Self-Driving Car Nanodegree

P2 – Build a Traffic Sign Recognition Classifier

Raymond Ngiam
October 07th, 2017

I. Definition

Project Overview

With the advent of self driving car technology, the need for reliable and robust object recognition systems becomes very crucial and indispensable. One subclass of the object recognition system of a self driving car is the recognition of traffic signs. Correct recognition of road traffic signs during driving is essential and serves as an input for autonomous decision making of a self driving car. This consequently affects the safety and effectiveness of the self driving car technology.

Problem Statement

The problem that this project intended to solve is the autonomous recognition of road traffic signs.

The goal of this project is to create a classifier that is capable of recognizing various traffic signs on the road reliably and robustly. The tasks involved in this project include the following:

1. Develop an image preprocessing pipeline for the input data.
2. Train a classifier for recognizing traffic signs.
3. Make the classifier's output a probabilistic measure that quantifies the confidence level of classifier toward the prediction.

Metrics

The metrics to be used for performance evaluation of this project are the accuracy, precision, recall and F_{beta} score.

The definition of accuracy, precision and recall in context are as depicted below:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$where\ \ TP\ = number\ of\ true\ positive\ cases$

$TN = number\ of\ true\ negative\ cases$

$FP = number\ of\ false\ positive\ cases$

$FN = number\ of\ false\ negative\ cases$

Accuracy measures the ratio of correct predictions to the total number of cases evaluated. It may seem obvious that the ratio of correct predictions to cases should be a key metric. However, a model may have high accuracy, but have low predictive power. This is what known as accuracy paradox.

Hence, in addition to the accuracy metric, two additional metrics, namely precision and recall are used. Precision measures the ratio of positive predictions to the total number of positive class values predicted. Precision can be thought of as a measure of a classifier’s exactness. A low precision indicates a large number of False Positives.

Recall measures the ratio of positive predictions to the number of positive class values in the class labels. Recall can be thought of as a measure of a classifier’s completeness. A low recall indicates many False Negatives.

F_{beta} score is a measure that combines precision and recall into one single metric, and is defined as follow:

$$F_{beta}\ = (1 + beta^2) \cdot \frac{Precision \cdot Recall}{beta^2 \cdot Precision + Recall}$$

$where\ \ beta\ = hyperparameter\ for\ trade\text{-}off\ between\ Precision\ and\ Recall\ (beta > 0)$

The three commonly used F_{beta} measures are the F_1 measure, which weights precision and recall equally, the F_2 measure, which weights recall higher than precision, and the $F_{0.5}$ measure, which puts more emphasis on precision than recall.

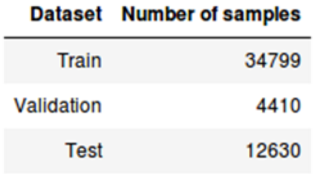
In the context of this problem, the implications of false positives and false negatives have equal weights, i.e. falsely classified a traffic sign and failed to recognize a particular traffic sign, do not have larger implication over one another. Hence, F1 score measure will be used.

II. Analysis

Dataset Exploration

The dataset to be used is the German Traffic Sign Recognition Benchmark (GTSRB) dataset. GTSRB is a multi-class, single-image classification challenge held at the International Joint Conference on Neural Networks (IJCNN) 2011. The dataset consists of 43 classes and more than 50,000 images in total.

The dataset is divided into 3 portions, i.e. training, validation and test sets. Each sample in the datasets is a 32x32 color image of a particular German street sign. Table 1 summarizes the number of samples in the three datasets.



Dataset	Number of samples
Train	34799
Validation	4410
Test	12630

Table 1 – Number of samples in the three datasets.

Figure 1 shows one of the images for each label in the training set.

Exploratory Visualization

Let’s explore the distribution of the class labels in the training set. The histograms of class labels for all three datasets are visualized in Figure 2.

From Figure 2, it is observed that the distributions of class labels in all three datasets are not uniformly distributed.

