REAL-TIME TRAFFIC SIGN RECOGNITION AND CLASSIFICATION

In today's dynamic transportation landscape, ensuring road safety and efficient traffic management is paramount. Real-Time Traffic Sign Recognition and Classification (RTTSRC) systems offer vital solutions by harnessing computer vision and machine learning techniques to swiftly detect and interpret traffic signs.

Our project aims to develop an RTTSRC system capable of accurately identifying and classifying traffic signs in real-time scenarios. By leveraging computer vision and the power of deep learning models, we aim to revolutionize road safety and traffic flow efficiency.

This report entails the methodologies, dataset, algorithms, models, evaluation metrics and the future work for the system. Our goal is to contribute to the creation of smarter and safer transportation systems, where vehicles autonomously interpret and respond to traffic signs, enhancing overall road safety and efficiency.

Highlighting the Real World Problem

In the real world, traffic signs play a pivotal role in guiding and regulating vehicular movement, ensuring road safety, and minimizing traffic-related incidents. However, manually detecting and interpreting these signs in varying environmental conditions poses significant challenges for both human drivers and autonomous vehicles. Our deep learning model addresses this critical real-world problem by automating the process of traffic sign detection and classification from images captured by onboard cameras. By accurately identifying and interpreting traffic signs in real-time, our model enhances the capabilities of autonomous vehicles to navigate complex road networks safely and efficiently. This solution not only reduces the burden on human drivers but also lays the groundwork for the widespread adoption of autonomous driving technology, ultimately contributing to the advancement of intelligent transportation systems and the promotion of road safety worldwide

About the Dataset

The dataset we have chosen to work with is the “Real_Time_Traffic_Signs_Dataset”. There are 55 classes of distinct traffic signs in the Traffic Sign Dataset v2, which are divided into four primary groups: forb, info, mand, and warn. These groups stand for prohibited, informational, required, and warning signs, respectively. Every image in the collection has 640 x 640 pixels of natural size, and all labels have been carefully checked to correct any labeling problems. The dataset reflects real-world frequencies with an unbalanced class distribution, but attempts to preserve label integrity are ongoing, with updates and improvements including the recent addition of 2,000 new photos and label double-checking in version 2 helping to preserve label integrity. In order to further enhance the dataset, future iterations plan to include more classes and make use of the already-existing unlabeled traffic signs.

Novelty

The fusion model is unusual because it combines two separate architectures, ResNet34 and Vision Transformer (ViT), which are tailored to various aspects of representation learning and image processing. The fusion model leverages the complementary capabilities of two models to produce improved performance in tasks involving the recognition of traffic signs.

With its deep residual blocks, the ResNet34 model—which is well-known for its efficacy in picture classification—provides strong feature extraction capabilities. Conversely, the Vision Transformer model presents a revolutionary method of image processing by utilizing self-attention processes to efficiently extract global context information and long-range relationships.

By use of fusion, the model utilizes the global contextual understanding offered by the Vision Transformer and the hierarchical features acquired by ResNet34, thereby enhancing the representation space for images of traffic signs. This fusion improves recognition accuracy and robustness in a variety of settings by allowing the model to collect both local and global properties of traffic signs.

All things considered, the fusion model is a novel method to traffic sign identification that makes use of the advantages of several different architectures to provide better performance and versatility in practical applications.

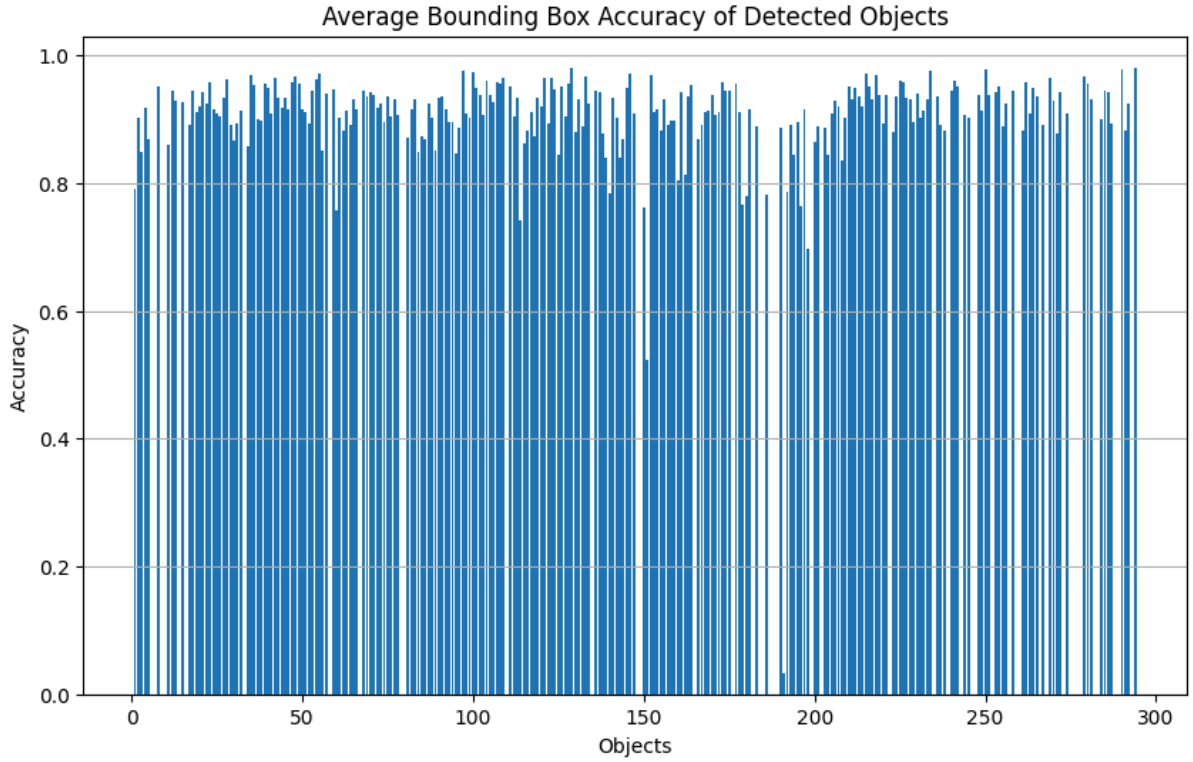
Models Used

Our experiment focuses on object detection in the Traffic Sign Dataset v2, using the pretrained model YOLO-V8. YOLO-V8's sophisticated architecture allows it to provide strong real-time traffic sign detection and classification skills, which fit in well with our dataset's extensive 55-class collection. We accelerate the development process and achieve great accuracy and efficiency in detecting different types and sizes of traffic signs within our standardized 640 x 640 pixel images by utilizing the pretrained weights of the model. Our detection pipeline's dependability and efficacy are continuously verified against ground truth labels, enabling us to make a significant contribution to the fields of traffic sign identification and safety optimization in transportation systems.

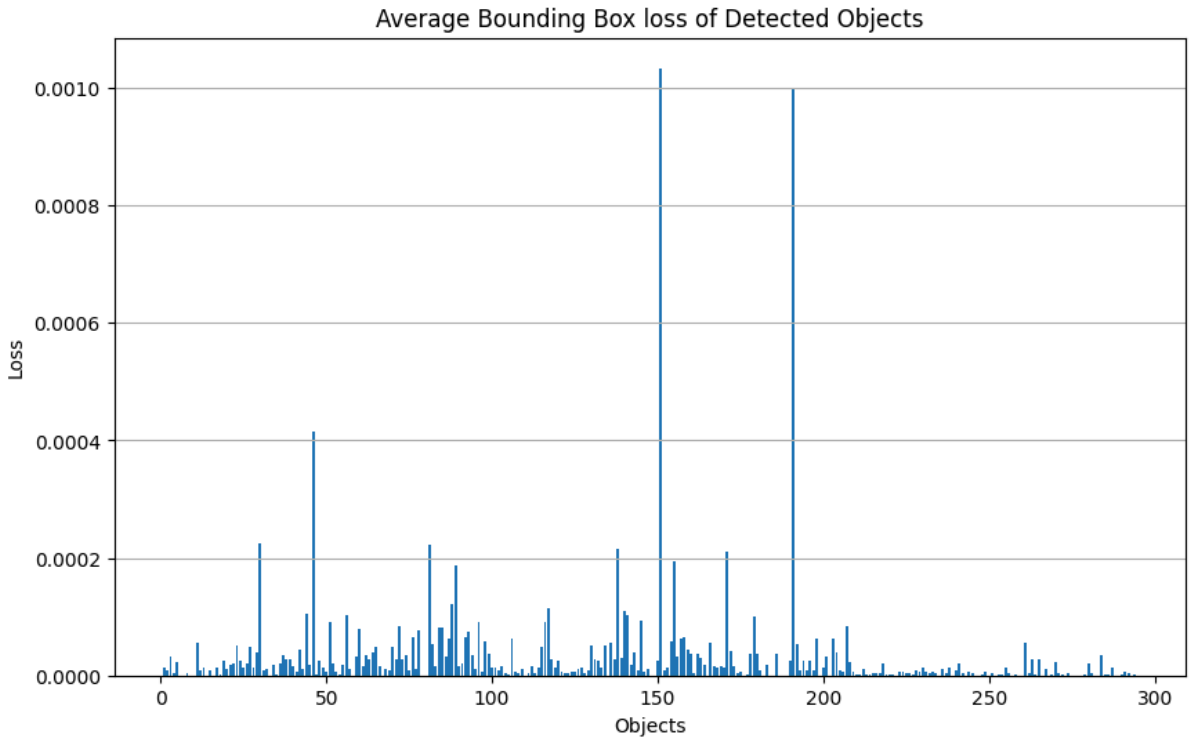
YOLO V8 (Object Detection)

The pretrained model, YOLO-V8, is a state-of-the-art convolutional neural network architecture for real-time object detection tasks. It builds on previous versions, YOLO-V8 incorporates advances in network depth, feature extraction, and context aggregation, leading to better detection efficiency and accuracy. The model learns rich representations of object features through pretraining on a large-scale dataset, which allows it to generalize well across different domains. The pretrained weights capture knowledge from extensive training on large datasets, allowing for quick adaptation to specific object detection tasks with little need for fine-tuning. YOLO-V8's architecture prioritizes speed without sacrificing detection quality, making it a promising candidate for various applications.

From the below image we can interpret the accuracy for each detected object from the YOLO model.



This image tell us about the average bounding box loss for each objects.



Below are the cropped images based on the output from the YOLO model.

