

Objective/Aim:

The project aims at developing a deep neural network model to recognize road signs such as speed limit signs , pedestrian crossing signs , hazard signs etc. This is achieved by using Convolutional Neural Networks to make the machine learn different types of road signs, and then providing the machine with Test data (unseen data) in order to check how well it can recognize different signs.

Technical Details:

The model is trained by using the TensorFlow Library of Python developed by Google.

Convolutional Neural Networks are Deep Neural Networks used to perform multitude of tasks such as Image & Video recognition, Image Analysis & Classification, Media Recreation, Recommendation Systems, Natural Language Processing, etc. all without any human intervention to the machine in whatsoever way.

A typical Grayscale image is defined as a matrix of pixels in a machine with each cell having a specific “weight” stored in it which defines the gradient of that pixel.

Convolutional Neural Network Architecture typically has three layers:

- 1) Convolutional Layer
- 2) Pooling Layer
- 3) Fully Connected Layer

Convolutional Layer

In the Convolutional Layer, the layer performs a dot product between two matrices, where one matrix is the set of learnable parameters otherwise known as a **kernel**, and the other matrix is the restricted portion of the **receptive field** (a small chunk of the entire image). The kernel is spatially smaller than an image but is more in-depth. This means that, if the image is composed of three (RGB) channels, the kernel height and width will be spatially small, but the depth extends up to all three channels.

During the forward pass, the kernel slides across the height and width of the image-producing the image representation of that receptive region. This produces a two-dimensional representation of the image known as an activation map that gives the response of the kernel at each spatial position of the image. The sliding size of the kernel is called a stride.

If we have an input of size $W \times W \times D$ and D_{out} number of kernels with a spatial size of F with stride S and amount of padding P , then the size of output volume can be determined by the following formula:

