Traffic Sign Classification

Gopireddy Arunteja Reddy
B180679EC
arunteja_b180679ec@nitc.ac.in

Jarapala Mohith Pamar
B181027EC
mohith_b181027ec@nitc.ac.in

Gurram Deepak
B180700EC
deepak_b180700ec@nitc.ac.in

***Abstract*—Emerging technologies like Artificial Intelligence play a vital role in the industry of automation. Automobile industry in particular has a very important connection with AI and Deep learning. Multinational companies like Tesla, Uber, Google, Mercedes-Benz, Audi, etc are working on autonomous vehicles and self-driving cars. One of the key aspects to this technology, is that the vehicles should be able to interpret traffic signs and make decisions accordingly. There are various classifications of traffic signs like speed limits, no entry, turn left or right, children crossing, no passing of heavy vehicles, etc. This project is intended to build a convolutional neural network model that can classify images of traffic sign boards into different categories.**

I. Introduction

Classification of road traffic sign boards using Convolutional Neural Networks.

II. Data set

Traffic sign recognition is a multi-class classification problem with unbalanced class frequencies. Traffic signs can provide a wide range of variations between classes in terms of color, shape, and the presence of pictographs or text. However, there exist subsets of classes (e.g., speed limit signs) that are very similar to each other. The classifier has to cope with large variations in visual appearances due to illumination changes, partial occlusions, rotations, weather conditions, etc. Humans are capable of recognizing the large variety of existing road signs with close to 100% correctness. This does not only apply to real-world driving, which provides both context and multiple views of a single traffic sign, but also to the recognition from single images.

We are using **GTSRB** - German Traffic Sign Recognition Benchmark. The German Traffic Sign Benchmark is a multi-class, single-image classification challenge held at the International Joint Conference on Neural Networks (IJCNN) 2011.

To discuss exact numbers, the data set consists of 51,840 images in total, which are divided as 39209 Train data images and 12631 Test data images. The Train data is further classified into 43 classes each stating an attribute depicting some of the very common traffic board signs we come across in daily commute.

III. Architecture

We are building a sequential CNN architecture model using keras.

- We start building our model by a convolution layer with 16 filters, kernel size = (3,3) and Rectified Linear Unit Activation function and we mention input shape.
  This is followed by another convolutional layer with number of filters changed to 32. The function used is keras.layers.Conv2D
  In Convolutional Neural Networks, Filters detect spatial patterns such as edges in an image by detecting the changes in intensity values of the image.
- We do Max Pooling with pool size = (2,2) using the command keras.layers.MaxPool2D
  Max pooling is done to help over-fitting by providing an abstracted form of the representation. Pooling mainly helps in extracting sharp and smooth features. It is also done to reduce variance and computations.
- Then, We have a Batch Normalization layer.
  Batch normalization is a method to regularize a convolutional network. On top of a regularizing effect, batch normalization also gives your convolutional network a resistance to vanishing gradient during training. This can decrease training time and result in better performance.

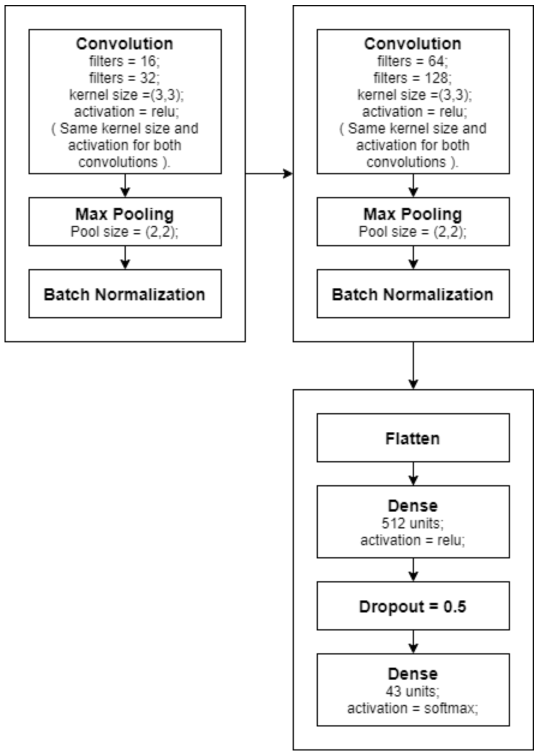


Fig. 1. CNN Model

- Next layer is again a convolutional layer with 64 filters, same kernel size and activation function as before. Followed by another convolutional layer with 128 filters and same attributes as above.
- We have the same Max pooling layer and Batch Normalization layer as done before.
- Then, We have a Flatten layer followed by a dense layer with 512 units and relu activation function.
  Flattening is converting the data into a 1-dimensional array for inputting it to the next layer. We flatten the output of the convolutional layers to create a single long feature vector.
  The dense layer is a neural network layer that is connected deeply, which means each neuron in the dense layer receives input from all neurons of its previous layer. The dense layer is found to be the most commonly used layer in the models.
  In the background, the dense layer performs a matrix-vector multiplication. The values used in the matrix are actually parameters that can be trained and updated with the help of back-propagation.
- We then introduce a Dropout layer with dropout = 0.5.
  The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent over fitting.
- Finally, We have a dense layer with 43 units matching the number of classes of data under consideration and with softmax activation function

IV. Working

A. Importing dependencies

We have used library packages like numpy, pandas, matplotlib, cv2, tensorflow, Python Imaging Library (PIL), sklearn, keras to achieve this project.

