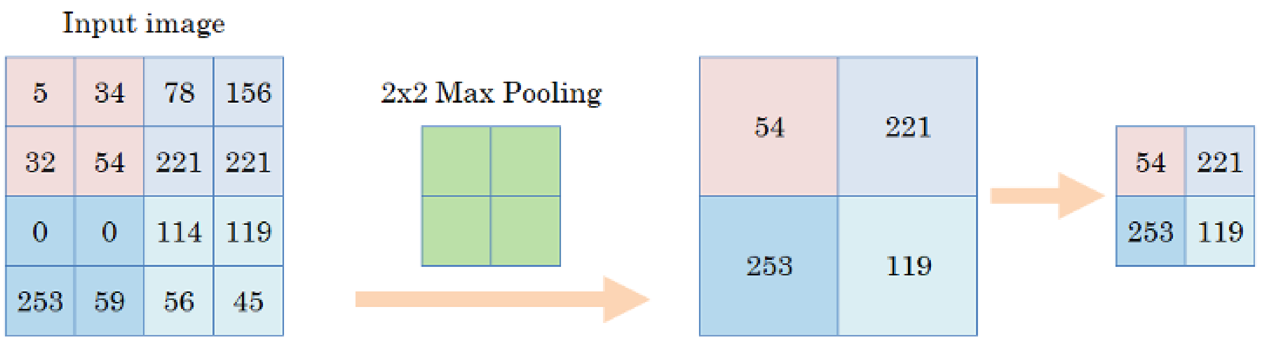
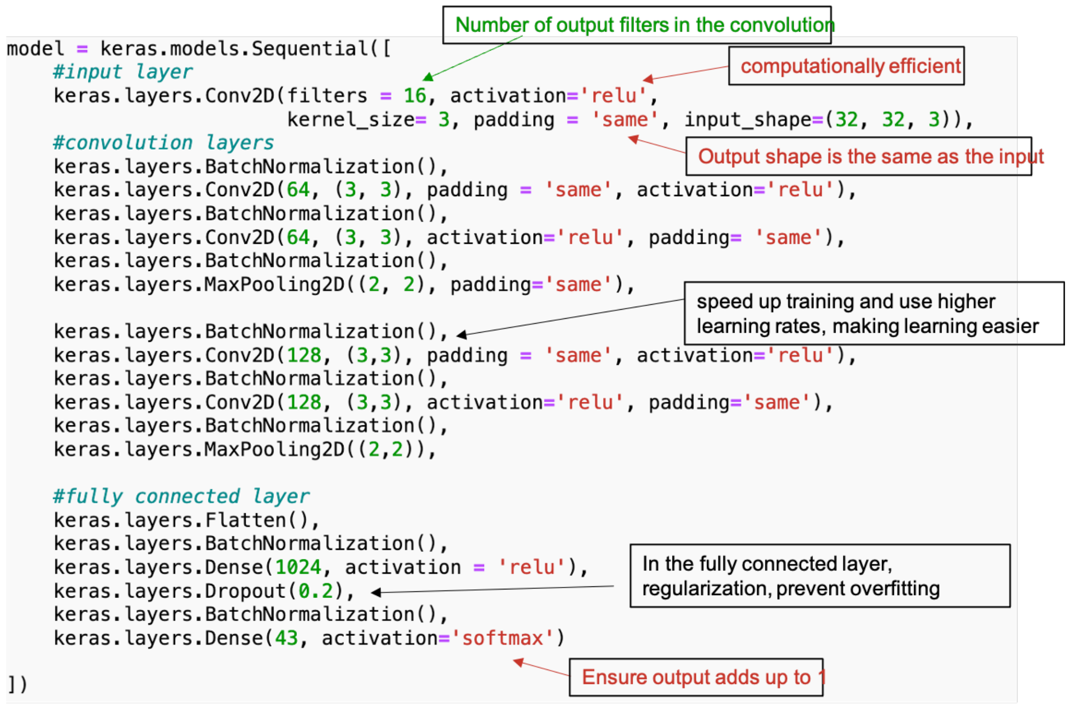


Figure 6: Illustration of pooling in a CNN.

5 Results

The architecture of the CNN used in this study is shown in Figure 7 and shows the input layer, hidden convolution layers and the final fully connected layer. The Rectified Linear Unit, ReLU, activation function is used in the convlutional layers to increase the non-linearity in the images. The final dense layer uses the softmax activation function to ensure that all the values in the output add up to 1.