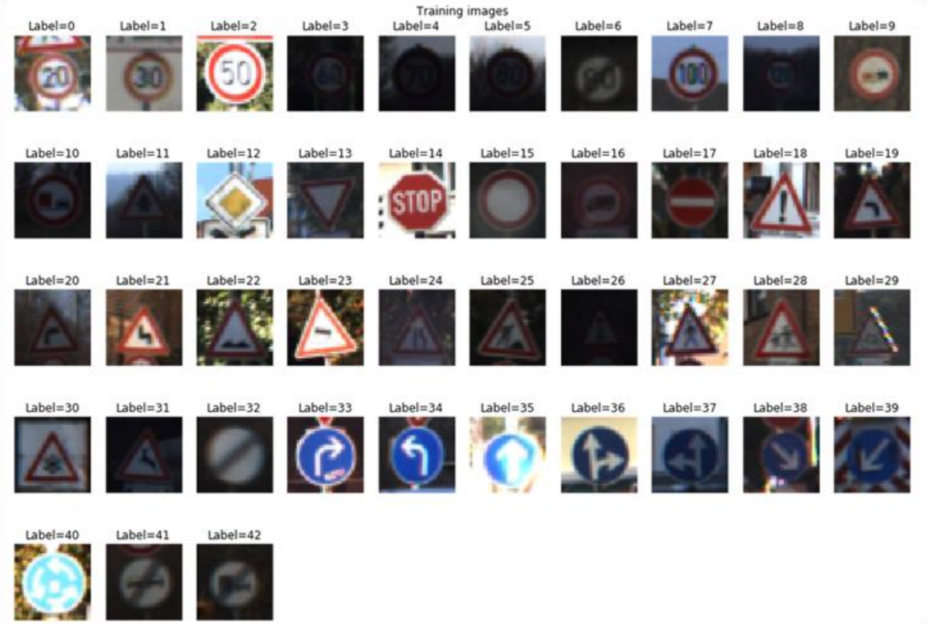
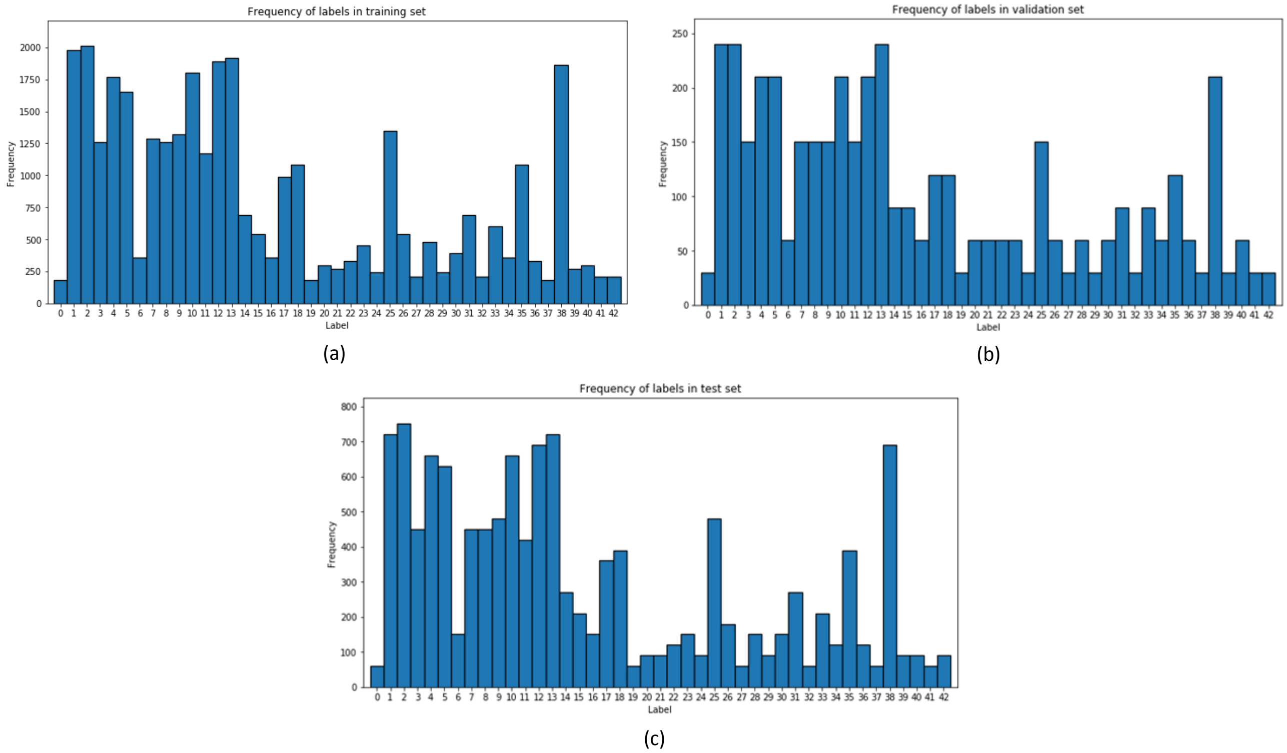


Figure 1 – One image per class label in the training set.



(a)

(b)

(c)

Figure 2 – (a) Histogram of class labels in training set, (b) Histogram of class labels in validation set, (c) Histogram of class labels in test set.

Algorithms and Techniques

The algorithm to be used in tackling this problem is the Convolutional Neural Network (CNN), which is the current state-of-the-art method for image classification.

CNN uses deep layers of interconnected neurons, known as convolutional layer to perform image convolution operations. A convolutional layer typically has many depth slices, whereby each depth slice represents on feature computed by convolution with the input from previous layer.

The number of parameters used to parameterize a convolutional layer is dependent of the size of convolution mask, depth of input channel, and depth of output channel, e.g. a 3x3 mask for a 3 layer input depth and 1 layer output depth has 10 parameters (3x3x3x1=9 weights and 1 bias).

For the same convolutional layer as above, with stride 1 and 'valid' padding, an input image of size 32x32x3 will produce an output is 30x30x1. If the same configuration is represented with a fully connected layer, the number of parameters required to parameterize the layer is over 2.76 millions.

Convolutional layers exploit an assumption that useful features in an image are translation invariant. In other words, if one feature is useful to be computed at some spatial positions, then it should also be useful to be computed at different spatial positions. By sharing parameters of a convolution mask and translating across the entire image, it serves to generate a map for a particular image feature, as well as significantly reduces the number of parameters relative to a fully connected layer. This makes training of deep convolutional layers computationally feasible in real time.

Benchmarks

The benchmark model to be used is the model by Sermanet, LeCun (2011) which reported test set accuracy of 99.17%.

Sermanet, LeCun (2011) also mentioned that the human performance on the GTSRB test dataset is 98.81%.