B. Image Preprocessing

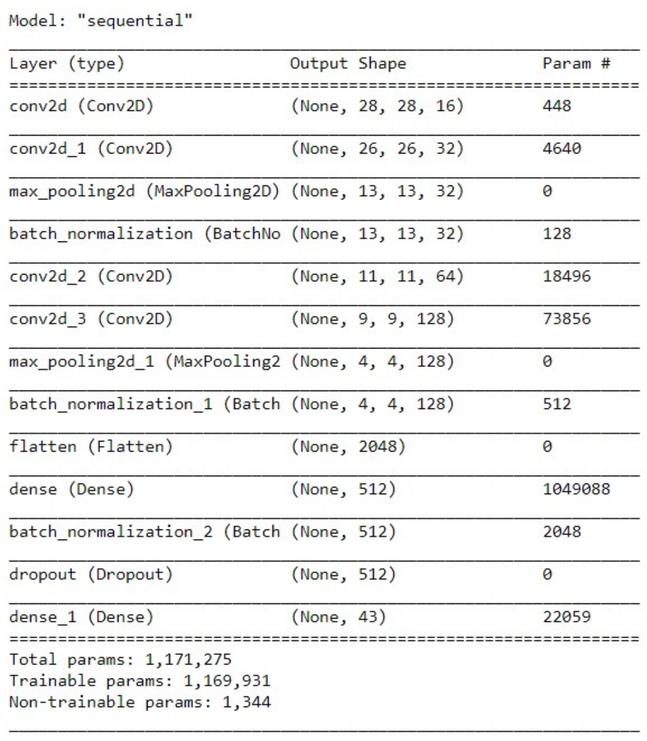
Changing the directory path to where the dataset is placed, We access the train data using necessary commands and then do the following:

- Create two null lists to store the image data and their respective labels.
- We access the data in Train folder, in which there are a total of 43 classes.
- Each image in those folders is resized, converted into an array with corresponding pixel data and is appended to the data list.
- And the respective image's class is appended to the labels list.
- Now, both these lists are converted into numpy arrays.
- As the above mentioned preprocessing for all the images in the dataset is a tedious job and time consuming depending on the size of the dataset, we save the data and labels for future reference.

C. Training and Testing

- We have divided both the data and labels by the ratio of 80% for training and 20% for testing by using train_test_split function in sklearn.model_selection.
- The y_train and y_test labels are converted to one hot encoding. A one hot encoding allows the representation of categorical data to be more expressive. Many machine learning algorithms cannot work with categorical data directly. The categories must be converted into numbers. This is required for both input and output variables that are categorical.
- Next, We build the CNN model as per the architecture and then pass the one hot encoded training data. Essentially, we are fitting the model with training data and compiling the model.
- For the model compilation, we have used "categorical cross entropy loss function" and "adam" optimizer with metrics for accuracy.
- We have considered 20 epochs with batch size = 32 to train the data. And then we have taken the 20% test data as validation data as you fit the model with data.
- We have plotted the accuracy and loss of the fitted model using matplotlib.
- For Testing, We use Test.csv file in dataset and preprocess all the images similar to what is done with training data initially.
- Then, We predict the classes of this original data using the function predict_classes on our model to check how the model is performing. We verify the same using accuracy_score from sklearn.metrics library.
- To test the model on a new image, We write a function to take that image as input and preprocess it much like it was done with training data and testing data i.e., resize it, convert it into an array, append the information to data list.
- We mention the names of the 43 classes we considered by creating a dictionary with keys ranging from 0 to 42 and their corresponding values are names of the classes.
- Now, We predict the class of the new image using the function model.predict_classes and return the prediction output.