$$W_{out} = \frac{(W - F + 2P)}{S} + 1$$

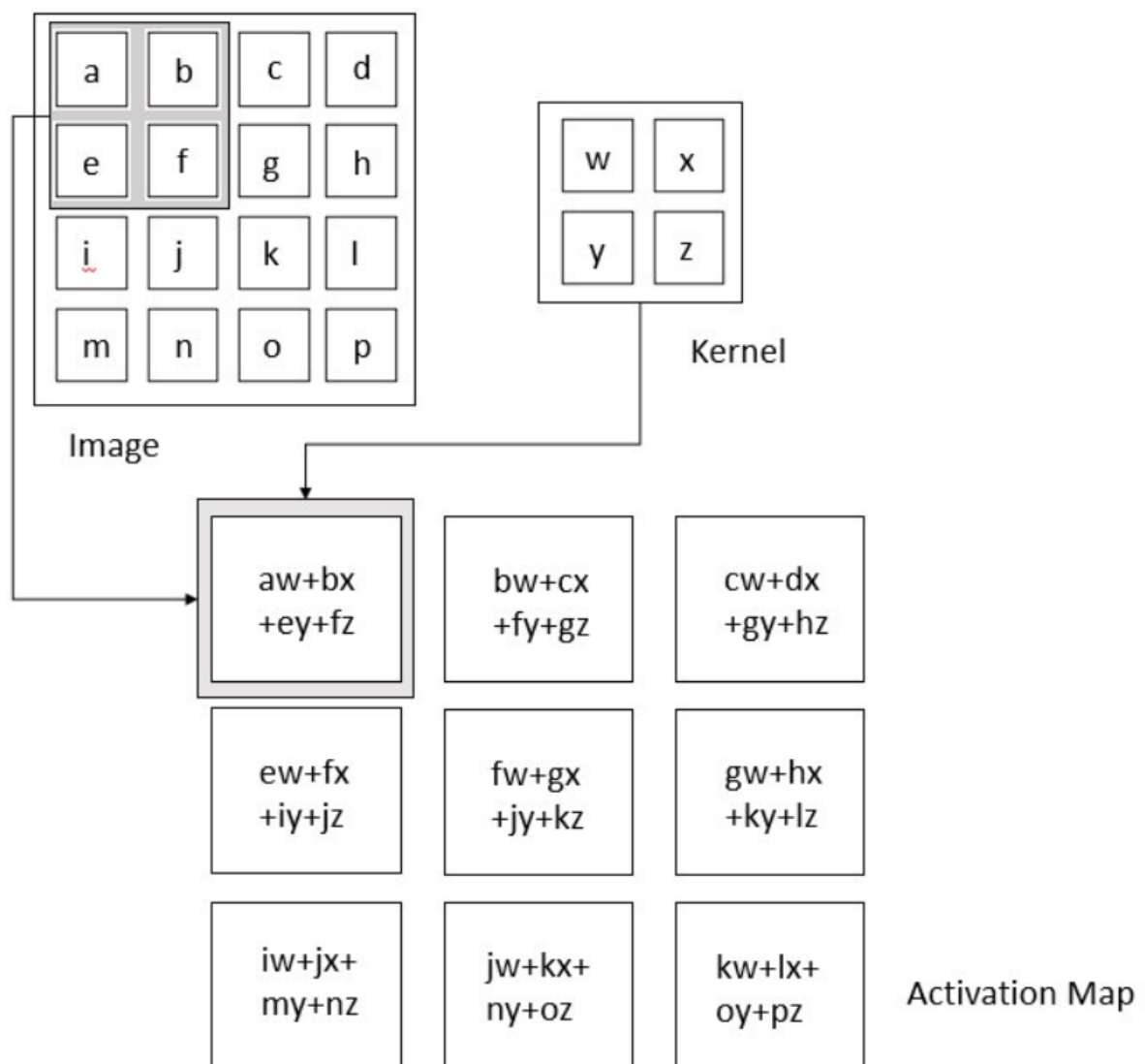
Where,

W = Input Volume Size

F = The receptive field size of the Conv Layer Neurons

S = The stride which is applied by the kernel

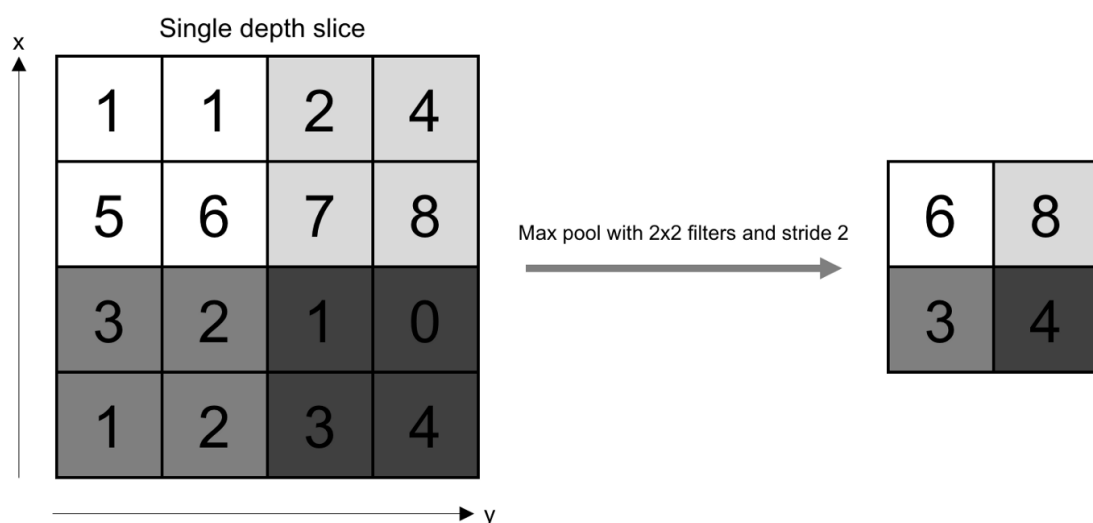
P = Amount of Padding used on the border



Pooling Layer

The pooling layer replaces the output of the network at certain locations by deriving a summary statistic of the nearby outputs. This helps in reducing the spatial size of the representation, which decreases the required amount of computation and weights. The pooling operation is processed on every slice of the representation individually.