III. Methodology

Data Preprocessing

The images in the GTSRB datasets are color images. Although color plays a certain role in the visual perception of traffic signs, however it is not the most crucial or indispensable feature. Geometric shape and visual content within the sign are relatively more important and can be interpreted from gray images. Hence to simplify the processing and ensure only the more important features are taken into account, the color images will be convert into gray images with the following transformation:

$$G(x, y) = 0.299\, R(x, y) + 0.587\, G(x, y) + 0.114\, B(x, y)$$

The above transformation weights the color components similar to human visual perception, which gives higher weight for the green component. The output image is a gray image as shown in the Figure 3 below:

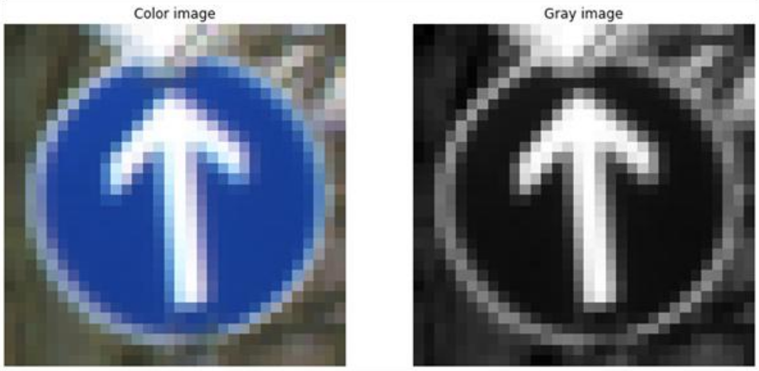


Figure 3 – Conversion of color image to gray image.

As mentioned in the Data Exploration section, the distributions of class labels among the three datasets are not uniformly distributed. This may cause erroneous interpretation of the accuracy due to class labels are not equally weighted.

Furthermore, the unevenly weighted class labels in the training set may also hinder the training of the neural networks if conventional loss function is used where all training samples are weighted equally.

Hence, to solve the problem of imbalanced class labels in the datasets, data augmentation will be performed by generating variations of existing images where sample size is less. A framework for generating variations in translation and rotation of the existing images is introduced.

For generating variations in translation, the images are offset by two pixels relative to the image center in one of the eight directions (NW, N, NE, E, SE, S, SW, and W). See Figure 4 for visualization of the translation in action.

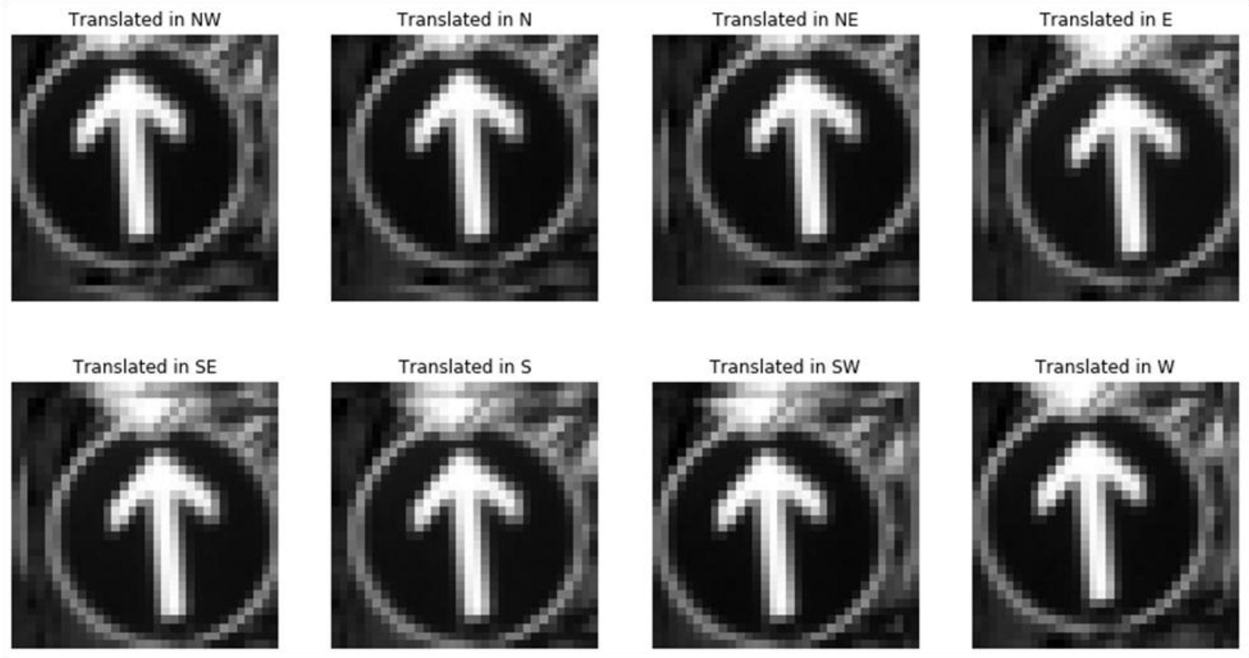


Figure 4 – Translational variations in eight directions.

For generating variations in rotation, the images are rotated in either 8° or -8°. Figure 5 shows the rotational variations of an original gray image.

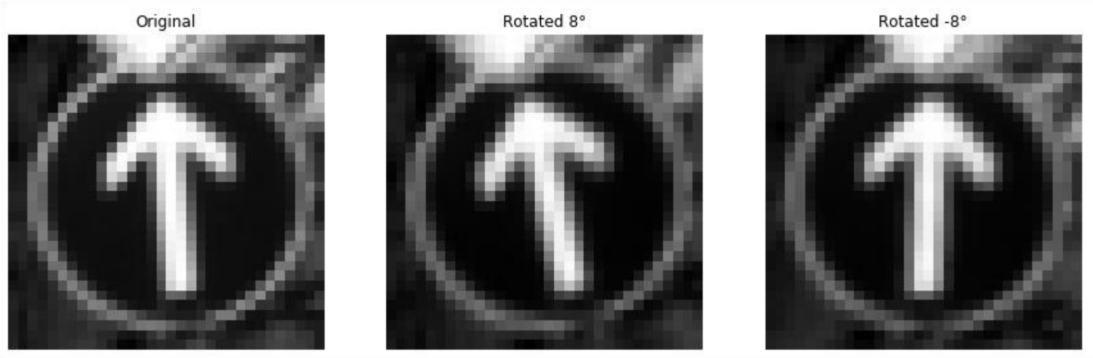


Figure 5 – Rotational variations with angles 8° and -8°.

