Report Prepared By:

**Saswati Halder**

GitHub link: <https://github.com/haldersas/Recognizing-Traffic-Signs-using-CNN-Keras/blob/main/DSCp_MidtermProjRTS.ipynb>

Submitted to:

**Dr. Claudio Delrieux**

Recognizing traffic Signs using CNN

# Executive Summary:

Traffic signs come in different colors, shapes, in form of pictures and are easy to understand. They represent rules that keep us safe, and help communicate messages to drivers and pedestrians. It’s important to know what each picture represents. Failing to do so could result in serious accidents or fine.

Computer vision is advancing rapidly with the help of Deep Learning which plays an important role in driver-assistance systems and driverless operations to detect other cars and objects on the roads to avoid collision. Equally important for these driverless vehicles is to recognize traffic signs under different traffic and lighting conditions, day and night. Being able to correctly recognize the traffic signs ensures road safety, avoid breaking traffic laws and collisions, and finally contributes to the success of the cause.

For the predictive modeling and image classification of traffic signs, I have used CNN (Convolutional Neural Network), a subnet of Neural Network which is very good at image and video processing. With small colored images of size 32x32x3, taken under different traffic and lighting condition, most of the images are of poor quality and hard to detect. The data came split into train, validation and test set. Keeping the test data unseen, we prepared the data and trained a CNN model. We trained two models, one with not normalized features and the second one with normalized features (to reduce effect of uneven illuminations). Bost models performed really well during training as well as on validation data. When evaluating on test data, the not-normalized trained model was tested on not-normalized test data and the normalized trained with normalized test.

With both the test models, we achieved an accuracy of close to 98% which proved that the CNN model that we trained was working successfully.

Analysis and Predictive Modeling using CNN Keras: Recognizing Traffic Signs

# Dataset:

The source of the dataset is Kaggle.com. The data comes as archived pickle files in train, validation and test set. There are 37499 rows of data in train set, 4410 rows in validation set and 12360 rows in the test set. The features are colored images of size 32x32x3 in form of NumPy arrays, data type is uint-8 (unsigned integer 8-bit). There are 43 different classes of labels in form of integers ranging from 0 to 42 and there is no missing data.

The main challenge was that the images were taken under different lighting conditions, from light to dark. With small size, the images lacked clarity with some images quite blurry.

# Data Preparation:

All the train, validation and test datasets were split into features and labels. Some of the images were previewed from the train set. As said before, the images are not clear and some dark. Hence, we will normalize all the features (including test) to eliminate the effect of uneven illumination.

While doing exploratory analysis, we also found imbalance in the class distribution of data in the train set which have been addressed using data augmentation. Data augmentation, applies random image height or width shift, image rotation and zoom, keeping the original images to increase the number of images. This helps avoid overfitting and generalize better.

Here are some visuals from the train set from exploratory data analysis and preparations.

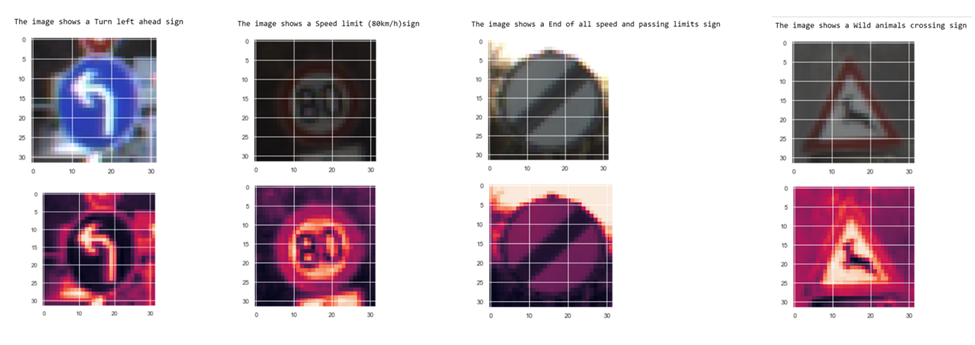


Fig. 1. Figure shows before and after images went through normalization. Note the more distinct features and mainly the changes in the second image set.