Data Preprocessing to Train Classification Models

The traffic sign dataset is preprocessed through a series of procedures in order to make it ready for model training. To ensure that each cropped image only includes the pertinent traffic sign, photographs are first cropped based on annotations supplied in YOLOv8 format. Reading the annotations, figuring out the bounding box coordinates, and removing the area of interest (ROI) from the original picture are the steps in this method. Following cropping, photos are stored with the appropriate class designations.

Afterwards, the train and validation datasets are merged into a single folder structure to make training easier. In order to do this merging operation, pictures from each class are copied to the appropriate class directory in the combined dataset folder by iterating over the class directories in the train and validation folders.

Lastly, the class directories inside the combined dataset folder are renamed using two-digit numerical labels to guarantee uniform directory naming rules. In this stage, photos are moved from the old class directories to the newly formed directories with standardized labels. A new directory structure is also created.

All things considered, this preparation pipeline simplifies the organization of the dataset and gets it ready for later model training, making sure that photos are correctly cropped and labeled for efficient learning and traffic sign categorization.

Below are the images after preprocessing with their labels

VGG-13:

A sequence of convolutional layers, max-pooling procedures for feature extraction, and fully connected layers for classification make up the VGG13 model. To maintain spatial dimensions, 3x3 kernels with padding are used in the convolutional layers. Four blocks make up the feature extraction pipeline; each block consists of two convolutional layers activated by ReLU, followed by max-pooling. From 64 to 512 output channels, the number gradually rises as max-pooling causes the spatial dimensions to contract. The final output layer, which has a size equal to the number of classes (55 in this case), is reached by gradually decreasing the feature dimensions through the use of dropout regularization and ReLU activation in three linear layers that make up the fully connected classifier. The model is versatile, accommodating input images of varying sizes, with the convolutional layers adapting accordingly.

ResNet-34:

The ResNet34 model is composed of four residual blocks, each of which contains multiple residual units (ResidualBlock). The first convolutional block is a 7x7 convolutional layer with 64 output channels, batch normalization, ReLU activation, and max-pooling. The subsequent residual blocks are built using the _make_layer method, which generates a series of residual units based on the input/output channel dimensions and the number of blocks that are specified. Each residual unit is made up of two 3x3 convolutional layers that have batch normalization and ReLU activation, which helps with the learning of residual mappings. The output of the last residual block is subjected to adaptive average pooling, which produces a fixed-size feature map that is then flattened and fed into a fully connected layer for classification. The model's architecture enables effective feature extraction and representation learning, making it suitable for a variety of image classification tasks, including the prediction of traffic sign classes in this scenario.

Vision Transformer:

Using the transformer architecture, the Vision Transformer model is made to analyze images for categorization tasks. The process starts with a patch embedding layer that divides the input image into patches. A convolutional layer is then used to project each patch into an embedding space. Batch normalization and flattening operations are then performed. After that, the embedded patches are sent into a transformer encoder, which is made up of several layers. Positional encodings are used to preserve spatial information, and multi-head self-attention methods are applied by each encoder layer to capture global dependencies. Mean pooling across sequence dimensions is used to aggregate the resultant embeddings, and a linear layer then generates the final output logits for classification. For a variety of visual identification applications, including the categorization of traffic signs, this architecture provides a strong substitute for conventional convolutional neural networks by enabling efficient representation learning from image data.

ViTResNet34 Fusion Model:

A ResNet34 model and a Vision Transformer (ViT) model, each designed for a distinct traffic sign identification task, make up the combined architecture. With its fully connected layer for classification and a sequence of residual blocks for efficient feature extraction, the ResNet34 model excels in image classification tasks. In contrast, the Vision Transformer model is very good at identifying global relationships in images by using self-attention mechanisms, which are then followed by positional encoding and embedding aggregation. An embedding space representation is created by integrating both models, which makes it easier to do subsequent operations like multi-modal fusion and similarity search. With this hybrid architecture, comprehensive traffic sign identification and understanding is achieved by utilizing the complementing strengths of transformer-based and convolutional techniques.

RESULTS OF TRAINING

i) VGG-13 MODEL

Summary of the Model

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 40, 40]           1,792
              ReLU-2          [-1, 64, 40, 40]               0
            Conv2d-3          [-1, 64, 40, 40]          36,928
              ReLU-4          [-1, 64, 40, 40]               0
         MaxPool2d-5          [-1, 64, 20, 20]               0
            Conv2d-6         [-1, 128, 20, 20]          73,856
              ReLU-7         [-1, 128, 20, 20]               0
            Conv2d-8         [-1, 128, 20, 20]         147,584
              ReLU-9         [-1, 128, 20, 20]               0
        MaxPool2d-10         [-1, 128, 10, 10]               0
           Conv2d-11         [-1, 256, 10, 10]         295,168
             ReLU-12         [-1, 256, 10, 10]               0
           Conv2d-13         [-1, 256, 10, 10]         590,080
             ReLU-14         [-1, 256, 10, 10]               0
        MaxPool2d-15           [-1, 256, 5, 5]               0
           Conv2d-16           [-1, 512, 5, 5]       1,180,160
             ReLU-17           [-1, 512, 5, 5]               0
           Conv2d-18           [-1, 512, 5, 5]       2,359,808
             ReLU-19           [-1, 512, 5, 5]               0
        MaxPool2d-20           [-1, 512, 2, 2]               0
           Conv2d-21           [-1, 512, 2, 2]       2,359,808
             ReLU-22           [-1, 512, 2, 2]               0
           Conv2d-23           [-1, 512, 2, 2]       2,359,808
             ReLU-24           [-1, 512, 2, 2]               0
        MaxPool2d-25           [-1, 512, 1, 1]               0
           Linear-26                 [-1, 4096]       2,101,248
             ReLU-27                 [-1, 4096]               0
          Dropout-28                 [-1, 4096]               0
           Linear-29                 [-1, 4096]      16,781,312
             ReLU-30                 [-1, 4096]               0
          Dropout-31                 [-1, 4096]               0
           Linear-32                   [-1, 55]         225,335
================================================================
Total params: 28,512,887
Trainable params: 28,512,887
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.02
Forward/backward pass size (MB): 6.47
Params size (MB): 108.77
Estimated Total Size (MB): 115.26
----------------------------------------------------------------
```

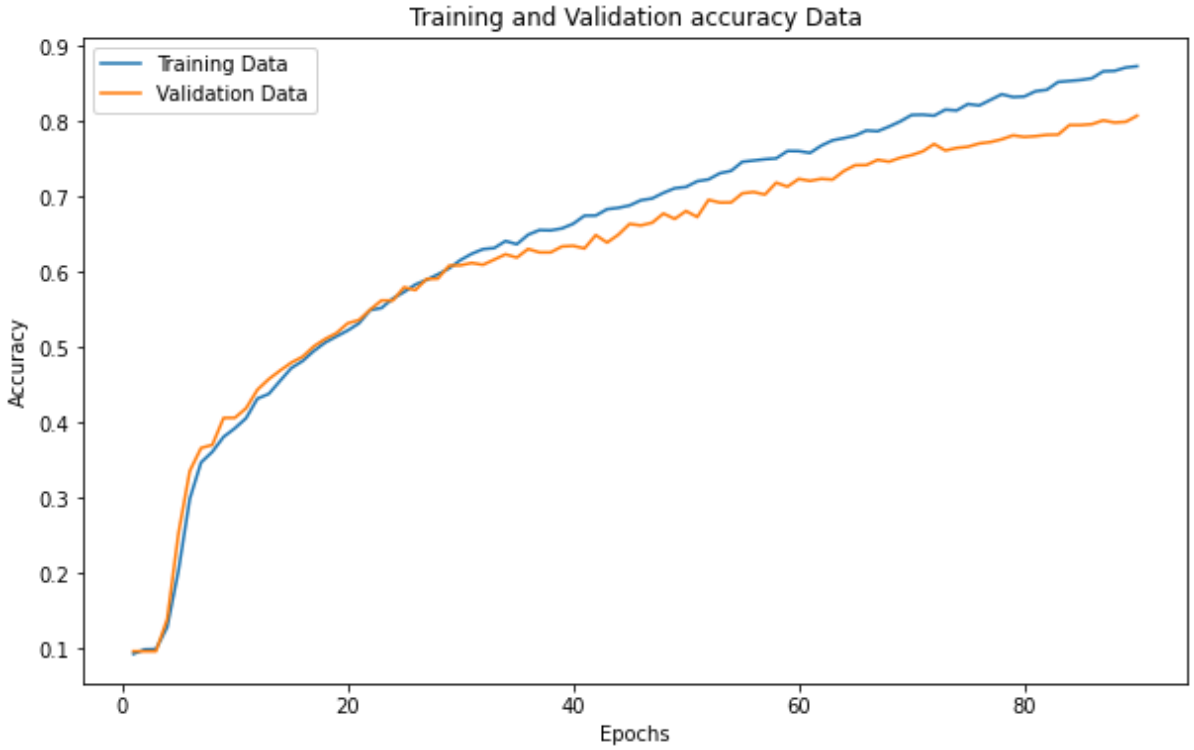
After training the model for 90 epochs, the training results and performance metrics are as follows.

Training Summary & Performance Metrics

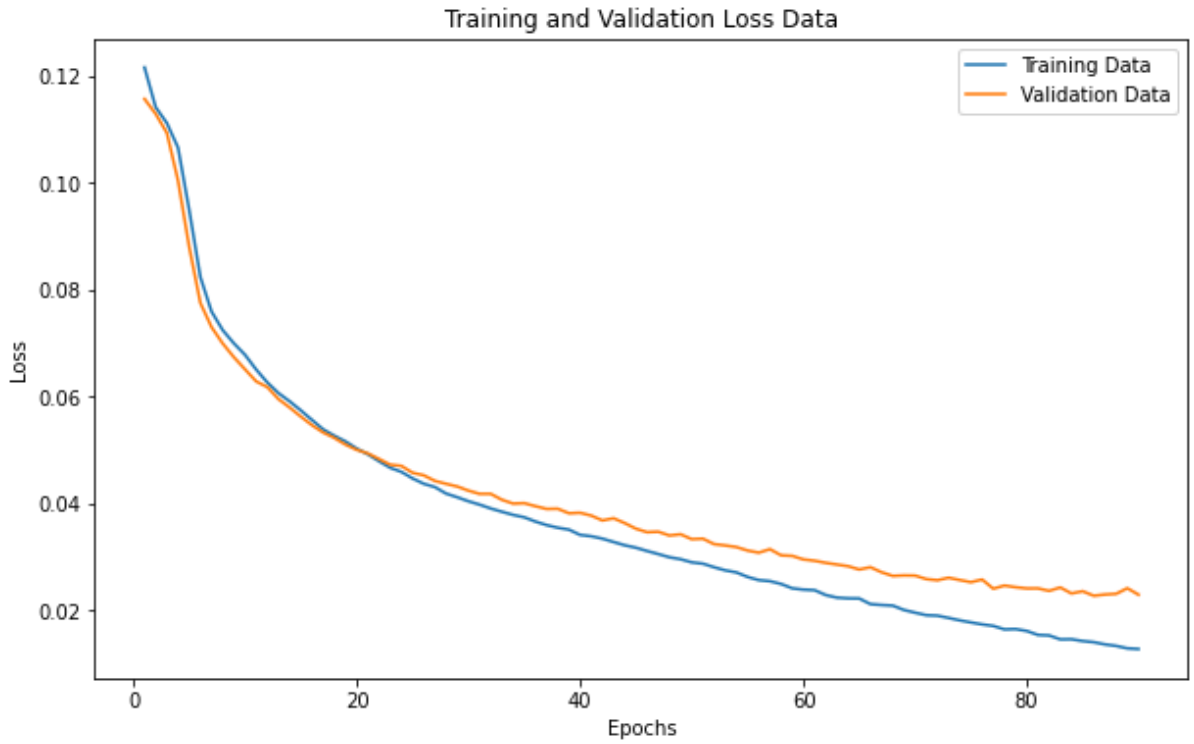
```
Accuracy and Loss values:

Training Accuracy : 87.35 %
Validation Accuracy : 80.77 %
Testing Accuracy : 81.04 %

Training Loss : 0.037866791109287326
Validation Loss : 0.04136828962689302
Testing Loss : 0.020754993922411 36
```

```
Performance Metrics:
Time to Train : 4.581313371658325 seconds
Accuracy      : 81.01 %
Precision     : [0.96875    0.92592593 0.96666667 0.87878788 0.66666667 0.2
 0.33333333 0.33333333 0.7037037  0.6        0.83333333 0.5
 0.13333333 0.93333333 0.28571429 0.         0.94230769 0.4
 0.69230769 0.73913043 0.875      0.96       0.97196262 0.92307692
 0.78125    0.94736842 0.8        0.33333333 0.88888889 0.66666667
 0.625      1.         0.83783784 0.87878788 0.97142857 0.9
 1.         0.86666667 0.86486486 0.94736842 0.83870968 0.34782609
 0.83870968 0.78947368 0.625      0.71428571 0.66666667 0.63636364
 0.         0.6875     0.875      0.77777778 0.         0.54545455]
Recall        : [0.88571429 0.92592593 0.90625    0.87878788 0.83333333 0.25
 0.25       0.33333333 0.7037037  0.9375     0.90909091 0.22727273
 0.22222222 0.7        0.22222222 0.         1.         0.5
 0.81818182 0.65384615 0.93333333 0.88888889 0.93693694 0.92307692
 0.92592593 0.92307692 0.8        0.66666667 1.         0.5
 0.83333333 0.79310345 0.86111111 0.90625    0.97142857 0.75
 0.66666667 0.86666667 0.86486486 1.         0.94545455 0.72727273
 0.92857143 0.71428571 0.625      0.76923077 0.625      0.46666667
 0.         0.64705882 1.         0.63636364 0.         1.        ]
F1 Score      : [0.92537313 0.92592593 0.93548387 0.87878788 0.74074074 0.22222222
 0.28571429 0.33333333 0.7037037  0.73170732 0.86956522 0.3125
 0.16666667 0.8        0.25       0.         0.97029703 0.44444444
 0.75       0.69387755 0.90322581 0.92307692 0.95412844 0.92307692
 0.84745763 0.93506494 0.8        0.44444444 0.94117647 0.57142857
 0.71428571 0.88461538 0.84931507 0.89230769 0.97142857 0.81818182
 0.8        0.86666667 0.86486486 0.97297297 0.88888889 0.47058824
 0.88135593 0.75       0.625      0.74074074 0.64516129 0.53846154
 0.         0.66666667 0.93333333 0.7        0.         0.70588235]
```

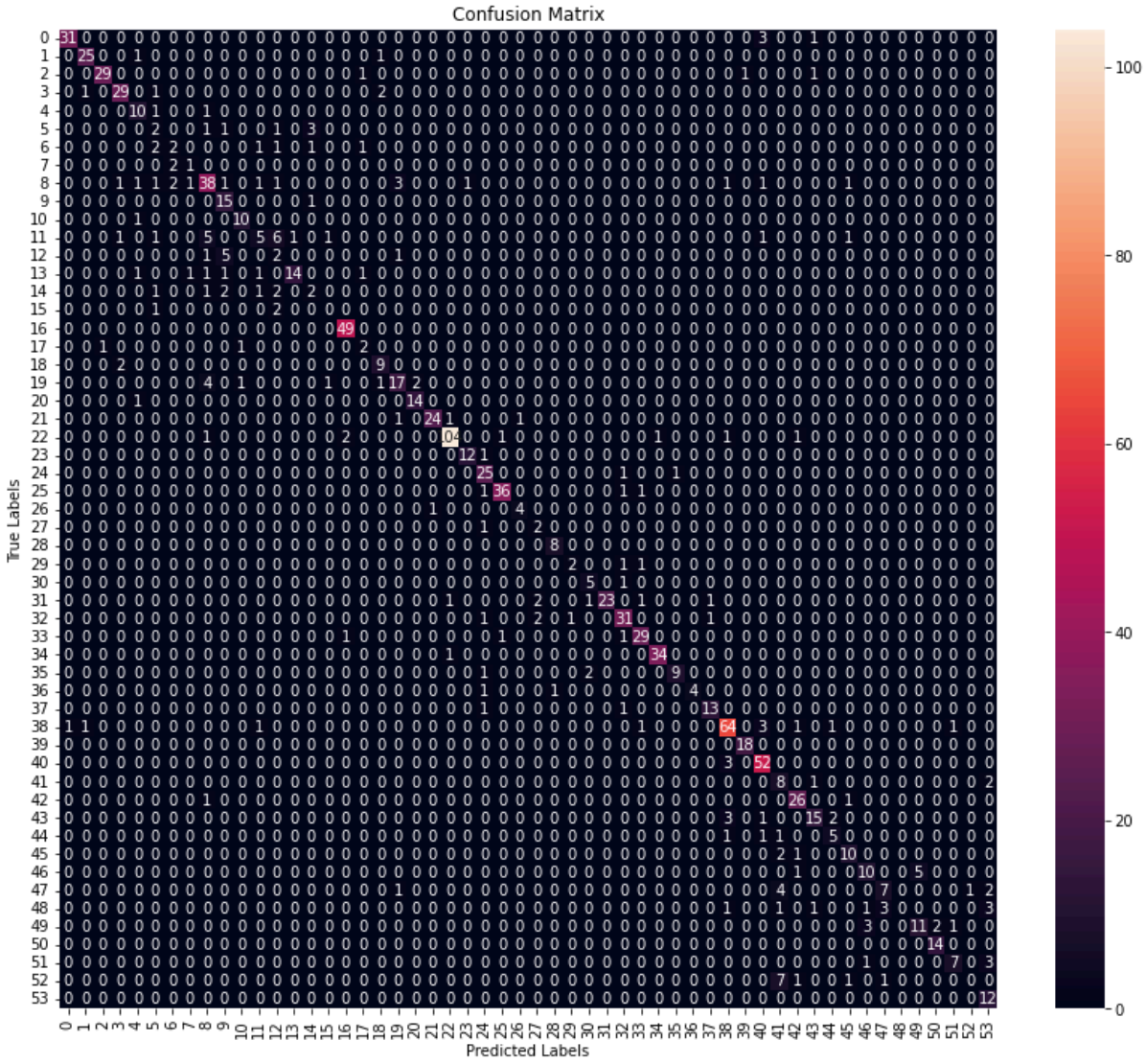
Visualization of Training & Validation Accuracy over Epochs



Visualization of Training & Validation Loss over Epochs



Confusion Matrix Generated from Testing the Model



ii) ResNet-34 Model

Summary of the Model

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 20, 20]	9,408
BatchNorm2d-2	[-1, 64, 20, 20]	128
ReLU-3	[-1, 64, 20, 20]	0
MaxPool2d-4	[-1, 64, 10, 10]	0
Conv2d-5	[-1, 64, 10, 10]	36,864
BatchNorm2d-6	[-1, 64, 10, 10]	128
ReLU-7	[-1, 64, 10, 10]	0
Conv2d-8	[-1, 64, 10, 10]	36,864
BatchNorm2d-9	[-1, 64, 10, 10]	128
ReLU-10	[-1, 64, 10, 10]	0
ResidualBlock-11	[-1, 64, 10, 10]	0
Conv2d-12	[-1, 64, 10, 10]	36,864
BatchNorm2d-13	[-1, 64, 10, 10]	128
ReLU-14	[-1, 64, 10, 10]	0
Conv2d-15	[-1, 64, 10, 10]	36,864
BatchNorm2d-16	[-1, 64, 10, 10]	128
ReLU-17	[-1, 64, 10, 10]	0
ResidualBlock-18	[-1, 64, 10, 10]	0
Conv2d-19	[-1, 64, 10, 10]	36,864
BatchNorm2d-20	[-1, 64, 10, 10]	128
ReLU-21	[-1, 64, 10, 10]	0
Conv2d-22	[-1, 64, 10, 10]	36,864
BatchNorm2d-23	[-1, 64, 10, 10]	128
ReLU-24	[-1, 64, 10, 10]	0
ResidualBlock-25	[-1, 64, 10, 10]	0
Conv2d-26	[-1, 128, 5, 5]	73,728
BatchNorm2d-27	[-1, 128, 5, 5]	256
ReLU-28	[-1, 128, 5, 5]	0
Conv2d-29	[-1, 128, 5, 5]	147,456
BatchNorm2d-30	[-1, 128, 5, 5]	256
Conv2d-31	[-1, 128, 5, 5]	8,192
BatchNorm2d-32	[-1, 128, 5, 5]	256
ReLU-33	[-1, 128, 5, 5]	0
ResidualBlock-34	[-1, 128, 5, 5]	0
Conv2d-35	[-1, 128, 5, 5]	147,456
BatchNorm2d-36	[-1, 128, 5, 5]	256
ReLU-37	[-1, 128, 5, 5]	0
Conv2d-38	[-1, 128, 5, 5]	147,456
BatchNorm2d-39	[-1, 128, 5, 5]	256
ReLU-40	[-1, 128, 5, 5]	0
ResidualBlock-41	[-1, 128, 5, 5]	0
Conv2d-42	[-1, 128, 5, 5]	147,456
BatchNorm2d-43	[-1, 128, 5, 5]	256
ReLU-44	[-1, 128, 5, 5]	0
Conv2d-45	[-1, 128, 5, 5]	147,456
BatchNorm2d-46	[-1, 128, 5, 5]	256
ReLU-47	[-1, 128, 5, 5]	0
ResidualBlock-48	[-1, 128, 5, 5]	0
Conv2d-49	[-1, 128, 5, 5]	147,456
BatchNorm2d-50	[-1, 128, 5, 5]	256
ReLU-51	[-1, 128, 5, 5]	0
Conv2d-52	[-1, 128, 5, 5]	147,456
BatchNorm2d-53	[-1, 128, 5, 5]	256
ReLU-54	[-1, 128, 5, 5]	0
ResidualBlock-55	[-1, 128, 5, 5]	0
Conv2d-56	[-1, 256, 3, 3]	294,912
BatchNorm2d-57	[-1, 256, 3, 3]	512
ReLU-58	[-1, 256, 3, 3]	0
Conv2d-59	[-1, 256, 3, 3]	589,824
BatchNorm2d-60	[-1, 256, 3, 3]	512
Conv2d-61	[-1, 256, 3, 3]	32,768
BatchNorm2d-62	[-1, 256, 3, 3]	512
ReLU-63	[-1, 256, 3, 3]	0
ResidualBlock-64	[-1, 256, 3, 3]	0
Conv2d-65	[-1, 256, 3, 3]	589,824
BatchNorm2d-66	[-1, 256, 3, 3]	512
ReLU-67	[-1, 256, 3, 3]	0
Conv2d-68	[-1, 256, 3, 3]	589,824
BatchNorm2d-69	[-1, 256, 3, 3]	512