Finally, the augmented gray image is normalized by subtracting the gray values with the mean gray values and then divides by the standard deviation of the gray values. Thus, the normalized image will have zero mean and standard deviation of 1. Figure 6 shows the histogram of a gray image before and after normalization.

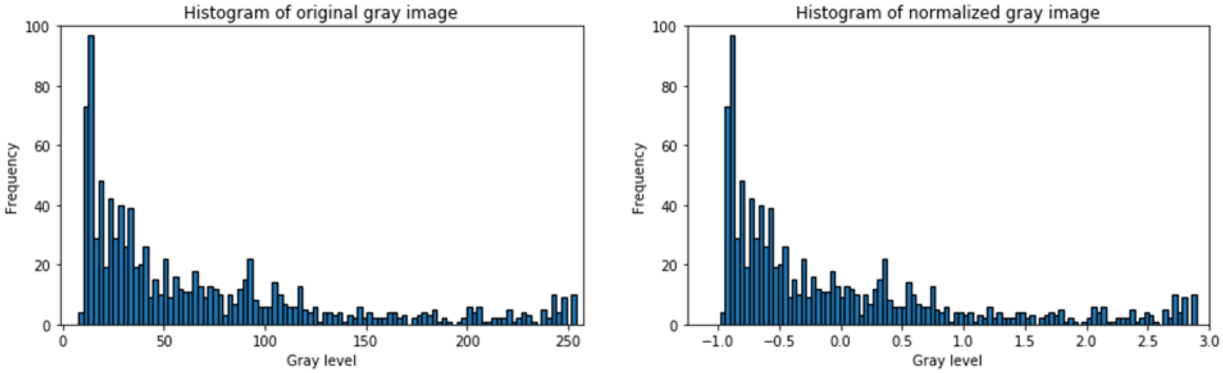


Figure 6 – Gray value normalization of an image.

For the training of neural networks, normalized training input with zero mean and equal variance would make the optimization problem well conditioned, thus improve the optimization performance.

With the data preprocessing pipeline in place, the train set is preprocessed such that each class label has the same number of samples as the mode class label. This is achieved by random sampling images within each class label and applies one of the random combinations between translations (8 variations) and rotations (2 variations).

Figure 7 shows class label histogram of the augmented training set.

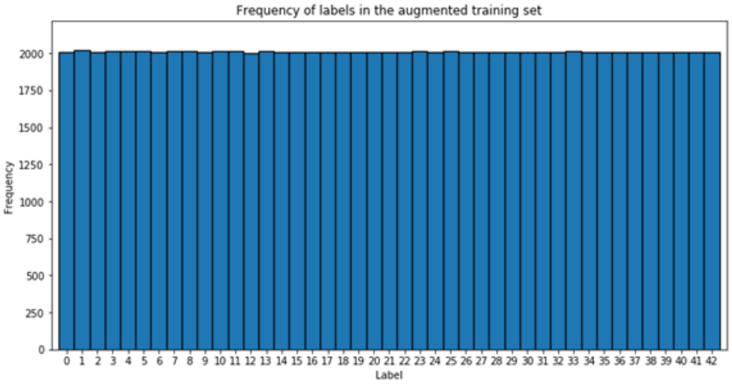
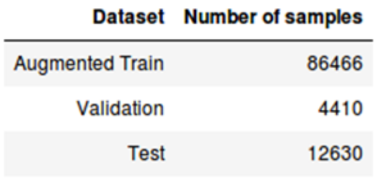


Figure 7 – Histogram of class labels in the augmented training set.

For the validation and test set, the preprocessing is only limited to gray image conversion and normalization. Table 2 shows the number of samples for augmented train set, validation set and test set.



Dataset	Number of samples
Augmented Train	86466
Validation	4410
Test	12630

Table 2 – Number of samples in the augmented train, validation and test set.

Model Architecture

The model essentially consists of 4 convolutional layers and 2 fully connected layers. Full configuration of the model is as depicted below:

```
INPUT: [32x32x1]
LAYER 1:  INPUT    -> CONV-3-1-32  -> RELU    -> BATCHNORM -> DROPOUT                     [30x30x32]
LAYER 2:  LAYER 1  -> CONV-3-1-32  -> RELU    -> BATCHNORM -> MAXPOOL-2-2 -> DROPOUT      [14x14x32]
LAYER 3:  LAYER 2  -> CONV-3-1-64  -> RELU    -> BATCHNORM -> DROPOUT                     [12x12x64]
LAYER 4:  LAYER 3  -> CONV-3-1-64  -> RELU    -> BATCHNORM -> MAXPOOL-2-2 -> DROPOUT      [5x5x64]
LAYER 5:  LAYER 4  -> FLATTEN                                                             [1600]
LAYER 6:  LAYER 5  -> FC-1000      -> RELU    -> BATCHNORM -> DROPOUT                     [1000]
LAYER 7:  LAYER 6  -> FC-1000      -> RELU    -> BATCHNORM -> DROPOUT                     [1000]
LAYER 8:  LAYER 7  -> FC-43        -> SOFTMAX                                             [43]

with:
          CONV-[patch size]-[stride]-[depth]
          MAXPOOL-[patch size]-[stride]
          FC-[output node]
```

In general, each convolutional layer starts with a convolution operation with stride 1 and 'valid' padding, followed by a Rectified Linear Unit (ReLU) activation. Batch normalization [5] is then applied to the activations.