There are several pooling functions such as the average of the rectangular neighbourhood, L2 norm of the rectangular neighbourhood, and a weighted average based on the distance from the central pixel. However, the most popular process is max pooling, which reports the maximum output from the neighbourhood.



Thus, if we have an input of size $W \times W \times D$, a pooling kernel of spatial size of F and a stride of S , the output volume comes out as :

$$W_{out} = \frac{(W - F)}{S} + 1$$

In all cases, pooling provides some translation invariance which means that an object would be recognizable regardless of where it appears on the frame.

Fully Connected Layer

Neurons in this layer have full connectivity with all neurons in the preceding and succeeding layer as seen in regular Neural Network. This is why it can be computed as usual by a matrix multiplication followed by a bias effect.

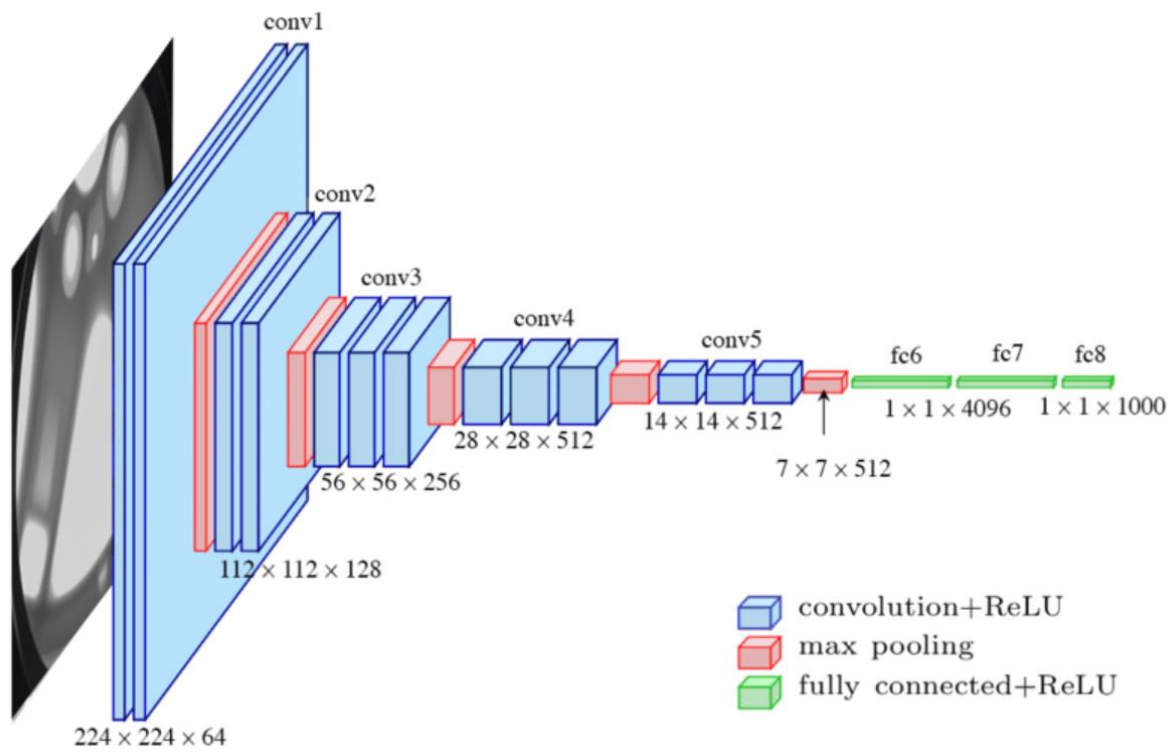
The FC layer helps to map the representation between the input and the output.