V. Results



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 16)        448
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 26, 26, 32)        4640
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 13, 13, 32)        128
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 9, 9, 128)         73856
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
batch_normalization_1 (Batch (None, 4, 4, 128)         512
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 512)               1049088
_________________________________________________________________
batch_normalization_2 (Batch (None, 512)               2048
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 43)                22059
=================================================================
Total params: 1,171,275
Trainable params: 1,169,931
Non-trainable params: 1,344
_________________________________________________________________
```

Fig. 2. Model Summary

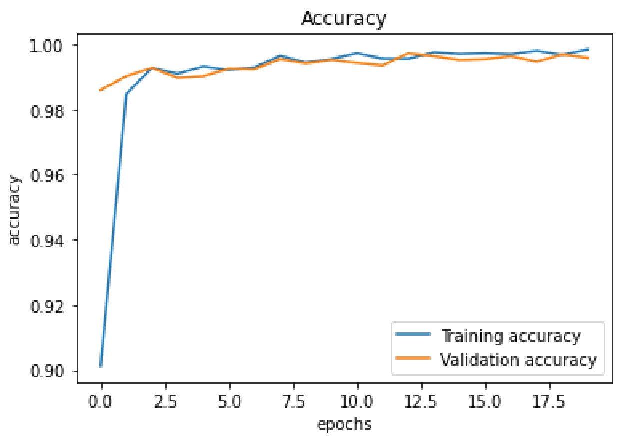


Fig. 3. Model Accuracy

- Accuracy with Training data = 99.84%
- Validation Accuracy = 99.58%

  Accuracy with Test data:
  - We have imported data from Test.csv file and preprocessed the data and We use predict_classes function on our model to predict the output.
  - We have imported the function accuracy_score from the library sklearn.metrics to verify our model with the test data present in dataset.
  - Accuracy score obtained = **97.5534**%

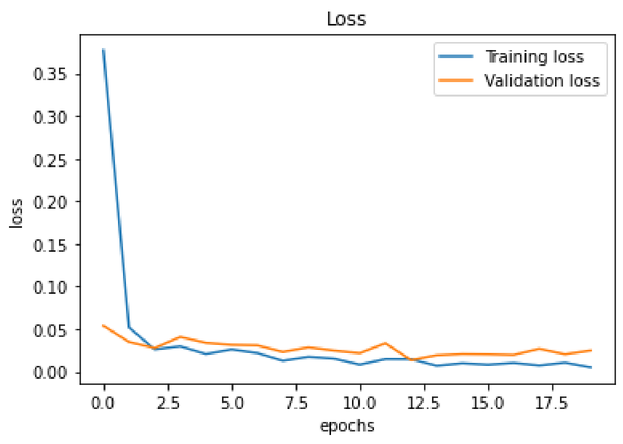


Fig. 4. Model Loss

- Loss with Training data = 0.5%
- Validation Loss = 2.47%

VI. Testing model with Real data

- We are using an image downloaded from the internet (not taken from dataset) to check the image preprocessing result and the prediction of our CNN model.

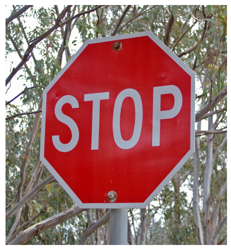
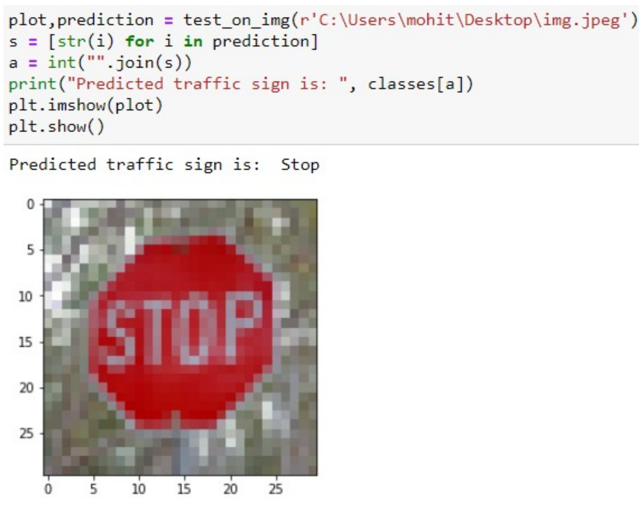


Fig. 5. Original Image



```
plot,prediction = test_on_img(r'C:\Users\mohit\Desktop\img.jpeg')
s = [str(i) for i in prediction]
a = int("".join(s))
print("Predicted traffic sign is: ", classes[a])
plt.imshow(plot)
plt.show()
```

```
Predicted traffic sign is:  Stop
```

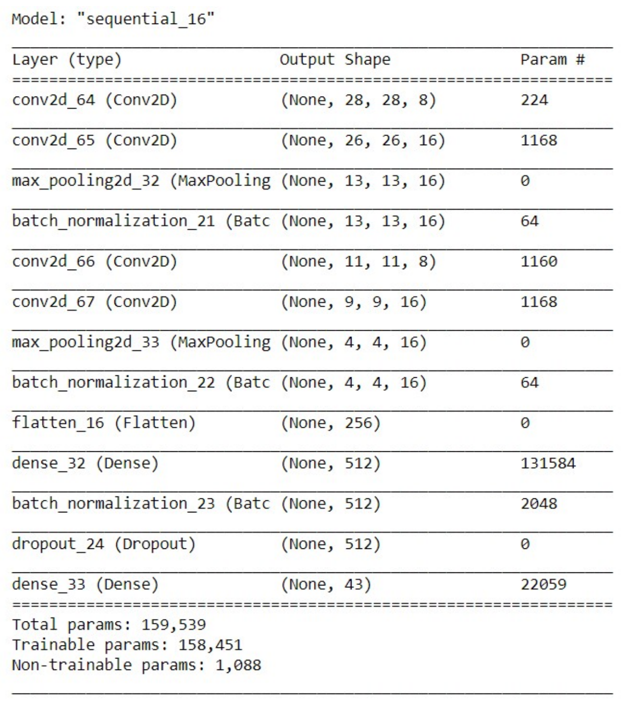
Fig. 6. Preprocessed image with Prediction result

VII. Experimentation

We have changed some of the model attributes so as to observe what change they constitute in the prediction and to observe the accuracy and loss metrics.

A. *Change in number of filters*

- We have changed the number of filters in convolutional layers to 8, 16; 8, 16 in sequential order with same kernel size = (3,3) and relu activation for all convolutional layers.