Number of output filters in the convolution
computationally efficient
Output shape is the same as the input
speed up training and use higher learning rates, making learning easier
In the fully connected layer, regularization, prevent overfitting
Ensure output adds up to 1

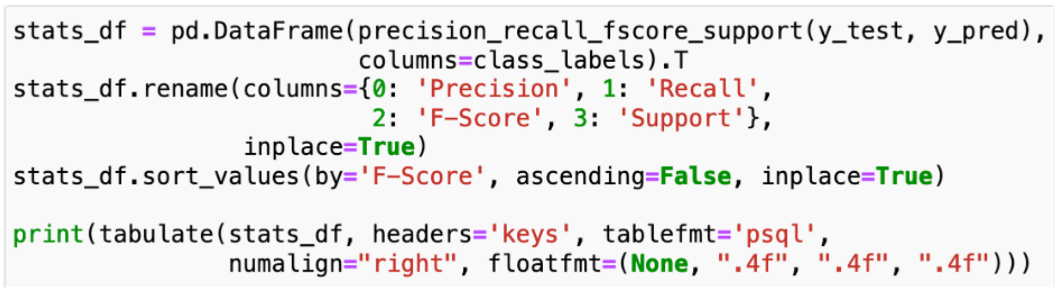
```
model = keras.models.Sequential([
    #input layer
    keras.layers.Conv2D(filters = 16, activation='relu',
                        kernel_size= 3, padding = 'same', input_shape=(32, 32, 3)),
    #convolution layers
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(64, (3, 3), padding = 'same', activation='relu'),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(64, (3, 3), activation='relu', padding= 'same'),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPooling2D((2, 2), padding='same'),

    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(128, (3,3), padding = 'same', activation='relu'),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(128, (3,3), activation='relu', padding='same'),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPooling2D((2,2)),

    #fully connected layer
    keras.layers.Flatten(),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(1024, activation = 'relu'),
    keras.layers.Dropout(0.2),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(43, activation='softmax')

])
```

Figure 7: Traffic sign classifier CNN architecture.

The 'kernal_size' or filter size was set to 3x3 and the padding was set to 'same' to ensure the output was the same shape as the input. Batch normalization was added in between the convolution and pooling layers to speed up training and use higher learning rates which usually makes the learning easily. A dropout of 0.2 was used in the final fully connected layer to prevent overfitting.