```
        Conv2d-68          [-1, 256, 3, 3]         589,824
   BatchNorm2d-69          [-1, 256, 3, 3]             512
          ReLU-70          [-1, 256, 3, 3]               0
  ResidualBlock-71          [-1, 256, 3, 3]               0
        Conv2d-72          [-1, 256, 3, 3]         589,824
   BatchNorm2d-73          [-1, 256, 3, 3]             512
          ReLU-74          [-1, 256, 3, 3]               0
        Conv2d-75          [-1, 256, 3, 3]         589,824
   BatchNorm2d-76          [-1, 256, 3, 3]             512
          ReLU-77          [-1, 256, 3, 3]               0
  ResidualBlock-78          [-1, 256, 3, 3]               0
        Conv2d-79          [-1, 256, 3, 3]         589,824
   BatchNorm2d-80          [-1, 256, 3, 3]             512
          ReLU-81          [-1, 256, 3, 3]               0
        Conv2d-82          [-1, 256, 3, 3]         589,824
   BatchNorm2d-83          [-1, 256, 3, 3]             512
          ReLU-84          [-1, 256, 3, 3]               0
  ResidualBlock-85          [-1, 256, 3, 3]               0
        Conv2d-86          [-1, 256, 3, 3]         589,824
   BatchNorm2d-87          [-1, 256, 3, 3]             512
          ReLU-88          [-1, 256, 3, 3]               0
        Conv2d-89          [-1, 256, 3, 3]         589,824
   BatchNorm2d-90          [-1, 256, 3, 3]             512
          ReLU-91          [-1, 256, 3, 3]               0
  ResidualBlock-92          [-1, 256, 3, 3]               0
        Conv2d-93          [-1, 256, 3, 3]         589,824
   BatchNorm2d-94          [-1, 256, 3, 3]             512
          ReLU-95          [-1, 256, 3, 3]               0
        Conv2d-96          [-1, 256, 3, 3]         589,824
   BatchNorm2d-97          [-1, 256, 3, 3]             512
          ReLU-98          [-1, 256, 3, 3]               0
  ResidualBlock-99          [-1, 256, 3, 3]               0
       Conv2d-100          [-1, 512, 2, 2]       1,179,648
  BatchNorm2d-101          [-1, 512, 2, 2]           1,024
         ReLU-102          [-1, 512, 2, 2]               0
       Conv2d-103          [-1, 512, 2, 2]       2,359,296
  BatchNorm2d-104          [-1, 512, 2, 2]           1,024
       Conv2d-105          [-1, 512, 2, 2]         131,072
  BatchNorm2d-106          [-1, 512, 2, 2]           1,024
         ReLU-107          [-1, 512, 2, 2]               0
  ResidualBlock-108          [-1, 512, 2, 2]               0
       Conv2d-109          [-1, 512, 2, 2]       2,359,296
  BatchNorm2d-110          [-1, 512, 2, 2]           1,024
         ReLU-111          [-1, 512, 2, 2]               0
       Conv2d-112          [-1, 512, 2, 2]       2,359,296
  BatchNorm2d-113          [-1, 512, 2, 2]           1,024
         ReLU-114          [-1, 512, 2, 2]               0
  ResidualBlock-115          [-1, 512, 2, 2]               0
       Conv2d-116          [-1, 512, 2, 2]       2,359,296
  BatchNorm2d-117          [-1, 512, 2, 2]           1,024
         ReLU-118          [-1, 512, 2, 2]               0
       Conv2d-119          [-1, 512, 2, 2]       2,359,296
  BatchNorm2d-120          [-1, 512, 2, 2]           1,024
         ReLU-121          [-1, 512, 2, 2]               0
  ResidualBlock-122          [-1, 512, 2, 2]               0
AdaptiveAvgPool2d-123          [-1, 512, 1, 1]               0
       Linear-124                [-1, 55]          28,215
================================================================
Total params: 21,312,887
Trainable params: 21,312,887
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.02
Forward/backward pass size (MB): 3.53
Params size (MB): 81.30
Estimated Total Size (MB): 84.85
----------------------------------------------------------------
```

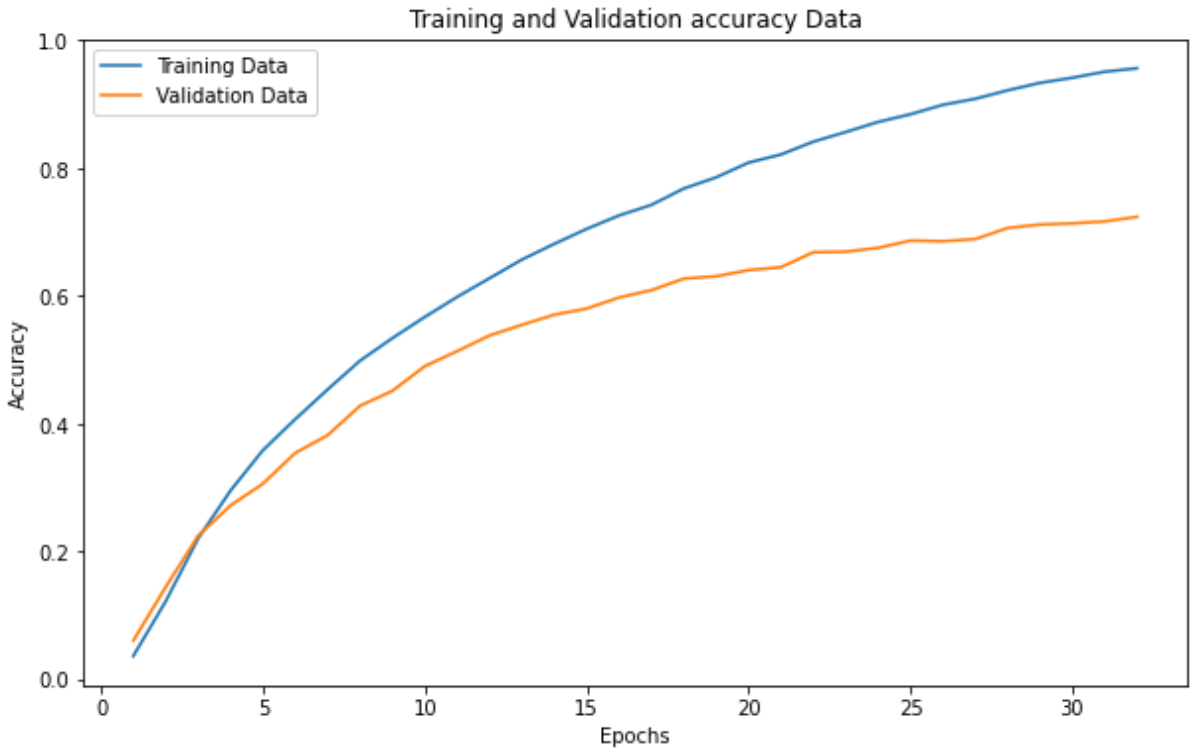
After training the model for 32 epochs, the training results and performance metrics are as follows.

Training Summary & Performance Metrics

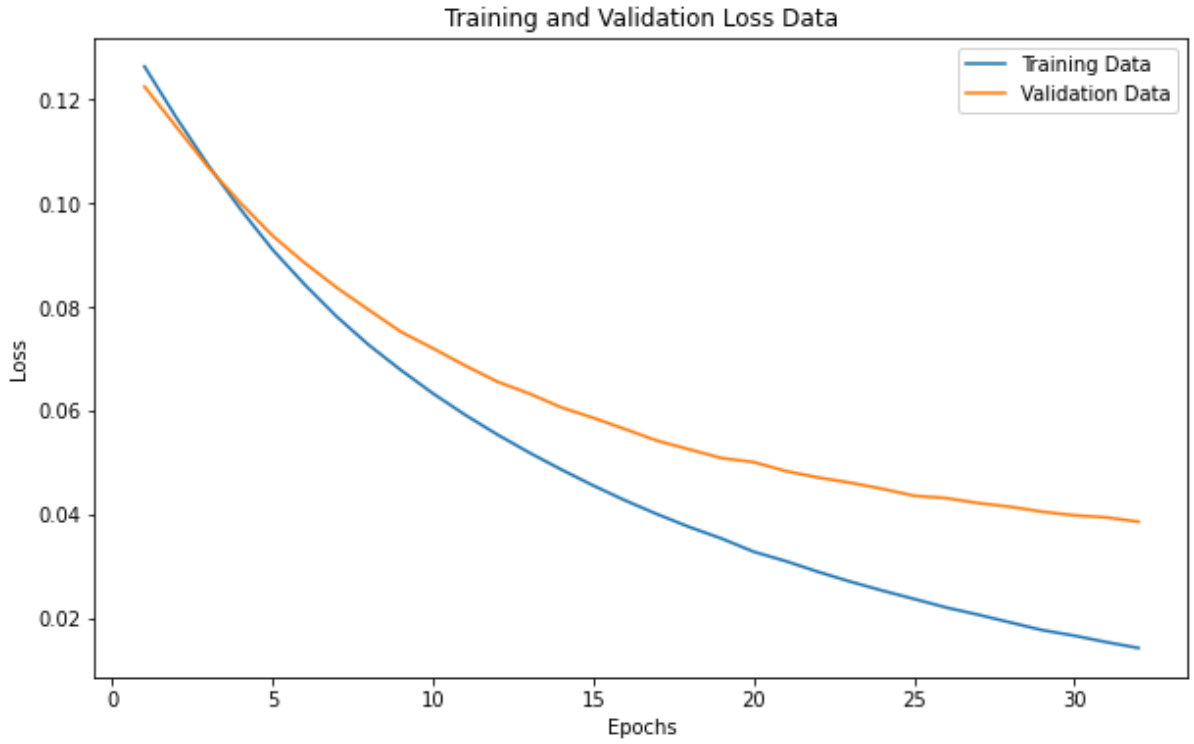
```
Accuracy and Loss values:

Training Accuracy : 95.67 %
Validation Accuracy : 72.41 %
Testing Accuracy : 72.59 %

Training Loss : 0.050468595702823756
Validation Loss : 0.06346135264637685
Testing Loss : 0.03670324556503562
```

```
Performance Metrics:
Time to Train : 4.149202585220337 seconds
Accuracy      : 72.56 %
Precision     : [0.91891892 0.76666667 0.75       0.81578947 1.         0.
 0.         0.         0.53164557 0.75       0.75       0.38461538
 0.36363636 0.71428571 0.         0.         0.92       0.
 1.         0.85714286 0.63636364 0.86206897 0.80451128 1.
 0.76666667 0.78947368 0.         0.         1.         0.
 0.8        0.85714286 0.64285714 0.875      0.88888889 1.
 1.         0.84615385 0.87179487 0.7826087  0.87096774 0.36363636
 0.80645161 0.2962963  0.         0.         0.71428571 0.57142857
 0.         0.5        0.5        0.         0.         0.29166667]
Recall        : [0.97142857 0.85185185 0.9375     0.93939394 0.08333333 0.
 0.         0.         0.77777778 0.5625     0.81818182 0.45454545
 0.44444444 0.25       0.         0.         0.93877551 0.
 0.36363636 0.69230769 0.93333333 0.92592593 0.96396396 0.92307692
 0.85185185 0.76923077 0.         0.         0.125      0.
 0.66666667 0.82758621 0.75       0.875      0.91428571 0.58333333
 0.16666667 0.73333333 0.91891892 1.         0.98181818 0.36363636
 0.89285714 0.76190476 0.         0.         0.3125     0.53333333
 0.         0.70588235 0.28571429 0.         0.         0.58333333]
F1 Score      : [0.94444444 0.80701754 0.83333333 0.87323944 0.15384615 0.
 0.         0.         0.63157895 0.64285714 0.7826087  0.41666667
 0.4        0.37037037 0.         0.         0.92929293 0.
 0.53333333 0.76595745 0.75675676 0.89285714 0.87704918 0.96
 0.80701754 0.77922078 0.         0.         0.22222222 0.
 0.72727273 0.84210526 0.69230769 0.875      0.90140845 0.73684211
 0.28571429 0.78571429 0.89473684 0.87804878 0.92307692 0.36363636
 0.84745763 0.42666667 0.         0.         0.43478261 0.55172414
 0.         0.58536585 0.36363636 0.         0.         0.38888889]
```

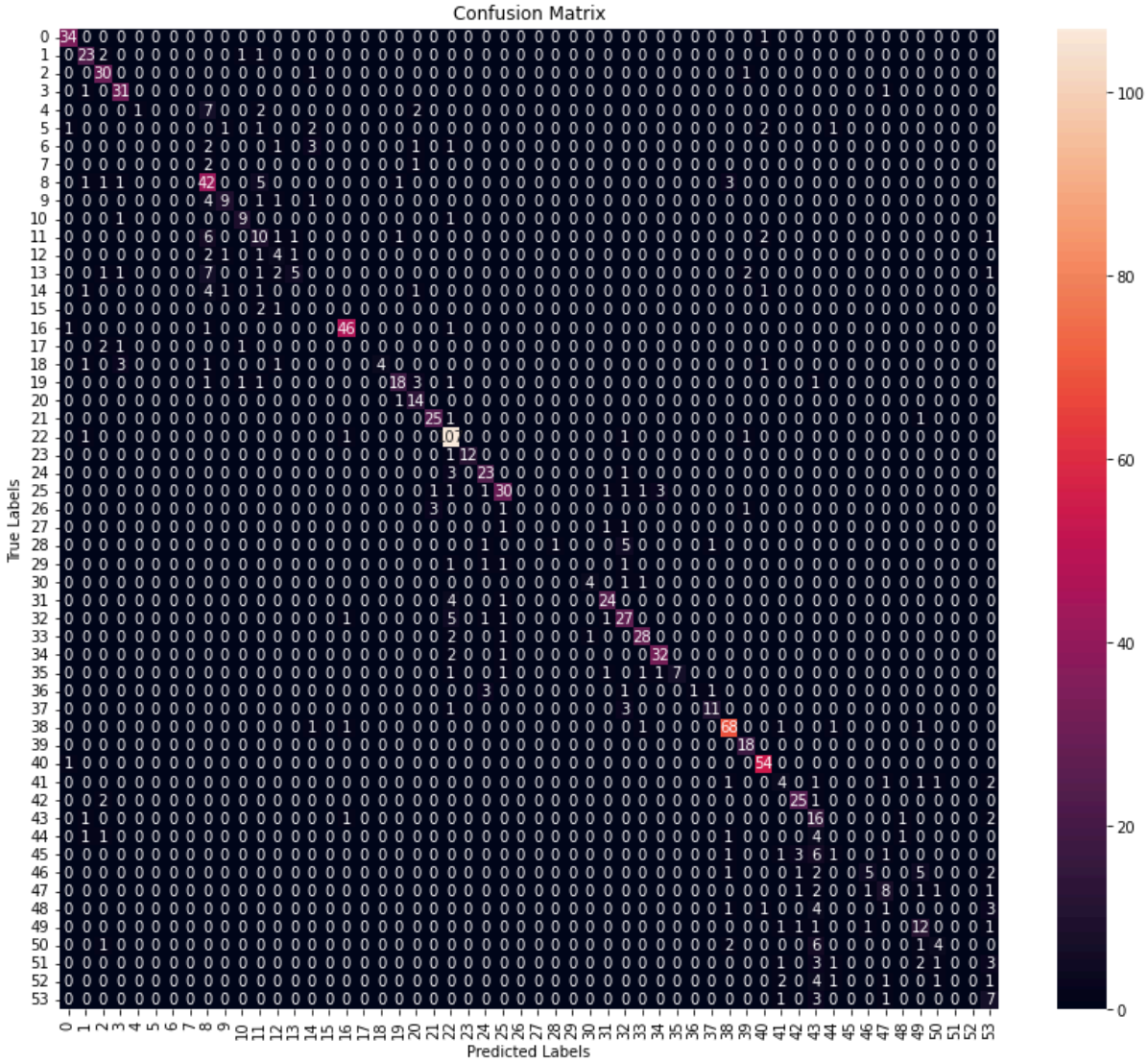
Visualization of Training & Validation Accuracy over Epochs



Visualization of Training & Validation Loss over Epochs



Confusion Matrix Generated from Testing the Model:



iii) Vision Image Transformer Model

Summary of the Model

```
VisionTransformer_Model(
  (patch_embedding): Sequential(
    (0): Conv2d(3, 169, kernel_size=(3, 3), stride=(3, 3))
    (1): BatchNorm2d(169, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): Flatten(start_dim=2, end_dim=-1)
  )
  (encoder): TransformerEncoder(
    (layers): ModuleList(
      (0): TransformerEncoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=169, out_features=169, bias=True)
        )
        (linear1): Linear(in_features=169, out_features=2048, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (linear2): Linear(in_features=2048, out_features=169, bias=True)
        (norm1): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
      )
      (1): TransformerEncoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=169, out_features=169, bias=True)
        )
        (linear1): Linear(in_features=169, out_features=2048, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (linear2): Linear(in_features=2048, out_features=169, bias=True)
        (norm1): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
      )
      (2): TransformerEncoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=169, out_features=169, bias=True)
        )
        (linear1): Linear(in_features=169, out_features=2048, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (linear2): Linear(in_features=2048, out_features=169, bias=True)
        (norm1): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
      )
      (3): TransformerEncoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=169, out_features=169, bias=True)
        )
        (linear1): Linear(in_features=169, out_features=2048, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (linear2): Linear(in_features=2048, out_features=169, bias=True)
        (norm1): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
      )
```

```
(4): TransformerEncoderLayer(
  (self_attn): MultiheadAttention(
    (out_proj): NonDynamicallyQuantizableLinear(in_features=169, out_features=169, bias=True)
  )
  (linear1): Linear(in_features=169, out_features=2048, bias=True)
  (dropout): Dropout(p=0.1, inplace=False)
  (linear2): Linear(in_features=2048, out_features=169, bias=True)
  (norm1): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
  (norm2): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
  (dropout1): Dropout(p=0.1, inplace=False)
  (dropout2): Dropout(p=0.1, inplace=False)
)
(5): TransformerEncoderLayer(
  (self_attn): MultiheadAttention(
    (out_proj): NonDynamicallyQuantizableLinear(in_features=169, out_features=169, bias=True)
  )
  (linear1): Linear(in_features=169, out_features=2048, bias=True)
  (dropout): Dropout(p=0.1, inplace=False)
  (linear2): Linear(in_features=2048, out_features=169, bias=True)
  (norm1): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
  (norm2): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
  (dropout1): Dropout(p=0.1, inplace=False)
  (dropout2): Dropout(p=0.1, inplace=False)
)
(6): TransformerEncoderLayer(
  (self_attn): MultiheadAttention(
    (out_proj): NonDynamicallyQuantizableLinear(in_features=169, out_features=169, bias=True)
  )
  (linear1): Linear(in_features=169, out_features=2048, bias=True)
  (dropout): Dropout(p=0.1, inplace=False)
  (linear2): Linear(in_features=2048, out_features=169, bias=True)
  (norm1): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
  (norm2): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
  (dropout1): Dropout(p=0.1, inplace=False)
  (dropout2): Dropout(p=0.1, inplace=False)
)
(7): TransformerEncoderLayer(
  (self_attn): MultiheadAttention(
    (out_proj): NonDynamicallyQuantizableLinear(in_features=169, out_features=169, bias=True)
  )
  (linear1): Linear(in_features=169, out_features=2048, bias=True)
  (dropout): Dropout(p=0.1, inplace=False)
  (linear2): Linear(in_features=2048, out_features=169, bias=True)
  (norm1): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
  (norm2): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
  (dropout1): Dropout(p=0.1, inplace=False)
  (dropout2): Dropout(p=0.1, inplace=False)
)
(8): TransformerEncoderLayer(
  (self_attn): MultiheadAttention(
    (out_proj): NonDynamicallyQuantizableLinear(in_features=169, out_features=169, bias=True)
  )
  (linear1): Linear(in_features=169, out_features=2048, bias=True)
  (dropout): Dropout(p=0.1, inplace=False)
  (linear2): Linear(in_features=2048, out_features=169, bias=True)
  (norm1): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
  (norm2): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
  (dropout1): Dropout(p=0.1, inplace=False)
  (dropout2): Dropout(p=0.1, inplace=False)
)
```