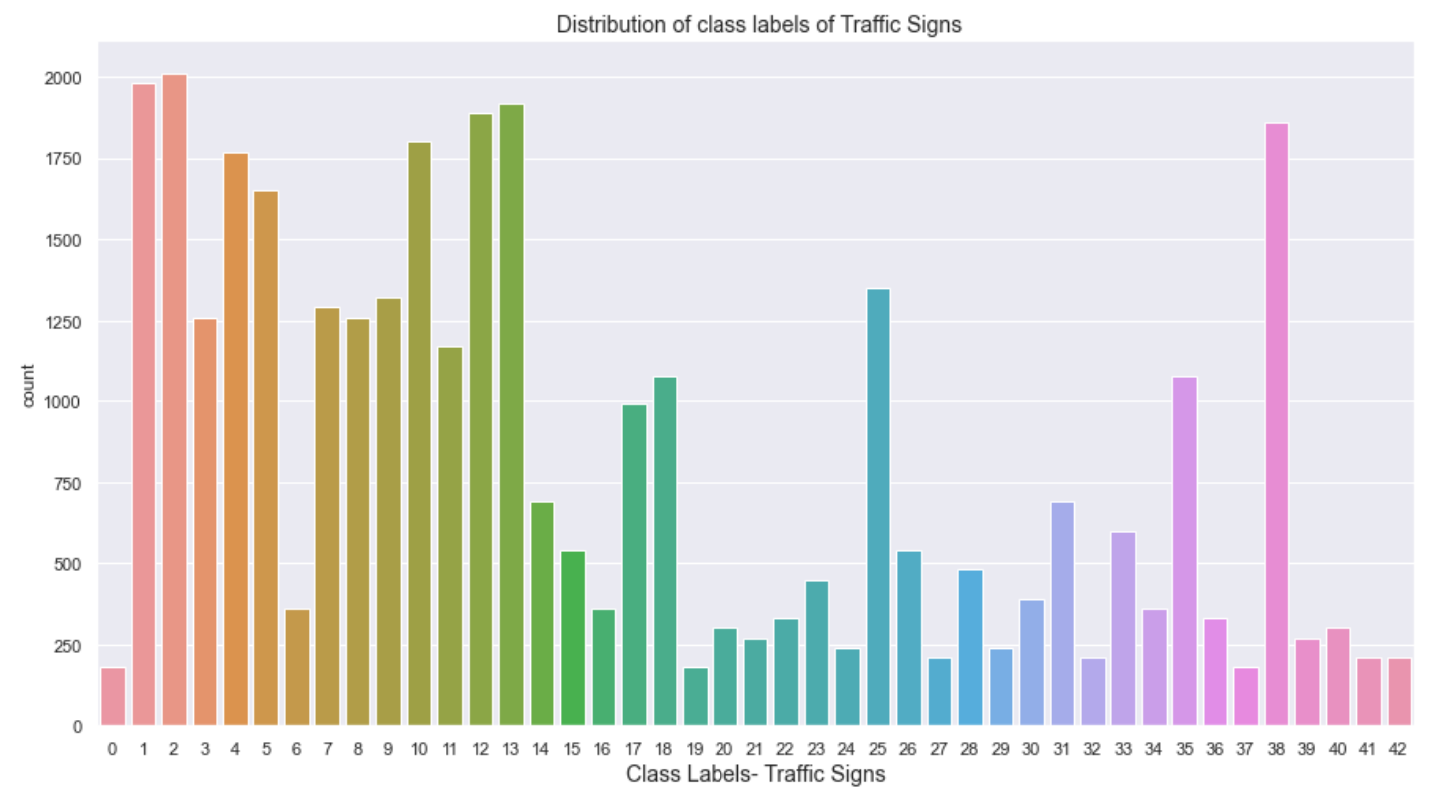


Fig. 2. Figure shows imbalanced class distribution (count of labels) from the train set

To prepare for modeling, the labels were encoded from integers to one-hot encoding.