The purpose of batch normalization is to ensure the outputs to all subsequent layers are well centered around zero with equal variance. It serves to make the optimization problem well conditioned and thus allows the network to learn at much higher learning rates.

Batch normalization usually has two trainable parameters which are scale and offset. Since we are using only ReLU activation function, the batch normalization will only has one trainable parameter, i.e. the offset, since scaling operation does not affect the output distribution of ReLU activations.

Max pooling with patch 2x2 and stride 2 is carried out at every even number convolutional layer, i.e. layer 2 and 4.

Dropout is applied at the end of every convolutional layer for regularization of the model.

The output from the last convolutional layer is flatten at layer 5, and then fed into two fully connected layers, i.e. layer 6 and 7. The two fully connected layers have 1000 nodes with ReLU activation. Batch normalization and dropout are also applied to these two layers.

Finally, the output of layer 7 is connected to layer 8, a fully connected layer of 43 nodes with softmax activation.

Graph visualization of the model in Tensorboard is as shown in Figure 8

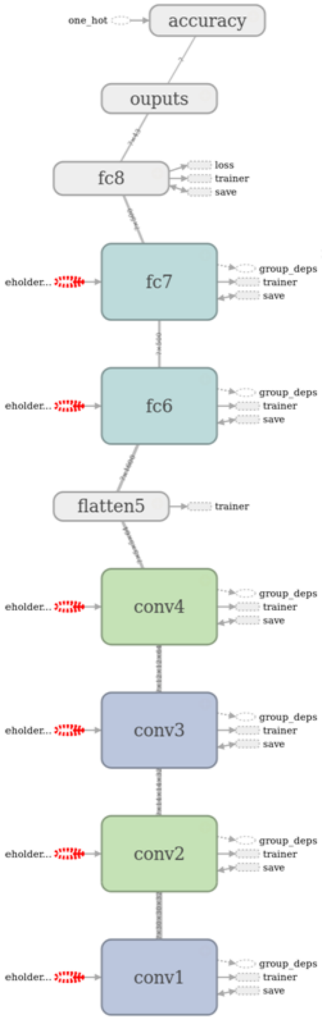


Figure 8 – Graph visualization of the model in Tensorboard.

Model Training

The loss functions used for the training of the model is the average cross entropy of the predicted classes, defined as follows:

$$loss = \frac{1}{N}\sum_{i=1}^{N} -\log P(prediction_i == label_i)$$

$where\ \ P(prediction_i == label_i) = softmax\ probability\ of\ the\ correct\ prediction$

$N = number\ of\ training\ samples$

The model is trained with an Adaptive Moment Estimation (Adam) Optimizer [7]. Adam optimizer computes adaptive learning rates for each parameter. It stores exponentially decaying averages of past gradients m_t, and past squared gradients v_t.

m_t and v_t are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively. They are initialized as vectors of 0's during the start of the training, and are updated for each parameter as follows:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) {g_t}^2$$

$where\ \ g_t\ = gradient\ of\ the\ parameter\ at\ step\text{-}t$

$\beta_1 = decay\ parameter\ for\ gradient$

$\beta_2 = decay\ parameter\ for\ squared\ gradient$

m_t and v_t are biased towards zero, especially during the initial time steps, and especially when the decay rates are small (i.e. β_1 and β_1 are close to 1). Bias-corrected first and second moment estimates can be determined as follows:

$$\widehat{m_t} = \frac{m_t}{1 - {\beta_1}^t}$$

$$\widehat{v_t} = \frac{v_t}{1 - {\beta_2}^t}$$

Update rule for the parameters during training is then defined as:

$$\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{\widehat{v_t}} + \varepsilon}\widehat{m_t}$$

where α *= learning rate*

ε *= a small constant for numerical stability, i.e. to prevent division with zero.*

Due to large size of the training set, the training of the model is implemented in mini-batches. The choice of batch size used is 128 and the training of the model will be executed for 40 epochs.

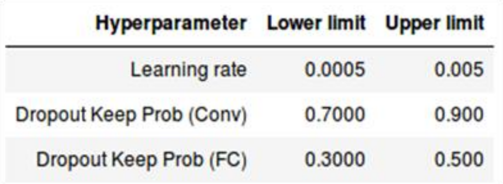
After each epoch, the model's performance metric, i.e. accuracy, on the validation set is evaluated. If the validation accuracy is improved in a new epoch, the new model will overwrite the previous one. Hence only the model with the best validation accuracy will be stored at the end of the training.

Model Selection

There are three hyperparameters of the model that need to be optimized, i.e. learning rate of the Adam optimizer, dropout keep probability of the convolutional layers, and dropout keep probability of the fully-connected layers.

The moment estimate decay rates (β_1 and β_2) of the Adam optimizer are not included into the hyperparameter optimization. They are set to the Tensorflow default values of 0.99 and 0.999 respectively.

Random search with uniform distribution is used to explore the possible combination of the hyperparameters. Ranges of interest for each of the hyperparameter are as listed in Table 3.



Hyperparameter	Lower limit	Upper limit
Learning rate	0.0005	0.005
Dropout Keep Prob (Conv)	0.7000	0.900
Dropout Keep Prob (FC)	0.3000	0.500

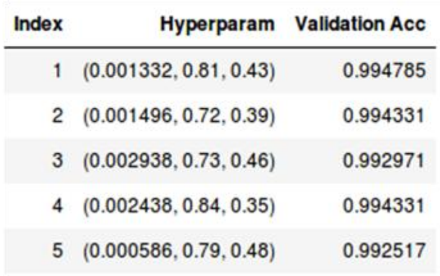
Table 3 – Ranges of interest for hyperparameter optimization.

The randomized search is run for 5 iterations. Model of the iteration with the best validation accuracy is used as the final model.