```
      (9): TransformerEncoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=169, out_features=169, bias=True)
        )
        (linear1): Linear(in_features=169, out_features=2048, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (linear2): Linear(in_features=2048, out_features=169, bias=True)
        (norm1): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
      )
      (10): TransformerEncoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=169, out_features=169, bias=True)
        )
        (linear1): Linear(in_features=169, out_features=2048, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (linear2): Linear(in_features=2048, out_features=169, bias=True)
        (norm1): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
      )
      (11): TransformerEncoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=169, out_features=169, bias=True)
        )
        (linear1): Linear(in_features=169, out_features=2048, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (linear2): Linear(in_features=2048, out_features=169, bias=True)
        (norm1): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
      )
    )
  )
  (op): Linear(in_features=169, out_features=55, bias=True)
)
```

After training the model for 25 epochs, the training results and performance metrics are as follows.

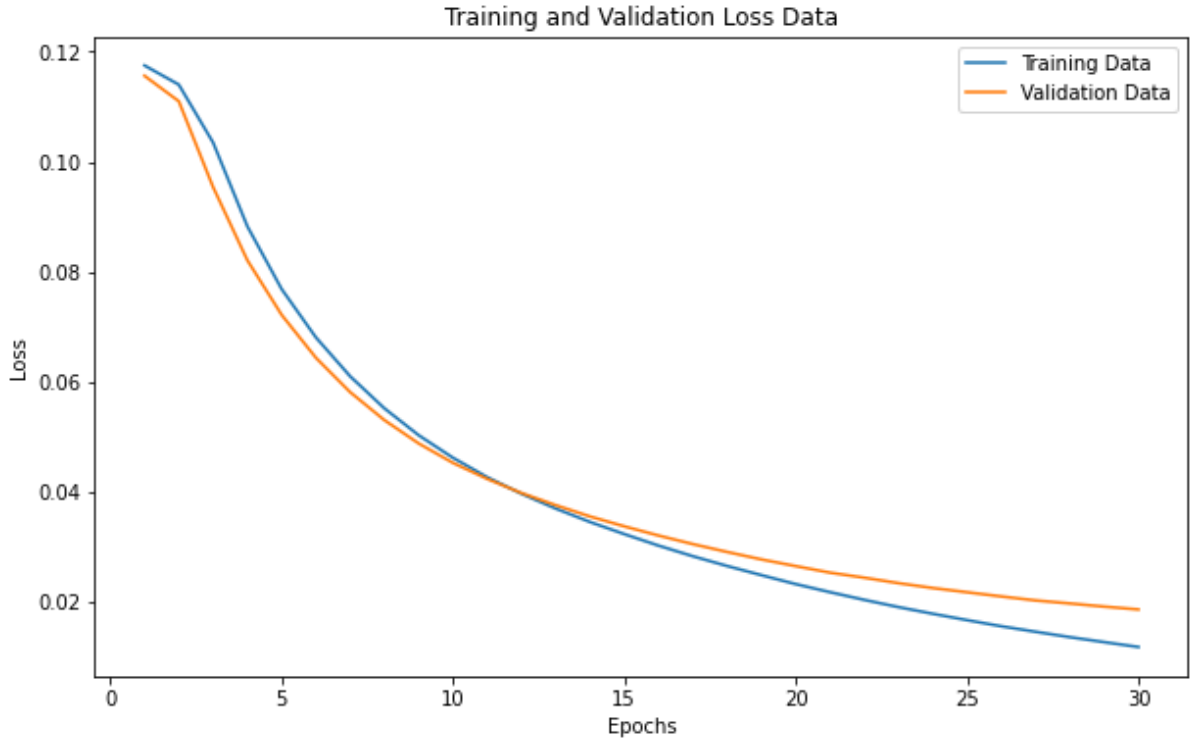
Training Summary & Performance Metrics

```
Accuracy and Loss values:

Training Accuracy : 95.11 %
Validation Accuracy : 87.10 %
Testing Accuracy : 89.30 %

Training Loss : 0.04205652087620772
Validation Loss : 0.04314945886792294
Testing Loss : 0.016550707251145033
```

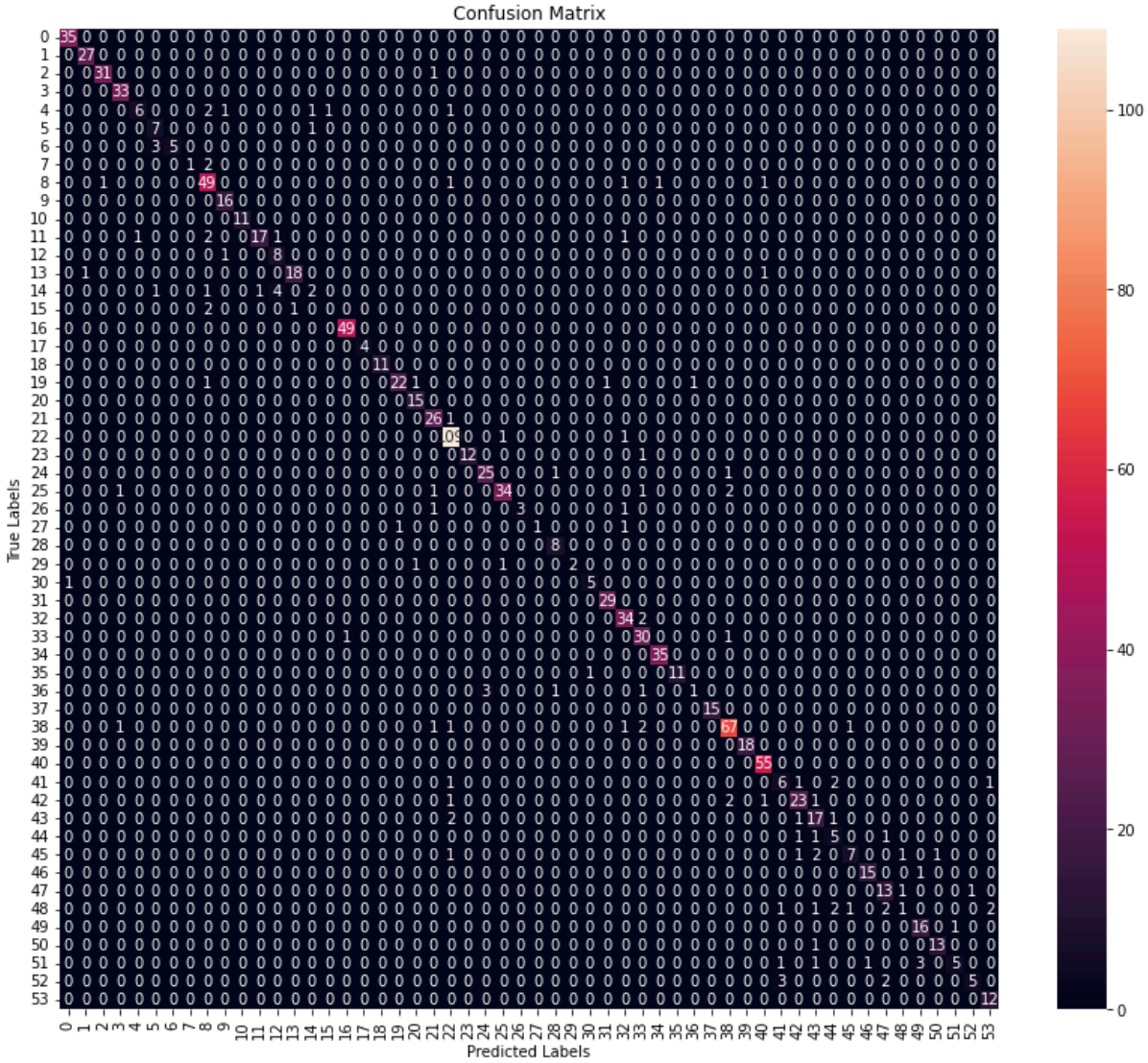
```
Performance Metrics:
Time to Train : 5.492368459701538 seconds
Accuracy      : 89.29 %
Precision     : [0.97222222 0.96428571 0.96875    0.94285714 0.85714286 0.63636364
 1.         1.         0.83050847 0.88888889 1.         0.94444444
 0.61538462 0.94736842 0.5        0.         0.98       1.
 1.         0.95652174 0.88235294 0.86666667 0.92372881 1.
 0.89285714 0.94444444 1.         1.         0.8        1.
 0.83333333 0.96666667 0.85       0.81081081 0.97222222 1.
 0.5        1.         0.91780822 1.         0.94827586 0.54545455
 0.85185185 0.70833333 0.5        0.77777778 0.9375     0.72222222
 0.33333333 0.8        0.92857143 0.83333333 0.83333333 0.8       ]
Recall        : [1.         1.         0.96875    1.         0.5        0.875
 0.625      0.33333333 0.90740741 1.         1.         0.77272727
 0.88888889 0.9        0.22222222 0.         1.         1.
 1.         0.84615385 1.         0.96296296 0.98198198 0.92307692
 0.92592593 0.87179487 0.6        0.33333333 1.         0.5
 0.83333333 1.         0.94444444 0.9375     1.         0.91666667
 0.16666667 1.         0.90540541 1.         1.         0.54545455
 0.82142857 0.80952381 0.625      0.53846154 0.9375     0.86666667
 0.1        0.94117647 0.92857143 0.45454545 0.5        1.        ]
F1 Score      : [0.98591549 0.98181818 0.96875    0.97058824 0.63157895 0.73684211
 0.76923077 0.5        0.86725664 0.94117647 1.         0.85
 0.72727273 0.92307692 0.30769231 0.         0.98989899 1.
 1.         0.89795918 0.9375     0.9122807  0.95196507 0.96
 0.90909091 0.90666667 0.75       0.5        0.88888889 0.66666667
 0.83333333 0.98305085 0.89473684 0.86956522 0.98591549 0.95652174
 0.25       1.         0.91156463 1.         0.97345133 0.54545455
 0.83636364 0.75555556 0.55555556 0.63636364 0.9375     0.78787879
 0.15384615 0.86486486 0.92857143 0.58823529 0.625      0.88888889]
```

Visualization of Training & Validation Accuracy over Epochs

Visualization of Training & Validation Loss over Epochs



Confusion Matrix Generated from Testing the Model



iv) ViTResNet34 Fusion Model

Summary of the Model