```
Model: "sequential_16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_64 (Conv2D)           (None, 28, 28, 8)         224       
_________________________________________________________________
conv2d_65 (Conv2D)           (None, 26, 26, 16)        1168      
_________________________________________________________________
max_pooling2d_32 (MaxPooling (None, 13, 13, 16)        0         
_________________________________________________________________
batch_normalization_21 (Batc (None, 13, 13, 16)        64        
_________________________________________________________________
conv2d_66 (Conv2D)           (None, 11, 11, 8)         1160      
_________________________________________________________________
conv2d_67 (Conv2D)           (None, 9, 9, 16)          1168      
_________________________________________________________________
max_pooling2d_33 (MaxPooling (None, 4, 4, 16)          0         
_________________________________________________________________
batch_normalization_22 (Batc (None, 4, 4, 16)          64        
_________________________________________________________________
flatten_16 (Flatten)         (None, 256)               0         
_________________________________________________________________
dense_32 (Dense)             (None, 512)               131584    
_________________________________________________________________
batch_normalization_23 (Batc (None, 512)               2048      
_________________________________________________________________
dropout_24 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_33 (Dense)             (None, 43)                22059     
=================================================================
Total params: 159,539
Trainable params: 158,451
Non-trainable params: 1,088
_________________________________________________________________
```

Fig. 7. Model summary after change in number of filters

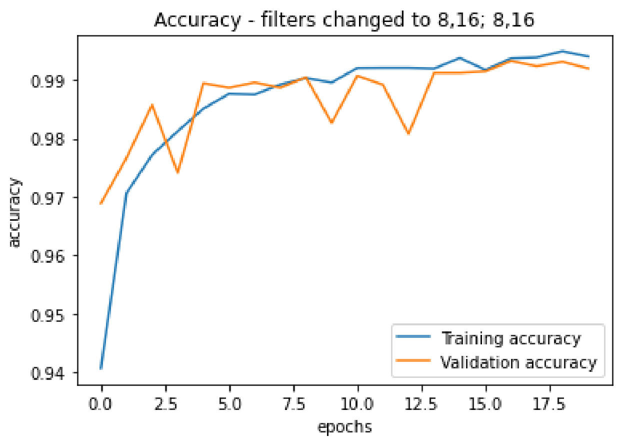


Fig. 8. Accuracy after change in number of filters

- Accuracy with Training data = 99.40%
- Validation Accuracy = 99.20%

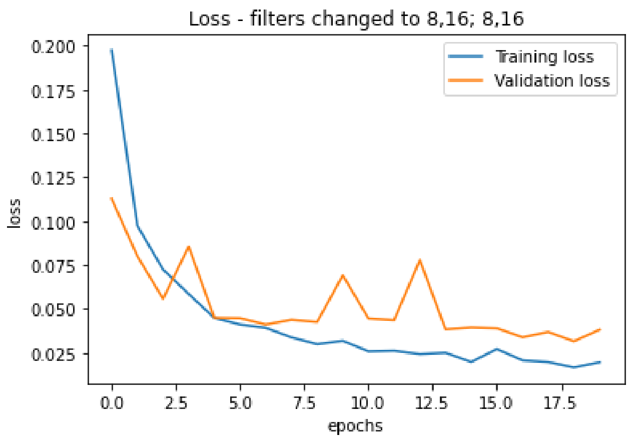


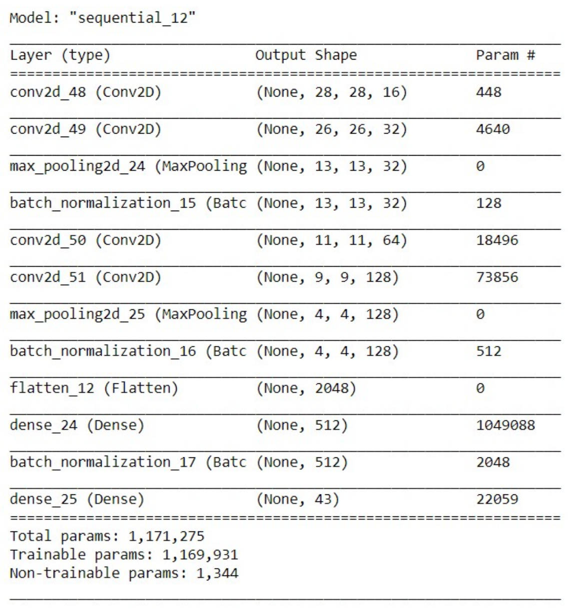
Fig. 9. Loss after change in number of filters

- Loss with Training data = 1.95%
- Validation loss = 3.80%

Inference: On comparison with our main model, we observe the accuracy is slightly decreased.
In contrast to this experimentation, having more number of filters will provide better results with less iterations(epochs) while compiling the model. i.e., Small batches of data can provide higher accuracy while fitting the data to our model.
We also observe that the loss of this experiment model is higher than that of our main model.

B. *Ignoring the dropout layer*

- We did not include a dropout layer after flatten, dense, batch normalization in this model.
  i.e., We only used the Batch Normalization layer to make up for the absence of dropout layer.