IV. Results

Model Evaluation

The result from the hyperparameter optimization is as shown in Table 4 below.

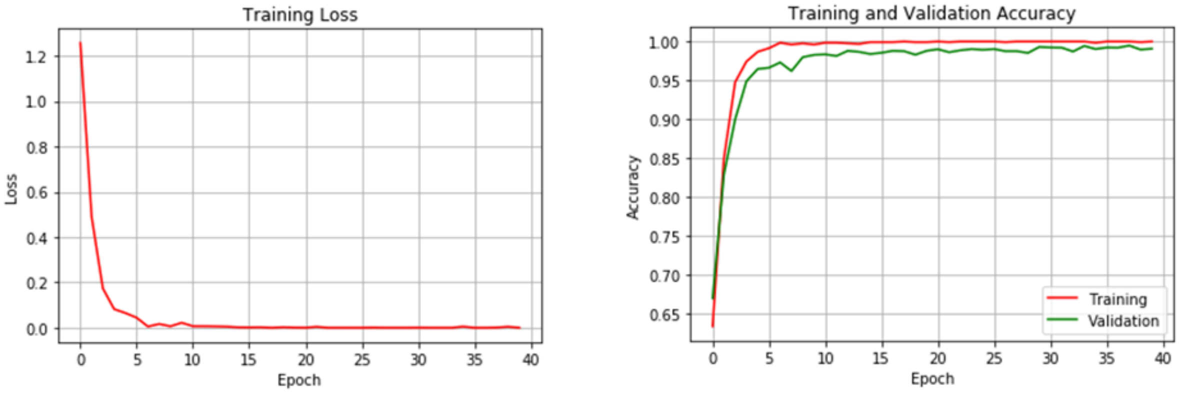


Index	Hyperparam	Validation Acc
1	(0.001332, 0.81, 0.43)	0.994785
2	(0.001496, 0.72, 0.39)	0.994331
3	(0.002938, 0.73, 0.46)	0.992971
4	(0.002438, 0.84, 0.35)	0.994331
5	(0.000586, 0.79, 0.48)	0.992517

Table 4 – Result of hyperparameter optimization.

The best model is model 1 which has best validation accuracy of 99.48%.

The loss and accuracy for model 3 during training are as shown in Figure 9.

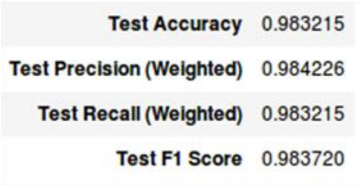


(a)

(b)

Figure 9 –(a) Training loss of the model during training, (b) Training and validation accuracy of the model during training.

By using this final model, performance on the test set is evaluated and as listed in Table 5.



Test Accuracy	0.983215
Test Precision (Weighted)	0.984226
Test Recall (Weighted)	0.983215
Test F1 Score	0.983720

Table 5 – Performance metrics for test set evaluation.

Test the Model on New Images

Five images of German traffic signs are downloaded from the web for evaluating performance of the model on new images. The images are as shown in Figure 10.

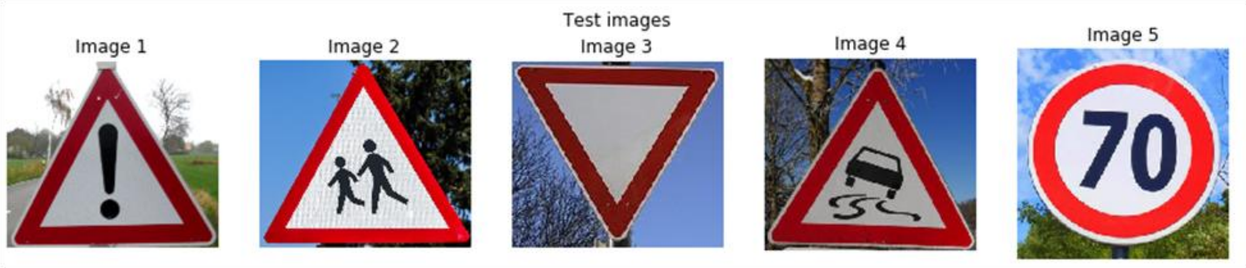


Figure 10 – New test images.

These images are preprocessed using the image preprocessing pipeline and are assigned the appropriate ground truth labels as shown in Figure 11.

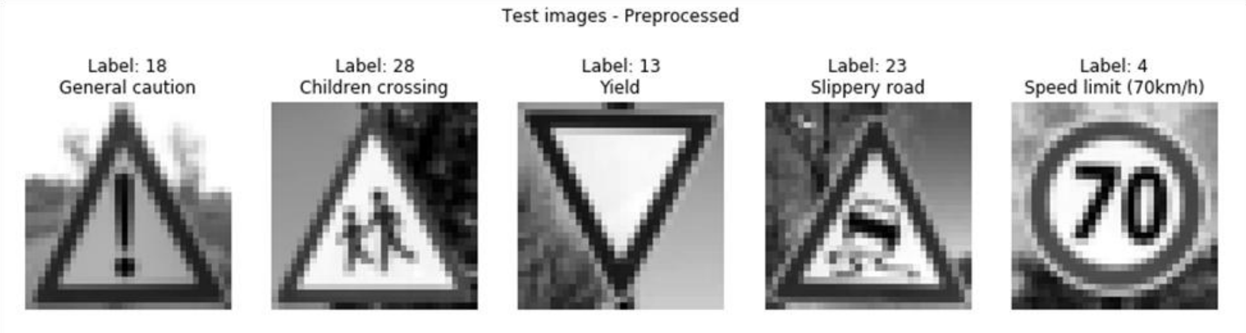
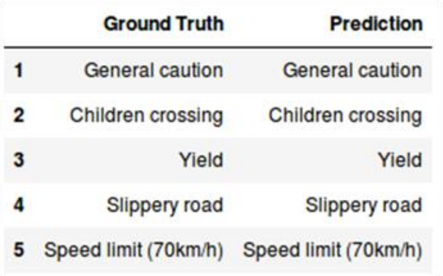


Figure 11 – Preprocessed new test images with assigned ground truth labels.