Since convolution is a linear operation and images are not linear for the majority of the time, linearity layers are also often placed directly after the convolution layers to introduce non-linearity to the activation map.

Some famous non-linearity functions used are :

- i) Sigmoid
- ii) Tanh
- iii) ReLU

A typical Convolutional Neural Network Looks as follows:

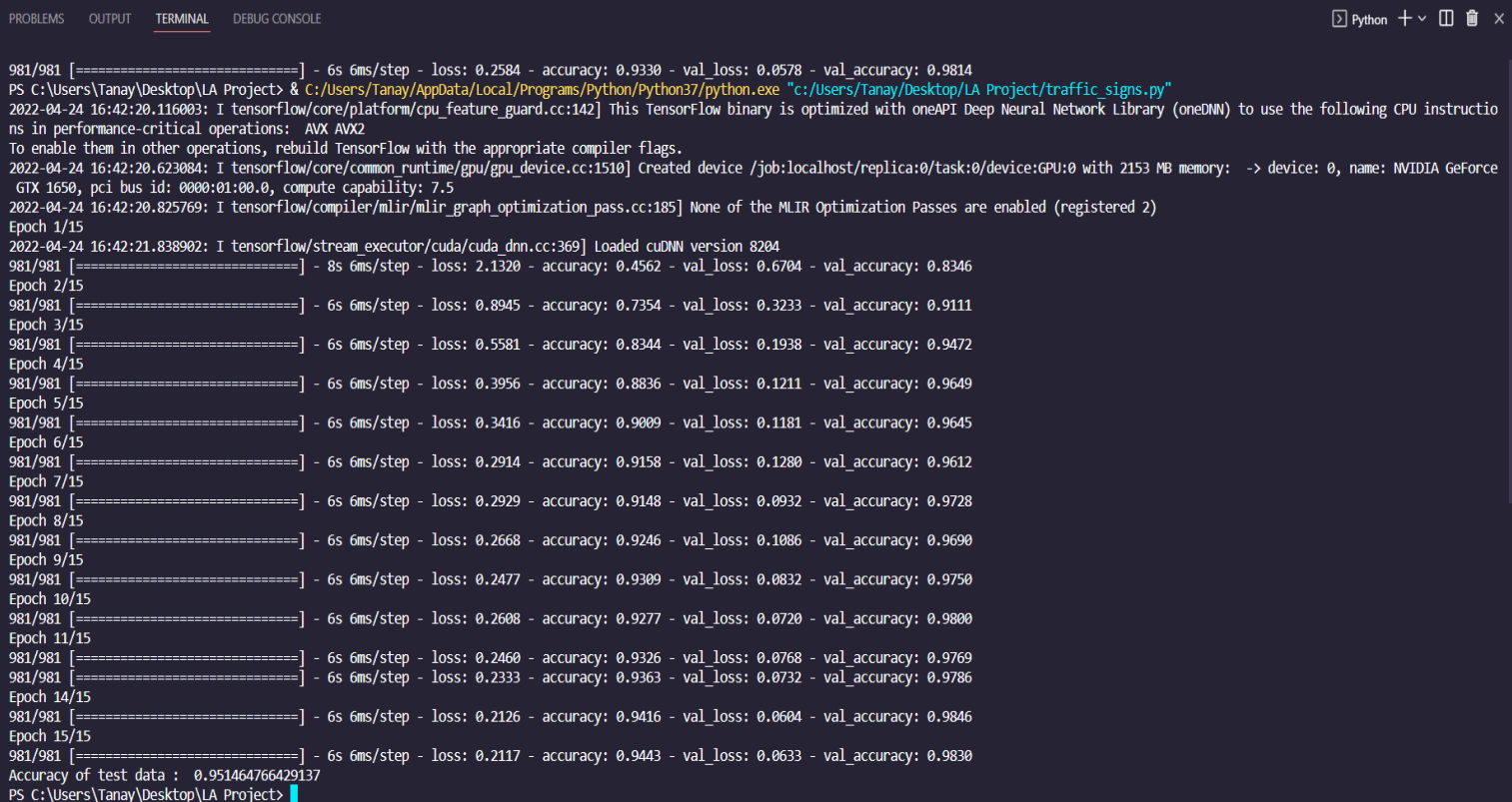


Libraries Used:

- i) NumPy
- ii) Pandas
- iii) Matplotlib
- iv) TensorFlow
- v) Pillow (Python Library to handle Images)
- vi) Scikit Learn
- vii) Keras

Observation:

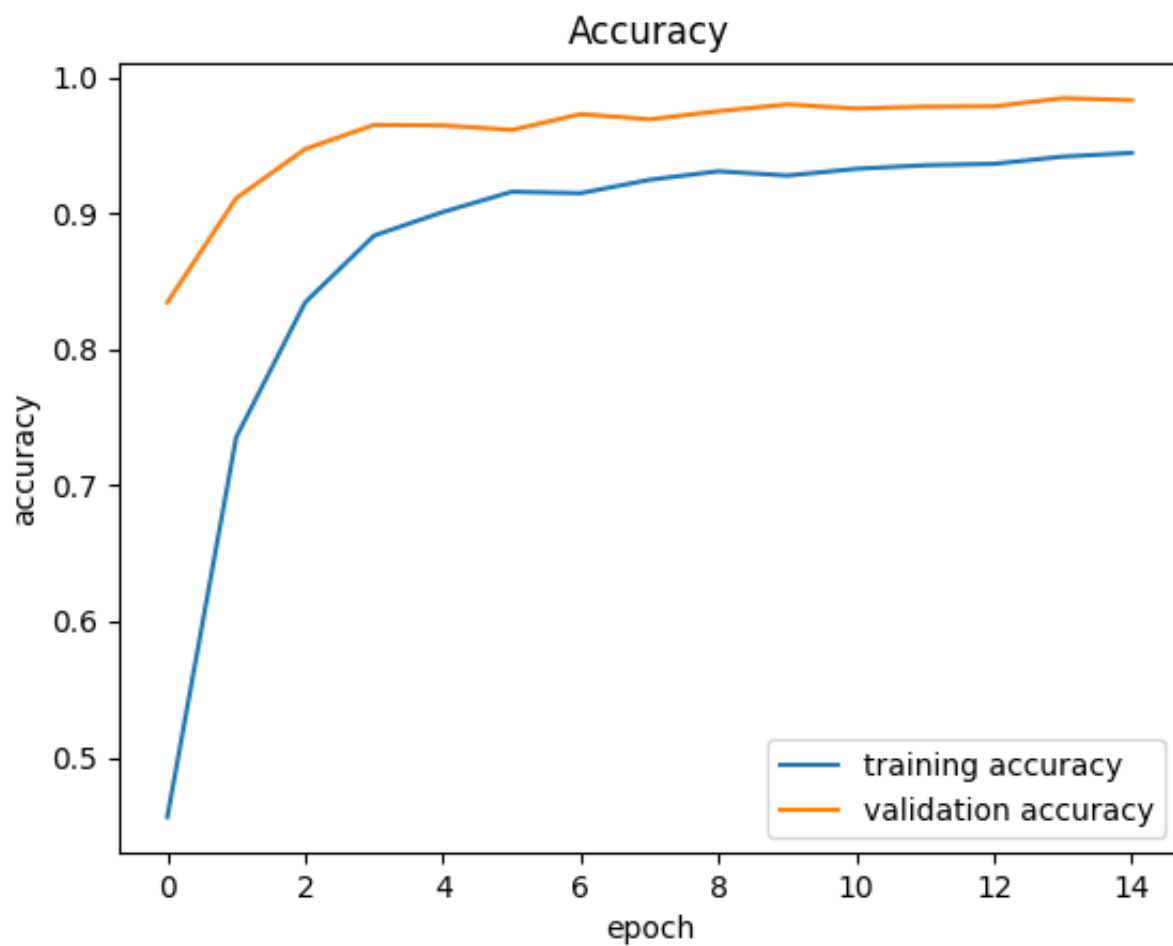
After training the model for 15 epochs the following results were achieved:



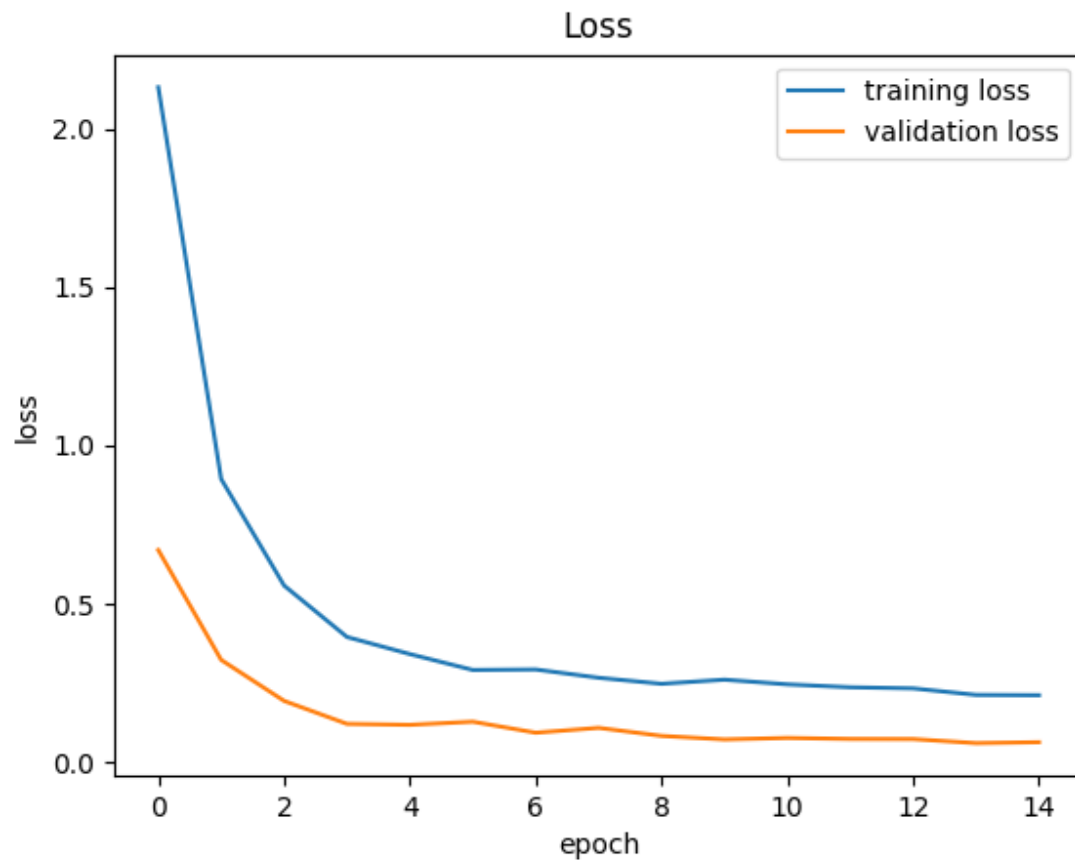
```
981/981 [=====] - 6s 6ms/step - loss: 0.2584 - accuracy: 0.9330 - val_loss: 0.0578 - val_accuracy: 0.9814
PS C:\Users\Tanay\Desktop\LA Project> & C:/Users/Tanay/AppData/Local/Programs/Python/Python37/python.exe "c:/Users/Tanay/Desktop/LA Project/traffic_signs.py"
2022-04-24 16:42:20.116003: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2022-04-24 16:42:20.623084: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 2153 MB memory: -> device: 0, name: NVIDIA GeForce GTX 1650, pci bus id: 0000:01:00.0, compute capability: 7.5
2022-04-24 16:42:20.825769: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimization Passes are enabled (registered 2)
Epoch 1/15
2022-04-24 16:42:21.838902: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loaded cuDNN version 8204
981/981 [=====] - 8s 6ms/step - loss: 2.1320 - accuracy: 0.4562 - val_loss: 0.6704 - val_accuracy: 0.8346
Epoch 2/15
981/981 [=====] - 6s 6ms/step - loss: 0.8945 - accuracy: 0.7354 - val_loss: 0.3233 - val_accuracy: 0.9111
Epoch 3/15
981/981 [=====] - 6s 6ms/step - loss: 0.5581 - accuracy: 0.8344 - val_loss: 0.1938 - val_accuracy: 0.9472
Epoch 4/15
981/981 [=====] - 6s 6ms/step - loss: 0.3956 - accuracy: 0.8836 - val_loss: 0.1211 - val_accuracy: 0.9649
Epoch 5/15
981/981 [=====] - 6s 6ms/step - loss: 0.3416 - accuracy: 0.9009 - val_loss: 0.1181 - val_accuracy: 0.9645
Epoch 6/15
981/981 [=====] - 6s 6ms/step - loss: 0.2914 - accuracy: 0.9158 - val_loss: 0.1280 - val_accuracy: 0.9612
Epoch 7/15
981/981 [=====] - 6s 6ms/step - loss: 0.2929 - accuracy: 0.9148 - val_loss: 0.0932 - val_accuracy: 0.9728
Epoch 8/15