```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_48 (Conv2D)           (None, 28, 28, 16)        448       
_________________________________________________________________
conv2d_49 (Conv2D)           (None, 26, 26, 32)        4640      
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 13, 13, 32)        0         
_________________________________________________________________
batch_normalization_15 (Batc (None, 13, 13, 32)        128       
_________________________________________________________________
conv2d_50 (Conv2D)           (None, 11, 11, 64)        18496     
_________________________________________________________________
conv2d_51 (Conv2D)           (None, 9, 9, 128)         73856     
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 4, 4, 128)         0         
_________________________________________________________________
batch_normalization_16 (Batc (None, 4, 4, 128)         512       
_________________________________________________________________
flatten_12 (Flatten)         (None, 2048)              0         
_________________________________________________________________
dense_24 (Dense)             (None, 512)               1049088   
_________________________________________________________________
batch_normalization_17 (Batc (None, 512)               2048      
_________________________________________________________________
dense_25 (Dense)             (None, 43)                22059     
=================================================================
Total params: 1,171,275
Trainable params: 1,169,931
Non-trainable params: 1,344
_________________________________________________________________
```

Fig. 10. Model Summary after ignoring dropout

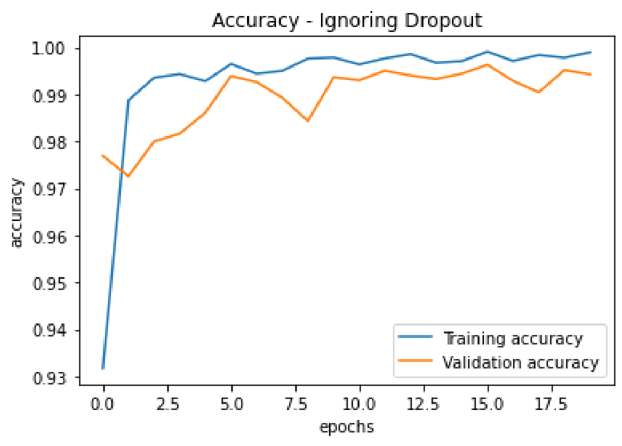


Fig. 11. Accuracy after ignoring dropout

- Accuracy with Training data = 99.89%
- Validation Accuracy = 99.43%

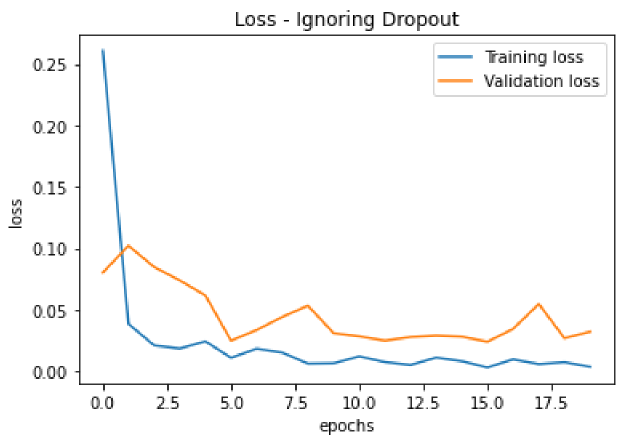
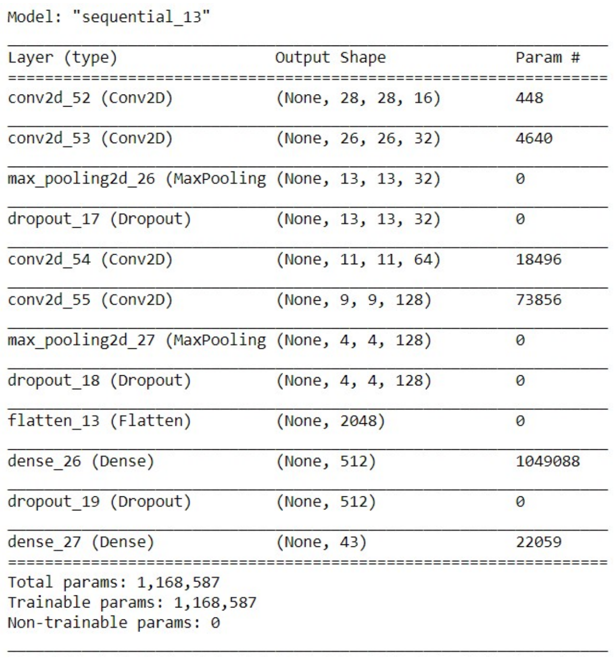


Fig. 12. Loss after ignoring dropout

- Loss with Training data = 0.38%
- Validation loss = 3.22%

  Inference: Ignoring dropout affects the validation accuracy.
  Because the training data is not being discarded which often results in higher accuracy with training data (in comparison with our main model).
  This also means the training data might be overfitting, which causes validation accuracy to be a lesser value than expected.