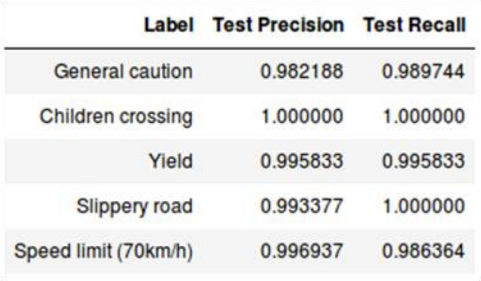
The final model achieved 100% accuracy for classification of these five images. The prediction outputs of the model are as listed in Table 6.



	Ground Truth	Prediction
1	General caution	General caution
2	Children crossing	Children crossing
3	Yield	Yield
4	Slippery road	Slippery road
5	Speed limit (70km/h)	Speed limit (70km/h)

Table 6 – Predictions of the model for new images.

To further comprehend the predictive power of the model, precision and recall of the model based on the test set are determined for class labels of the five new images. The two metrics are as shown in Table 7.



Label	Test Precision	Test Recall
General caution	0.982188	0.989744
Children crossing	1.000000	1.000000
Yield	0.995833	0.995833
Slippery road	0.993377	1.000000
Speed limit (70km/h)	0.996937	0.986364

Table 7 – Precision and recall for class labels in the new images.

From the test set precision and recall of the model, the 'General caution' class has the lowest precision among the others, hence the model may have a relatively higher chance of producing false positives for this class compared to the rest.

On the other hand, the 'Speed limit (70km/h)' class has the lowest recall, hence the model may have a relatively higher chance of producing false negatives, i.e. this class being mistaken classified as other classes.

Since the output nodes of the model are softmax activations, output value of each node is interpreted as probability of each class being the correct label. Instead of just producing one single most probable prediction, the prediction output of the model can be visualized as probabilities of top 5 most probable predictions, see Figure 12.

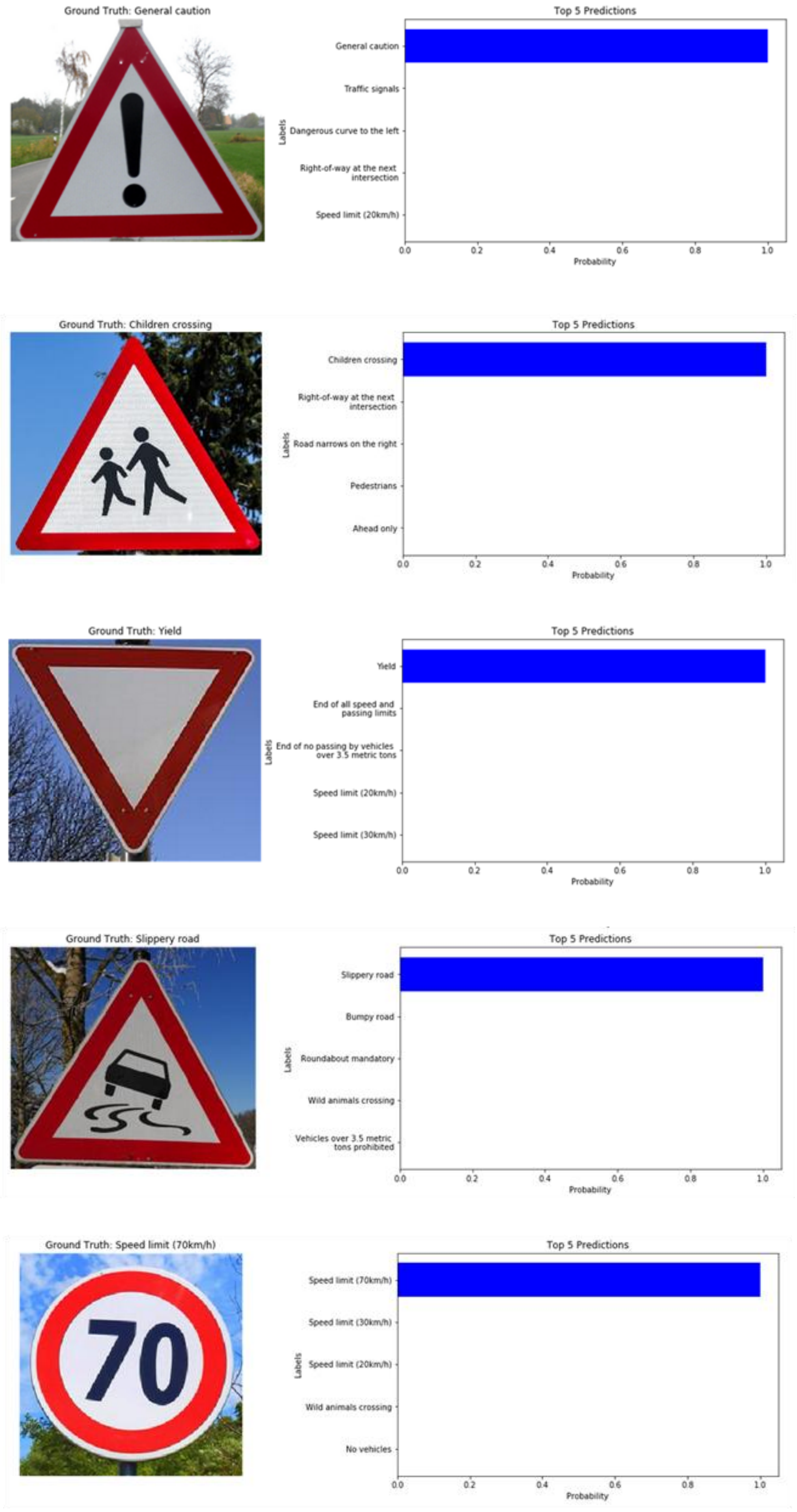


Figure 12 – Top 5 most probable predictions for the new test images.