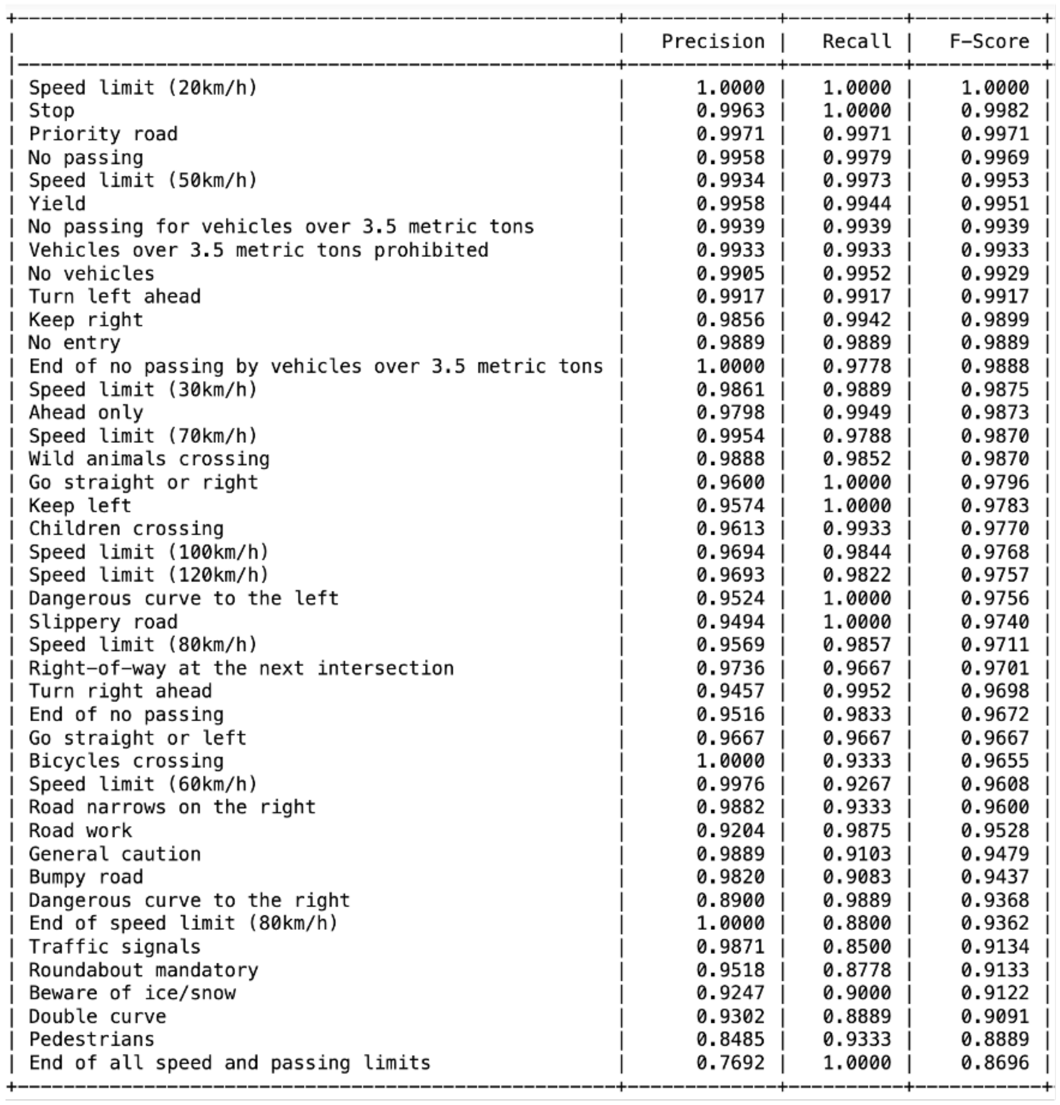
6 Conclusions

The best model had a train accuracy of 99.8%, validation accuracy of 98.7%, and an overall test accuracy was 97.8%. The precision, recall and F-score were calculated for each class and sort by the F-score in descending order. The code used to generate the table is shown in Figure 8, and the table in Figure 9.



```
stats_df = pd.DataFrame(precision_recall_fscore_support(y_test, y_pred),
                        columns=class_labels).T
stats_df.rename(columns={0: 'Precision', 1: 'Recall',
                         2: 'F-Score', 3: 'Support'},
                inplace=True)
stats_df.sort_values(by='F-Score', ascending=False, inplace=True)

print(tabulate(stats_df, headers='keys', tablefmt='psql',
               numalign="right", floatfmt=(None, ".4f", ".4f", ".4f")))
```

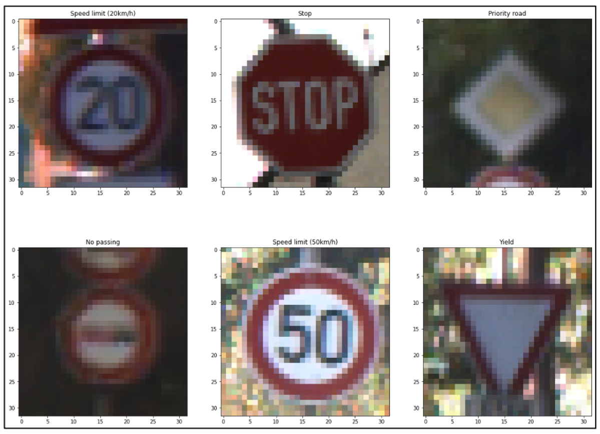
Figure 8: Python code to display the precision, recall and F-score for each class.