# CNN Model:

Convolutional Neural Network, a subset of Deep Learning is a very powerful Neural Network algorithm for image and video processing. The model has an input layer, the hidden layer and an output layer. The hidden layer usually has a few convolutional or conv layers, alternating with pooling layers that flattens out to connect to a fully connected layer and the output layer. We use ReLU (Rectified Linear Unit) as activation function to maintain non-linearity of the network model. Here is a basic CNN architecture.

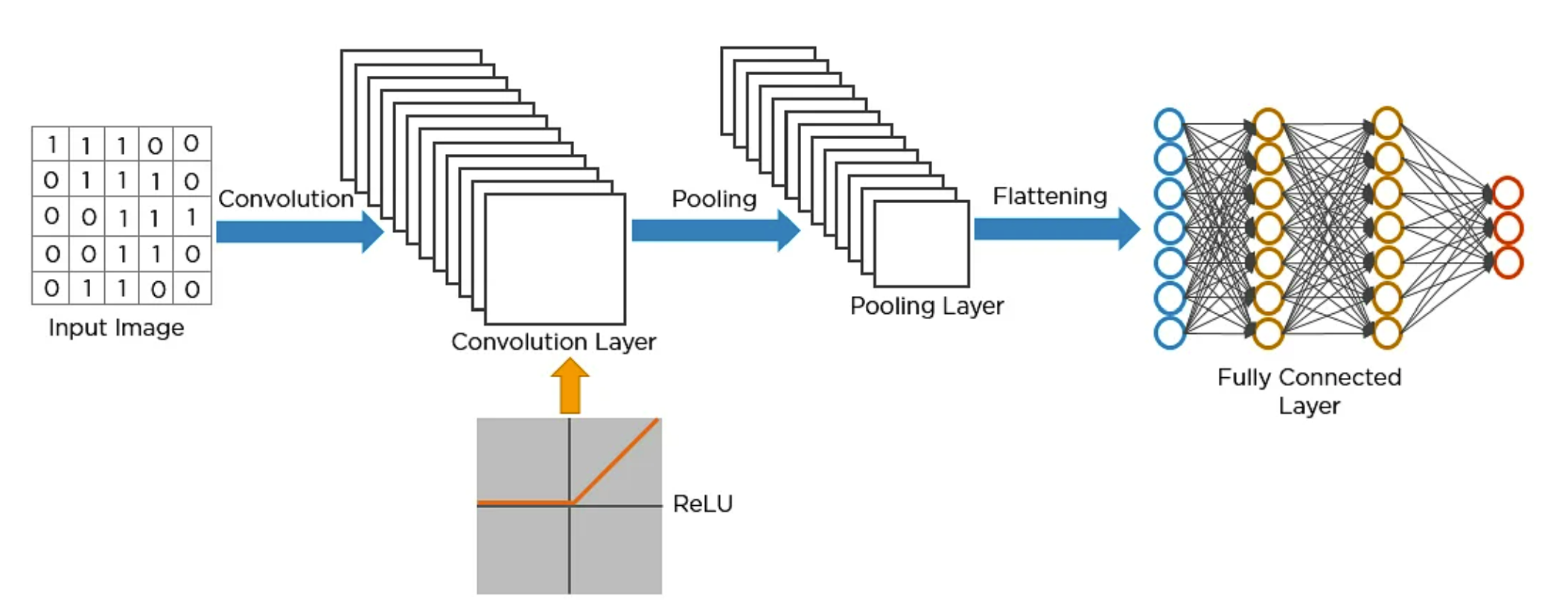


Fig. 3. Figure shows a basic CNN architecture with input image Convolution layers, Pooling layer flattening to a fully connected and output layer.

As an input image goes through the layers, it becomes smaller but deeper in size with distinct features retained as a feature map. Different filters are used to capture distinct features and edges. The pooling layer shrinks the input image size to reduce computational load. Finally, these layers flatten out to connect to a fully connected model that outputs the number of classes.