The visualization of the top 5 predictions sorts the class probabilities in descending order and displays them in a horizontal bar chart from top to down. It is observed that the final model is very confident about its top predictions for all five new images, with probability close to 1.0.

Visualizing the Neural Network

The activations of the first layer of the model can be visualized for a particular input image. These activations represent feature maps extracted by learnt convolution mask weights of the layer. The feature maps for one of the new test images are as shown in Figure 13.

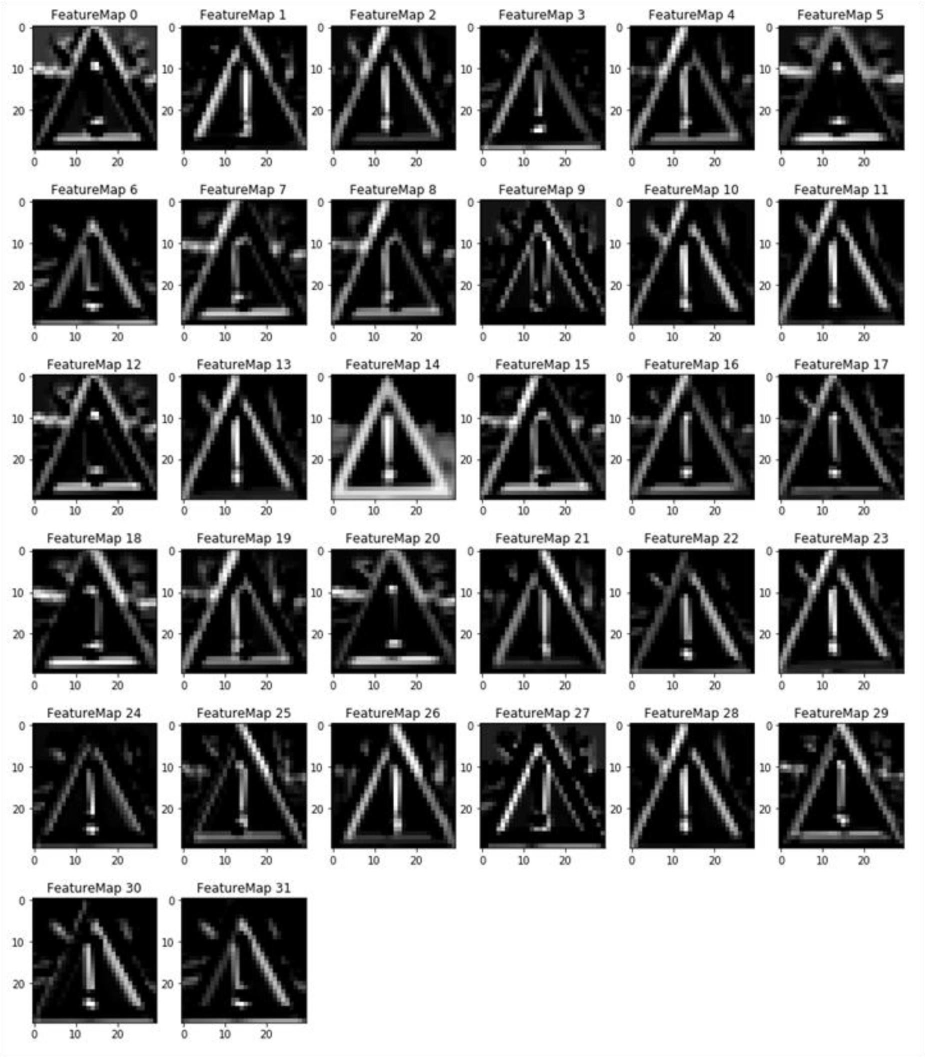


Figure 13 – Feature maps from the first convolutional layer.

Observing from the feature maps in Figure 13, the 3x3 convolution masks learnt in the first convolutional layer are similar to edge detection filter for edges with different orientations.

The convolutional layers of CNN essentially serve to extract features in a hierarchical manner. Early layers extract low level features such as edges of different orientations, while subsequent layers extract combinations of different features from the incoming feature maps. Through backpropagation with training data, optimum weights for the convolution masks are learnt with the objective to minimize the loss function.

V. Conclusion

Reflection

In this project, a Convolutional Neural Network is implemented for classification of German traffic signs with the GTSRB dataset. Test accuracy of 98.32% is achieved with a modest model architecture of 4 convolutional layers and 2 fully connected layers.

One interesting aspect during the implementation phase is the class imbalance issue in the training set. The validation performance during training is inferior when the class imbalance issue is not properly addressed. This seems to be related to the fact that the model is trained with relatively more samples for certain classes. Optimization at each epoch tends to refine the model to classify the classes with more samples better, while putting less weight to improve the other minority classes. When data augmentation is performed on the training data to make each class label has the same number of samples, significant improvement is observed in validation performance during training of the model.

Improvement

One potential way to achieve better performance is to explore transfer learning with pre-trained networks such as AlexNet, VGGNet, Inception, etc. One step further is to try to ensemble feature extraction part of multiple pre-trained networks and then build a deep neural network to perform classification based on these extracted features.