```
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_52 (Conv2D)           (None, 28, 28, 16)        448
_________________________________________________________________
conv2d_53 (Conv2D)           (None, 26, 26, 32)        4640
_________________________________________________________________
max_pooling2d_26 (MaxPooling (None, 13, 13, 32)        0
_________________________________________________________________
dropout_17 (Dropout)         (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_54 (Conv2D)           (None, 11, 11, 64)        18496
_________________________________________________________________
conv2d_55 (Conv2D)           (None, 9, 9, 128)         73856
_________________________________________________________________
max_pooling2d_27 (MaxPooling (None, 4, 4, 128)         0
_________________________________________________________________
dropout_18 (Dropout)         (None, 4, 4, 128)         0
_________________________________________________________________
flatten_13 (Flatten)         (None, 2048)              0
_________________________________________________________________
dense_26 (Dense)             (None, 512)               1049088
_________________________________________________________________
dropout_19 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_27 (Dense)             (None, 43)                22059
=================================================================
Total params: 1,168,587
Trainable params: 1,168,587
Non-trainable params: 0
_________________________________________________________________
```

Fig. 13. Model summary for considered dropout rates

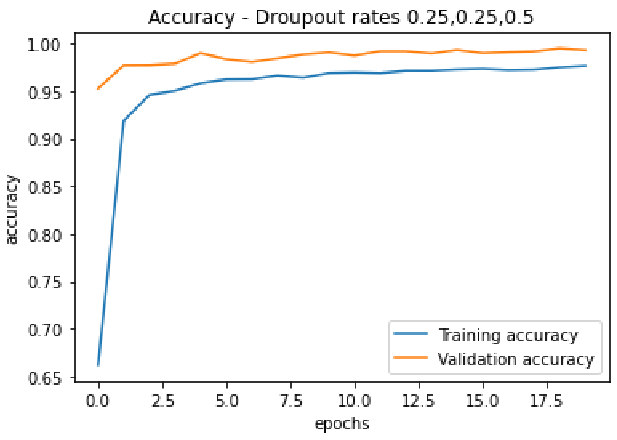


Fig. 14. Accuracy after taking only dropouts

- Accuracy with Training data = 97.65%
- Validation Accuracy = 99.31%

C. *Replacing Batch Normalization with Dropout layers*

- We have considered dropout layers and ignored Batch Normalization layers in this model.
  The rates of dropouts in the model are 0.25, 0.25, 0.5 sequentially.

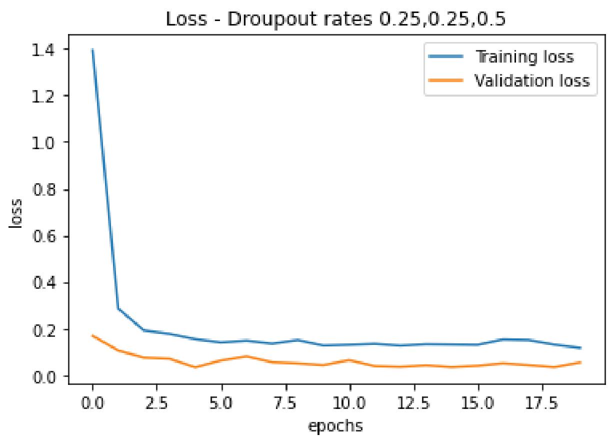


Fig. 15. Loss after taking only dropouts

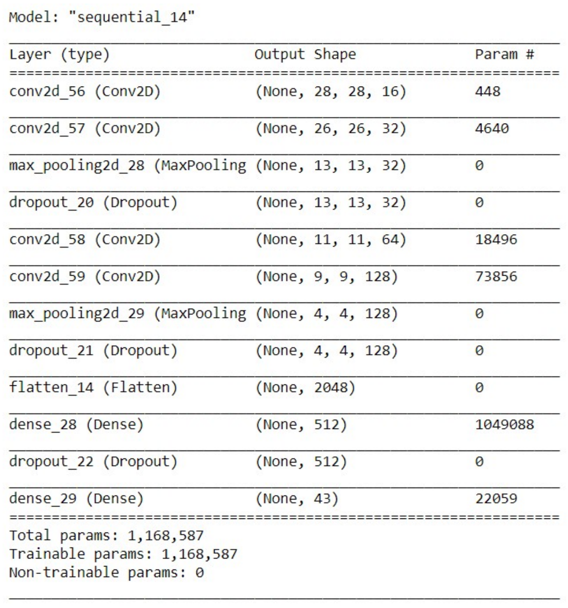
- Loss with Training data = 11.69%
- Validation loss = 5.44%

Inference: Due to more no. of dropout layers with high rates being used in this model, we can see that training accuracy significantly drops and the loss for training data is very huge.

This may also mean that the removal of training data more number of times may result in underfitting of the training data.

D. Altering dropout rates

- Same as the above model, but as the loss of training data is higher in previous case, we are decreasing the dropout rate values and see if that makes any considerable good.