# Why use Convolutions?

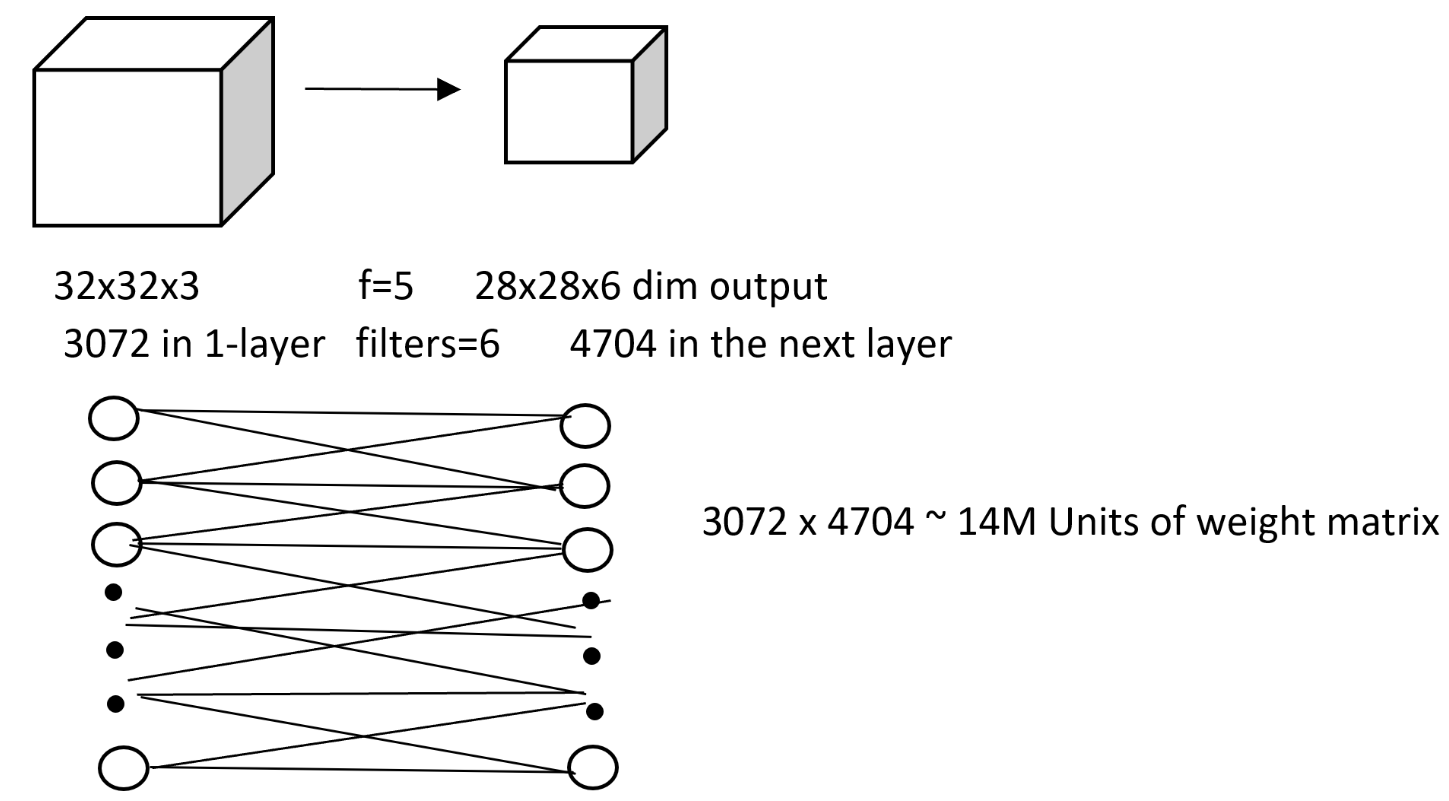


Fig. 4. Figure shows why CNN is used for image processing over NN model.

The convenience of using convolutional network is that is reduces computational load and uses:

Parameter Sharing: In convolutional network the number of parameters is reduced compared to neural network model. Conv net does parameter sharing, so features parameters from one part of the image can be used in lots of different positions within the same image.

Sparsity Connections: In each layer, each output value depends only on a small number of inputs.