```
VisionTransformer_ResNet_Concatenator(
  (vision_model): VisionTransformer_Model(
    (patch_embedding): Sequential(
      (0): Conv2d(3, 169, kernel_size=(3, 3), stride=(3, 3))
      (1): BatchNorm2d(169, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): Flatten(start_dim=2, end_dim=-1)
    )
    (encoder): TransformerEncoder(
      (layers): ModuleList(
        (0): TransformerEncoderLayer(
          (self_attn): MultiheadAttention(
```

```
            (out_proj): NonDynamicallyQuantizableLinear(in_features=169, out_features=169,
bias=True)
          )
          (linear1): Linear(in_features=169, out_features=2048, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
          (linear2): Linear(in_features=2048, out_features=169, bias=True)
          (norm1): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
          (norm2): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
          (dropout1): Dropout(p=0.1, inplace=False)
          (dropout2): Dropout(p=0.1, inplace=False)
        )
        (1): TransformerEncoderLayer(
          (self_attn): MultiheadAttention(
            (out_proj): NonDynamicallyQuantizableLinear(in_features=169, out_features=169,
bias=True)
          )
          (linear1): Linear(in_features=169, out_features=2048, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
          (linear2): Linear(in_features=2048, out_features=169, bias=True)
          (norm1): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
          (norm2): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
          (dropout1): Dropout(p=0.1, inplace=False)
          (dropout2): Dropout(p=0.1, inplace=False)
        )
        (2): TransformerEncoderLayer(
          (self_attn): MultiheadAttention(
            (out_proj): NonDynamicallyQuantizableLinear(in_features=169, out_features=169,
bias=True)
          )
          (linear1): Linear(in_features=169, out_features=2048, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
          (linear2): Linear(in_features=2048, out_features=169, bias=True)
          (norm1): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
          (norm2): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
          (dropout1): Dropout(p=0.1, inplace=False)
          (dropout2): Dropout(p=0.1, inplace=False)
        )
        (3): TransformerEncoderLayer(
          (self_attn): MultiheadAttention(
```

```
            (out_proj): NonDynamicallyQuantizableLinear(in_features=169, out_features=169,
bias=True)
          )
          (linear1): Linear(in_features=169, out_features=2048, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
          (linear2): Linear(in_features=2048, out_features=169, bias=True)
          (norm1): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
          (norm2): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
          (dropout1): Dropout(p=0.1, inplace=False)
          (dropout2): Dropout(p=0.1, inplace=False)
        )
        (4): TransformerEncoderLayer(
          (self_attn): MultiheadAttention(
            (out_proj): NonDynamicallyQuantizableLinear(in_features=169, out_features=169,
bias=True)
          )
          (linear1): Linear(in_features=169, out_features=2048, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
          (linear2): Linear(in_features=2048, out_features=169, bias=True)
          (norm1): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
          (norm2): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
          (dropout1): Dropout(p=0.1, inplace=False)
          (dropout2): Dropout(p=0.1, inplace=False)
        )
        (5): TransformerEncoderLayer(
          (self_attn): MultiheadAttention(
            (out_proj): NonDynamicallyQuantizableLinear(in_features=169, out_features=169,
bias=True)
          )
          (linear1): Linear(in_features=169, out_features=2048, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
          (linear2): Linear(in_features=2048, out_features=169, bias=True)
          (norm1): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
          (norm2): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
          (dropout1): Dropout(p=0.1, inplace=False)
          (dropout2): Dropout(p=0.1, inplace=False)
        )
        (6): TransformerEncoderLayer(
          (self_attn): MultiheadAttention(
```

```
            (out_proj): NonDynamicallyQuantizableLinear(in_features=169, out_features=169,
bias=True)
          )
          (linear1): Linear(in_features=169, out_features=2048, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
          (linear2): Linear(in_features=2048, out_features=169, bias=True)
          (norm1): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
          (norm2): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
          (dropout1): Dropout(p=0.1, inplace=False)
          (dropout2): Dropout(p=0.1, inplace=False)
        )
        (7): TransformerEncoderLayer(
          (self_attn): MultiheadAttention(
            (out_proj): NonDynamicallyQuantizableLinear(in_features=169, out_features=169,
bias=True)
          )
          (linear1): Linear(in_features=169, out_features=2048, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
          (linear2): Linear(in_features=2048, out_features=169, bias=True)
          (norm1): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
          (norm2): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
          (dropout1): Dropout(p=0.1, inplace=False)
          (dropout2): Dropout(p=0.1, inplace=False)
        )
        (8): TransformerEncoderLayer(
          (self_attn): MultiheadAttention(
            (out_proj): NonDynamicallyQuantizableLinear(in_features=169, out_features=169,
bias=True)
          )
          (linear1): Linear(in_features=169, out_features=2048, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
          (linear2): Linear(in_features=2048, out_features=169, bias=True)
          (norm1): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
          (norm2): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
          (dropout1): Dropout(p=0.1, inplace=False)
          (dropout2): Dropout(p=0.1, inplace=False)
        )
        (9): TransformerEncoderLayer(
          (self_attn): MultiheadAttention(
```

```
            (out_proj): NonDynamicallyQuantizableLinear(in_features=169, out_features=169,
bias=True)
          )
          (linear1): Linear(in_features=169, out_features=2048, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
          (linear2): Linear(in_features=2048, out_features=169, bias=True)
          (norm1): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
          (norm2): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
          (dropout1): Dropout(p=0.1, inplace=False)
          (dropout2): Dropout(p=0.1, inplace=False)
        )
        (10): TransformerEncoderLayer(
          (self_attn): MultiheadAttention(
            (out_proj): NonDynamicallyQuantizableLinear(in_features=169, out_features=169,
bias=True)
          )
          (linear1): Linear(in_features=169, out_features=2048, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
          (linear2): Linear(in_features=2048, out_features=169, bias=True)
          (norm1): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
          (norm2): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
          (dropout1): Dropout(p=0.1, inplace=False)
          (dropout2): Dropout(p=0.1, inplace=False)
        )
        (11): TransformerEncoderLayer(
          (self_attn): MultiheadAttention(
            (out_proj): NonDynamicallyQuantizableLinear(in_features=169, out_features=169,
bias=True)
          )
          (linear1): Linear(in_features=169, out_features=2048, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
          (linear2): Linear(in_features=2048, out_features=169, bias=True)
          (norm1): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
          (norm2): LayerNorm((169,), eps=1e-05, elementwise_affine=True)
          (dropout1): Dropout(p=0.1, inplace=False)
          (dropout2): Dropout(p=0.1, inplace=False)
        )
      )
    )
```

```
  )
  (resnet_model): ResNet34(
    (block1): Sequential(
      (0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU(inplace=True)
      (3): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    )
    (block2): Sequential(
      (0): ResidualBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): ResidualBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (2): ResidualBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
```

```
    (block3): Sequential(
      (0): ResidualBlock(
        (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): ResidualBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (2): ResidualBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (3): ResidualBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (block4): Sequential(
      (0): ResidualBlock(
        (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): ResidualBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (2): ResidualBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
```

```
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)

      )

      (3): ResidualBlock(

        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)

        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)

        (relu): ReLU(inplace=True)

        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)

        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)

      )

      (4): ResidualBlock(

        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)

        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)

        (relu): ReLU(inplace=True)

        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)

        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)

      )

      (5): ResidualBlock(

        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)

        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)

        (relu): ReLU(inplace=True)

        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)

        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)

      )

    )

    (block5): Sequential(

      (0): ResidualBlock(

        (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)

        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)

        (relu): ReLU(inplace=True)
```

```
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)

        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)

        (downsample): Sequential(

          (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)

          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)

        )

      )

      (1): ResidualBlock(

        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)

        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)

        (relu): ReLU(inplace=True)

        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)

        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)

      )

      (2): ResidualBlock(

        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)

        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)

        (relu): ReLU(inplace=True)

        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)

        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)

      )

    )

    (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))

    (fc): Linear(in_features=512, out_features=169, bias=True)

  )

  (conv1d): Conv1d(338, 128, kernel_size=(1,), stride=(1,))

  (fc): Linear(in_features=128, out_features=55, bias=True)