981/981 [=====] - 6s 6ms/step - loss: 0.2668 - accuracy: 0.9246 - val_loss: 0.1086 - val_accuracy: 0.9690
Epoch 9/15
981/981 [=====] - 6s 6ms/step - loss: 0.2477 - accuracy: 0.9309 - val_loss: 0.0832 - val_accuracy: 0.9750
Epoch 10/15
981/981 [=====] - 6s 6ms/step - loss: 0.2608 - accuracy: 0.9277 - val_loss: 0.0720 - val_accuracy: 0.9800
Epoch 11/15
981/981 [=====] - 6s 6ms/step - loss: 0.2460 - accuracy: 0.9326 - val_loss: 0.0768 - val_accuracy: 0.9769
981/981 [=====] - 6s 6ms/step - loss: 0.2333 - accuracy: 0.9363 - val_loss: 0.0732 - val_accuracy: 0.9786
Epoch 14/15
981/981 [=====] - 6s 6ms/step - loss: 0.2126 - accuracy: 0.9416 - val_loss: 0.0604 - val_accuracy: 0.9846
Epoch 15/15
981/981 [=====] - 6s 6ms/step - loss: 0.2117 - accuracy: 0.9443 - val_loss: 0.0633 - val_accuracy: 0.9830
Accuracy of test data : 0.951464766429137
PS C:\Users\Tanay\Desktop\LA Project>
```

Accuracy on the test data was : 95.14647%

The Accuracy Graph is as follows:



The Training Loss Graph is as follows:



Applications:

- i) This project can be used in many different higher-level projects such as in the development of delivery vehicles, autonomous vehicles etc. where the recognition of traffic road signs are very important.

References:

- i) Dataset:
<https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign>
- ii) Theory on Convolutional Neural Networks:
 - a) <https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939#:~:text=A%20Convolutional%20Neural%20Network%2C%20also,binary%20representation%20of%20visual%20data.>
 - b) Stanford University's Course — CS231n: Convolutional Neural Network for Visual Recognition by Prof. Fei-Fei Li, Justin Johnson, Serena Yeung
 - c) <https://cs231n.github.io/convolutional-networks/>