# Define our CNN Model:

A Tensorflow background and Keras Sequential API, was used to train our model, where we add one layers at a time, starting from the input layer.

The first layer is the input convolutional layer (Conv2D) followed by another convolutional layer with 32 filters each. The last two Conv2D layers have 64 filters each. Each filter detects a distinct type of features say horizontal edge from the image to create a feature map. The other parameters for Conv2D are padding and activation function. Padding="same" means we add a border of zero around our input image size so that the input dimensions remain same as the output dimensions. The activation function "ReLU" is used to add non-linearity to the network.

The convolutional layers alternates with the pooling layer that shrink the input image. Our model uses Max Pooling which preserves only the strongest feature, meaning, only the max input value in each receptive field makes it to the next layer. This reduces the computational load, the memory usage and prevents overfitting. After trial runs, Batch Normalization and Dropout layers were added to the model to acts as regularization and prevent overfitting.

# Train a Model with Features Not-Normalized

The first model CNN model that was compiled and fitted has features that were not normalized, but the features were augmented. We trained the model using the RMSprop optimizers with 30 epochs. Since the number of epochs were high, we also used Keras callbacks “Learning Rate Scheduling”. The training and validation accuracy were high with no overfitting. When we evaluated this trained model on not-normalized test data, it generalized really well with an accuracy close to 98%.

# Train a Model with the Features Normalized

For train set with features normalized, we first started fitting and training the CNN model with only four epochs. Epochs are the number of times the algorithm runs through the entire train set. For these smaller runs, three different types of optimizers were used to compile our model.

## Optimizers:

Adam: Adam which stands for adaptive moment estimation is an adaptive learning rate algorithm that requires less tuning. The default learning rate of 0.001 was used. Adam was the fastest to converge but had lower training accuracy than other optimizers.

Stochastic Gradient Descent: Used vanilla, SGD is slow to converge. Momentum=0.9 and Nesterov=True was used to help with faster convergence. SGD was fast to converse with high accuracy but can slow down with longer epochs.

RMSProp (Root Mean Square): RMSprop is an adaptive learning rate algorithm like Adam with good performance. RMSprop was fast to converge and had the best accuracy.

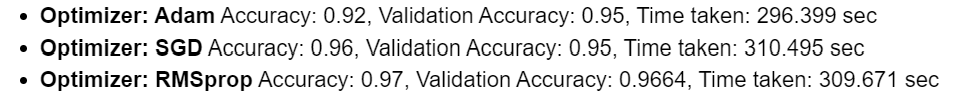


Fig. 3. Figure shows optimizer performance over 4 epochs

Selecting RMSprop to be the optimizer for our final model with 30 epochs to run, we set Keras callbacks as “Learning Rate Scheduling” which will be discussed under hyperparameter tuning. The final CNN model was compiled and fitted to run for 30 epochs. Using TensorFlow’s interactive performance evaluation dashboard called Tensorboard, the loss and accuracy could be traced after every epoch.

# Hyperparameter Tuning:

Some of the hyperparameter used for tuning the CNN model are Batch Normalization and Drop Out layers. Except for Adam and RMSprop, the SGD optimizer with momentum=0.9 and Nesterov=True were hyperparameters used for increasing performance. For training the model using 30 epochs, “Learning Rate Scheduling” as set as callbacks where starting learning rate was higher that decreased by half when preferred accuracy is not reached by certain epochs.

# Performance Evaluation:

After running 30 epochs, the final trained CNN model performance was evaluated. Time taken for 30 epochs was close to an hour (did not use GPU). With an accuracy of 0.9967 and validation accuracy of 0.9846 our CNN model performed well and showed no overfitting. When we evaluated our model on the unseen test data, accuracy was 0.9787 which was again, really good.