```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_56 (Conv2D)           (None, 28, 28, 16)        448       
_________________________________________________________________
conv2d_57 (Conv2D)           (None, 26, 26, 32)        4640      
_________________________________________________________________
max_pooling2d_28 (MaxPooling (None, 13, 13, 32)        0         
_________________________________________________________________
dropout_20 (Dropout)         (None, 13, 13, 32)        0         
_________________________________________________________________
conv2d_58 (Conv2D)           (None, 11, 11, 64)        18496     
_________________________________________________________________
conv2d_59 (Conv2D)           (None, 9, 9, 128)         73856     
_________________________________________________________________
max_pooling2d_29 (MaxPooling (None, 4, 4, 128)         0         
_________________________________________________________________
dropout_21 (Dropout)         (None, 4, 4, 128)         0         
_________________________________________________________________
flatten_14 (Flatten)         (None, 2048)              0         
_________________________________________________________________
dense_28 (Dense)             (None, 512)               1049088   
_________________________________________________________________
dropout_22 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_29 (Dense)             (None, 43)                22059     
=================================================================
Total params: 1,168,587
Trainable params: 1,168,587
Non-trainable params: 0
_________________________________________________________________
```

Fig. 16. Model summary with altered dropout rates

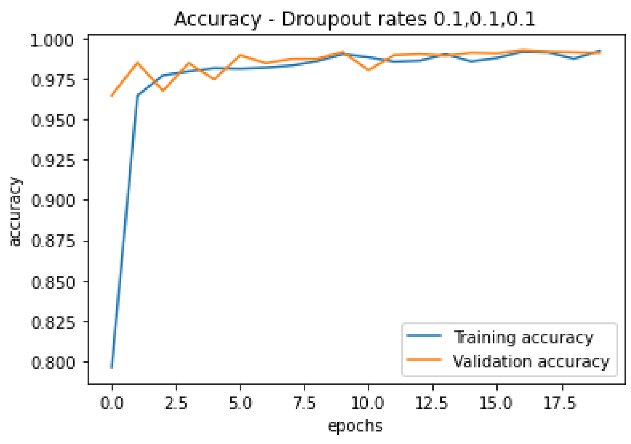


Fig. 17. Accuracy after altering dropout rates

- Accuracy with Training data = 99.24%
- Validation Accuracy = 99.12%

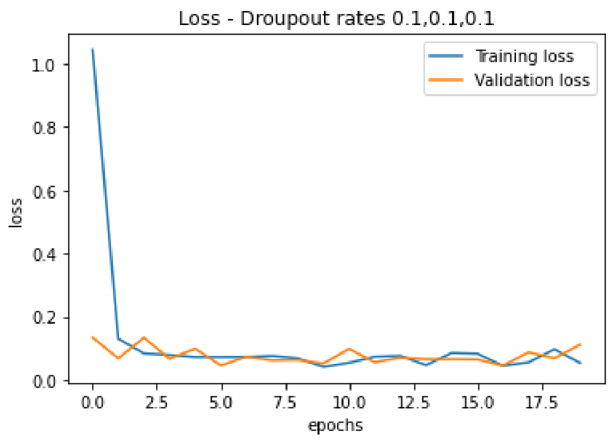


Fig. 18. Loss after altering dropout rates

- Loss with Training data = 5.42%
- Validation loss = 11.15%