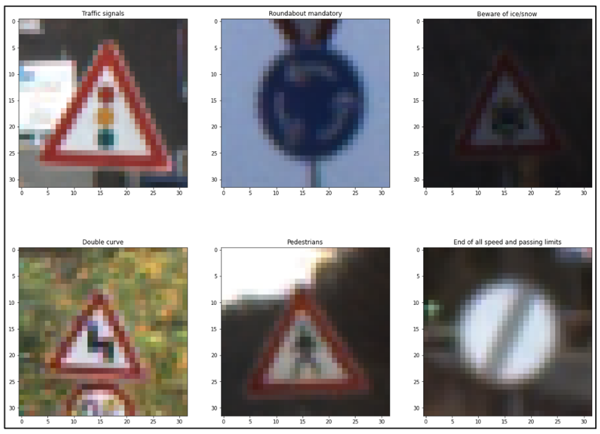
```
+----------------------------------------------------+-------------+----------+-----------+
|                                                    |  Precision  |  Recall  |  F-Score  |
|----------------------------------------------------+-------------+----------+-----------+
| Speed limit (20km/h)                               |      1.0000 |   1.0000 |    1.0000 |
| Stop                                               |      0.9963 |   1.0000 |    0.9982 |
| Priority road                                      |      0.9971 |   0.9971 |    0.9971 |
| No passing                                         |      0.9958 |   0.9979 |    0.9969 |
| Speed limit (50km/h)                               |      0.9934 |   0.9973 |    0.9953 |
| Yield                                              |      0.9958 |   0.9944 |    0.9951 |
| No passing for vehicles over 3.5 metric tons       |      0.9939 |   0.9939 |    0.9939 |
| Vehicles over 3.5 metric tons prohibited           |      0.9933 |   0.9933 |    0.9933 |
| No vehicles                                        |      0.9905 |   0.9952 |    0.9929 |
| Turn left ahead                                    |      0.9917 |   0.9917 |    0.9917 |
| Keep right                                         |      0.9856 |   0.9942 |    0.9899 |
| No entry                                           |      0.9889 |   0.9889 |    0.9889 |
| End of no passing by vehicles over 3.5 metric tons |      1.0000 |   0.9778 |    0.9888 |
| Speed limit (30km/h)                               |      0.9861 |   0.9889 |    0.9875 |
| Ahead only                                         |      0.9798 |   0.9949 |    0.9873 |
| Speed limit (70km/h)                               |      0.9954 |   0.9788 |    0.9870 |
| Wild animals crossing                              |      0.9888 |   0.9852 |    0.9870 |
| Go straight or right                               |      0.9600 |   1.0000 |    0.9796 |
| Keep left                                          |      0.9574 |   1.0000 |    0.9783 |
| Children crossing                                  |      0.9613 |   0.9933 |    0.9770 |
| Speed limit (100km/h)                              |      0.9694 |   0.9844 |    0.9768 |
| Speed limit (120km/h)                              |      0.9693 |   0.9822 |    0.9757 |
| Dangerous curve to the left                        |      0.9524 |   1.0000 |    0.9756 |
| Slippery road                                      |      0.9494 |   1.0000 |    0.9740 |
| Speed limit (80km/h)                               |      0.9569 |   0.9857 |    0.9711 |
| Right-of-way at the next intersection              |      0.9736 |   0.9667 |    0.9701 |
| Turn right ahead                                   |      0.9457 |   0.9952 |    0.9698 |
| End of no passing                                  |      0.9516 |   0.9833 |    0.9672 |
| Go straight or left                                |      0.9667 |   0.9667 |    0.9667 |
| Bicycles crossing                                  |      1.0000 |   0.9333 |    0.9655 |
| Speed limit (60km/h)                               |      0.9976 |   0.9267 |    0.9608 |
| Road narrows on the right                          |      0.9882 |   0.9333 |    0.9600 |
| Road work                                          |      0.9204 |   0.9875 |    0.9528 |
| General caution                                    |      0.9889 |   0.9103 |    0.9479 |
| Bumpy road                                         |      0.9820 |   0.9083 |    0.9437 |
| Dangerous curve to the right                       |      0.8900 |   0.9889 |    0.9368 |
| End of speed limit (80km/h)                        |      1.0000 |   0.8800 |    0.9362 |
| Traffic signals                                    |      0.9871 |   0.8500 |    0.9134 |
| Roundabout mandatory                               |      0.9518 |   0.8778 |    0.9133 |
| Beware of ice/snow                                 |      0.9247 |   0.9000 |    0.9122 |
| Double curve                                       |      0.9302 |   0.8889 |    0.9091 |
| Pedestrians                                        |      0.8485 |   0.9333 |    0.8889 |
| End of all speed and passing limits                |      0.7692 |   1.0000 |    0.8696 |
+----------------------------------------------------+-------------+----------+-----------+
```

Figure 9: Precision, recall and F-score for each traffic sign class.

The 6 classes with the higher F-score and the 6 classes with the lowest F-score are shown in Figure 10. As seen in the images, the classes with the highest F-scores contained numbers or simple graphics like a line on the no entry sign or letters like on the stop sign. The classes with low F-scores had more complex graphics like a mandatory roundabout, or pedestrian, or double curve. One way to improve the accuracy for these classes could be to include more augmented images for these specific classes in the training set. Hyper-parameter optimization was not done in this study due to the time investment required, but could be evaluated in the future. Some options for hyper-parameter tuning are (a) Random Search, (b) HyperBand and/or (c) Bayesian Optimization.



(a)



(b)

Figure 10: Representative images for the classes with (a) high F-score and (b) low F-score.