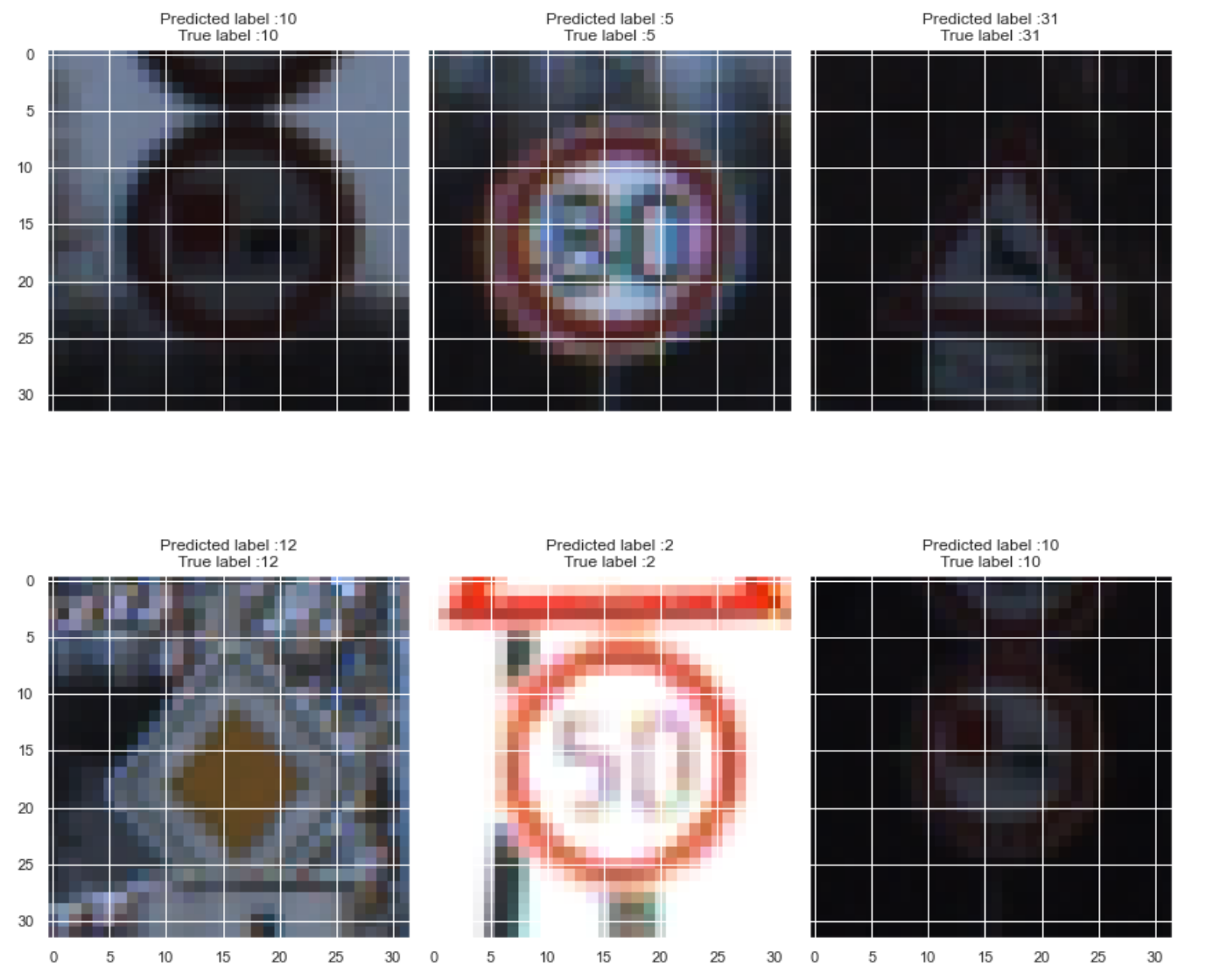


Fig. 5. Figure displays top 6 correctly predicted traffic signs

# Conclusion:

The conclusion was that our trained CNN model performed really well at bringing out distinct features and correctly predicting the traffic signs, with the given images. Looking at some of the correctly predicted images we can see that the image quality was very poor for even human beings to correctly determine the traffic signs.

# Credits:

Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow by Aurélien Géron

Stanford and Coursera Deep Learning Videos by Andrew Ng

<https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/>

<https://github.com/ageron>

Kaggle.com