Inference: As the dropout rates are decreased, the training data might have been overfitting, as we observe the validation loss to be a higher value this time.
Also, its worth noting that the Loss with training data and validation loss attributes are swapped when compared to the previous model.

E. Changing Optimizer

- Considering the same main model attributes, we are only changing the optimizer in the model compilation.
  We are considering ”rmsprop” and ”sgd” as alternatives to ”adam” optimizer we have used in our main model to check their performance with our model.

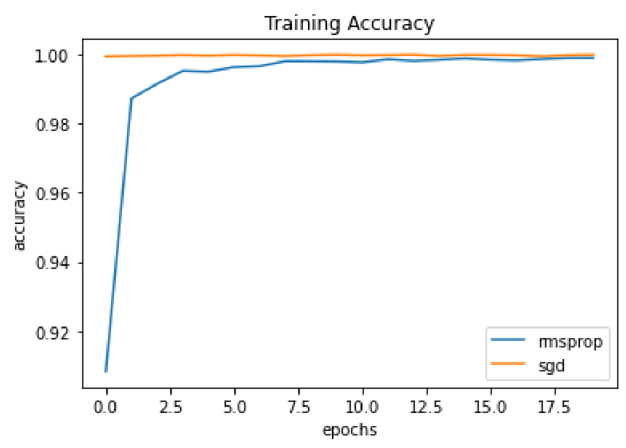


Fig. 19. Accuracy with Training data

- Training Accuracy using ”rmsprop” = 99.89%
- Training Accuracy using ”sgd” = 99.99%

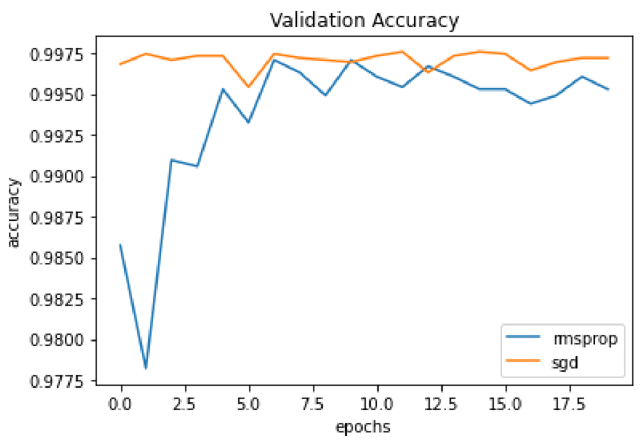


Fig. 20. Accuracy with Validation data

- Validation accuracy using ”rmsprop” = 99.53%
- Validation accuracy using ”sgd” = 99.72%

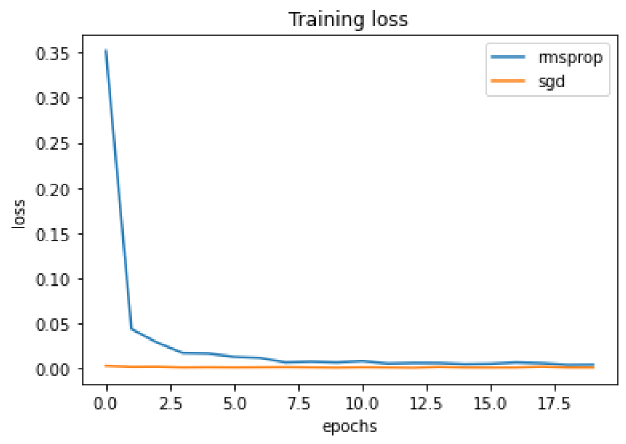


Fig. 21. Loss with Training data

- Training Loss using ”rmsprop” = 0.38%
- Training Loss using ”sgd” = 4.846 x 10^{-6}%

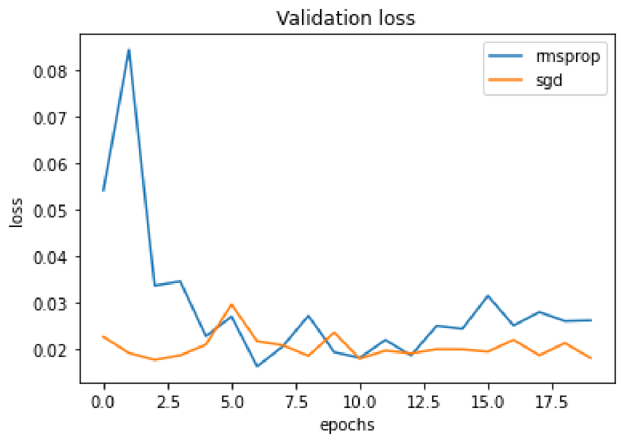


Fig. 22. Loss with Validation data

- Validation loss using ”rmsprop” = 2.62%
- Validation loss using ”sgd” = 1.82%

Inference: It is apparent that ”sgd” optimizer is better when compared to ”rmsprop” in terms of training and validation accuracy.
Generally, Adam optimizer is preferred because of its ease of design and it is more likely to return good results without an advanced fine tuning.

VIII. Conclusion

- The existence of Artificial Intelligence and deep learning in ever expanding and emerging industries like automobiles is of keen importance.
- Convolutional Neural Networks are used for image classification and recognition because of higher accuracy.
- Tuning various parameters while designing a CNN model is advised for matching the use case and to provide better results with higher accuracy.

References

[1] Data-set.[Online].Available:https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign
[2] Report-format.[Online].Available:https://www.ieee.org/conferences/publishing/templates.html