)
```

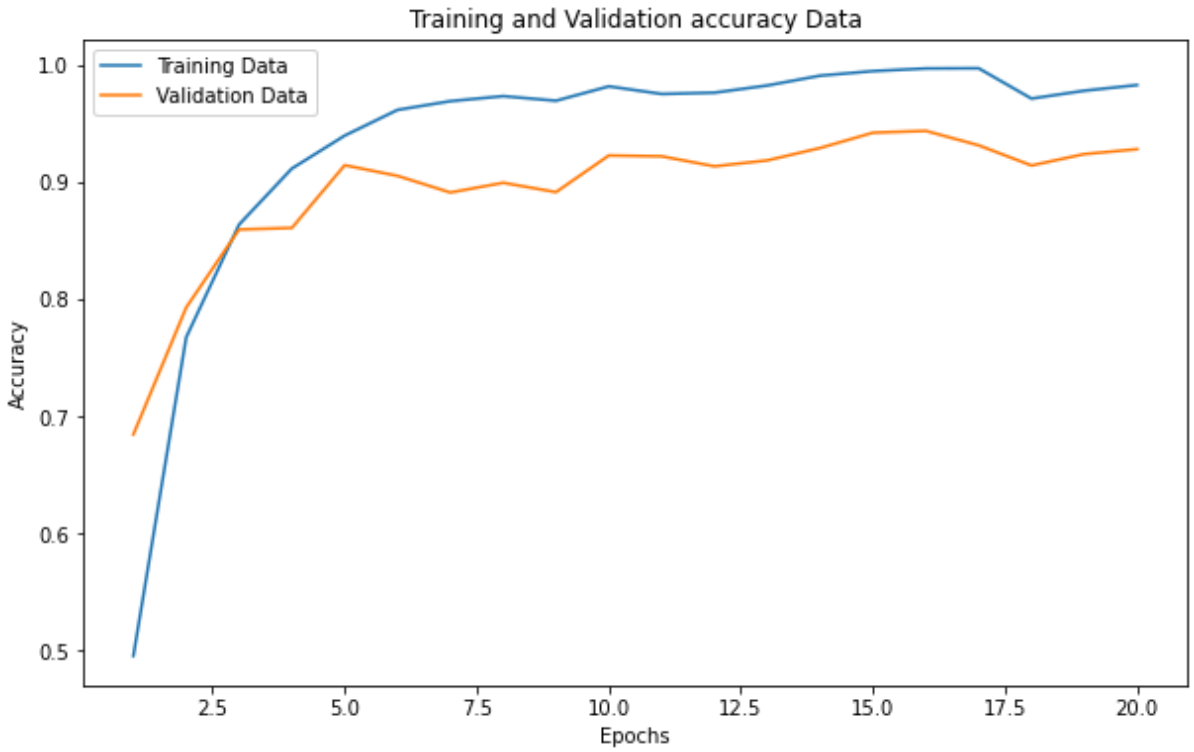
After training the model for 25 epochs, the training results and performance metrics are as follows.

Training Summary & Performance Metrics

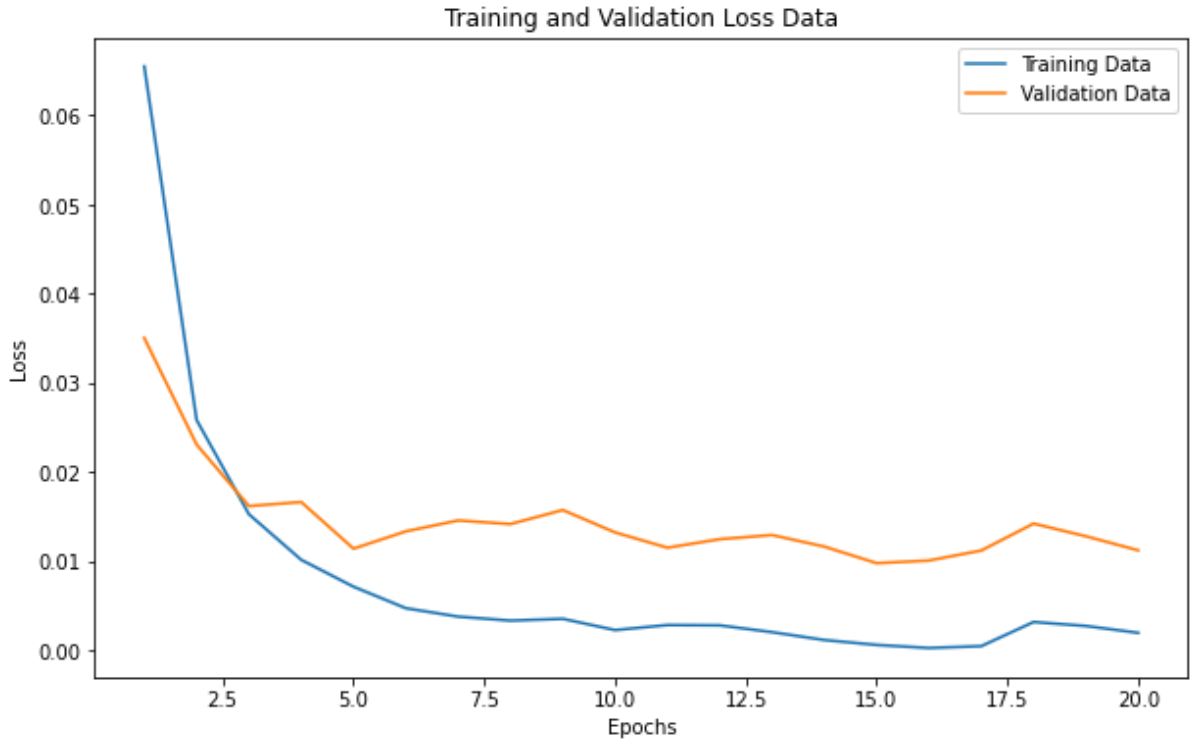
```
Accuracy and Loss values:

Training Accuracy : 98.25 %
Validation Accuracy : 92.78 %
Testing Accuracy : 92.26 %

Training Loss : 0.00802094470150421
Validation Loss : 0.014597275203240851
Testing Loss : 0.011283281189470652
```

```
Performance Metrics:
Time to Train : 5.317040205001831 seconds
Accuracy      : 92.25 %
Precision     : [0.9375     1.         1.         0.94736842 0.91666667 1.
 0.75       1.         0.92424242 0.85714286 0.90909091 0.89285714
 0.93333333 0.84210526 0.66666667 1.         0.97916667 0.75
 1.         0.9        0.9        0.97435897 0.94845361 0.875
 0.97222222 0.97368421 1.         0.75       0.42857143 0.77777778
 1.         0.96428571 1.         1.         1.         0.5
 0.90909091 0.94366197 0.96428571 0.94871795 1.         1.
 0.75675676 0.54545455 0.5        0.9        0.94117647 0.9
 0.54545455 1.         1.         1.         1.         0.81818182]
Recall        : [1.         0.92857143 0.97222222 0.94736842 0.57894737 1.
 1.         0.4        0.96825397 1.         1.         0.92592593
 1.         0.94117647 0.4        0.85714286 0.95918367 1.
 0.91666667 0.94736842 0.94736842 0.97435897 0.91089109 1.
 1.         0.94871795 0.9        0.75       1.         0.875
 1.         1.         0.94594595 1.         0.8        0.22222222
 0.83333333 0.93055556 1.         0.97368421 0.92857143 0.96428571
 0.8        0.85714286 0.5        0.9        1.         0.85714286
 0.85714286 1.         0.875      1.         0.6        0.75      ]
F1 Score      : [0.96774194 0.96296296 0.98591549 0.94736842 0.70967742 1.
 0.85714286 0.57142857 0.94573643 0.92307692 0.95238095 0.90909091
 0.96551724 0.88888889 0.5        0.92307692 0.96907216 0.85714286
 0.95652174 0.92307692 0.92307692 0.97435897 0.92929293 0.93333333
 0.98591549 0.96103896 0.94736842 0.75       0.6        0.82352941
 1.         0.98181818 0.97222222 1.         0.88888889 0.30769231
 0.86956522 0.93706294 0.98181818 0.96103896 0.96296296 0.98181818
 0.77777778 0.66666667 0.5        0.9        0.96969697 0.87804878
 0.66666667 1.         0.93333333 1.         0.75       0.7826087 ]
```

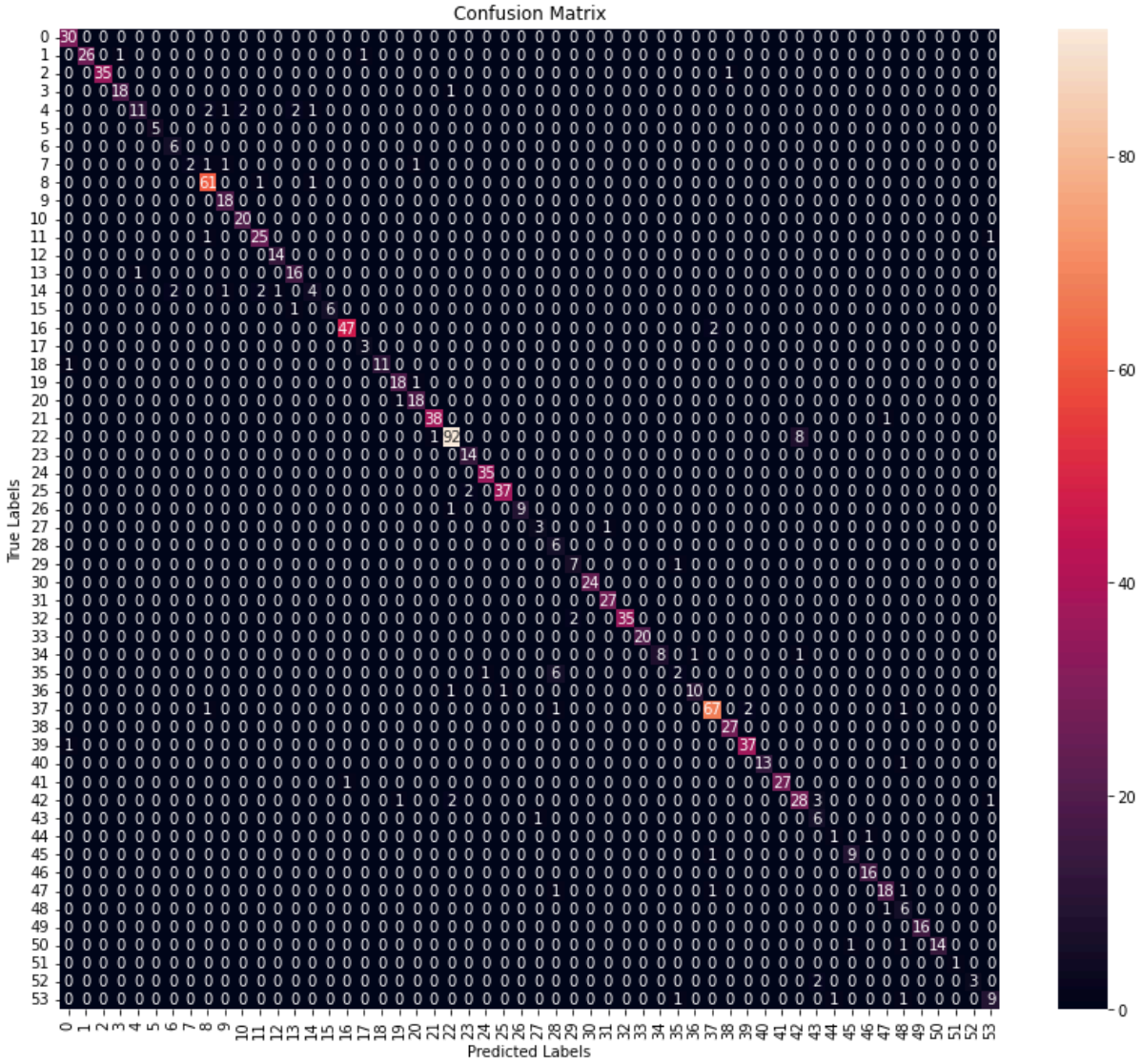
Visualization of Training & Validation Accuracy over Epochs



Visualization of Training & Validation Loss over Epochs



Confusion Matrix Generated from Testing the Model



From the above images we can see that the accuracy for the **VGG-13 is 87.35** , **ResNet-34 is 72.59**, **Vision Transformer is 89.30**, **VitResNet Fusion model is 92.26**.

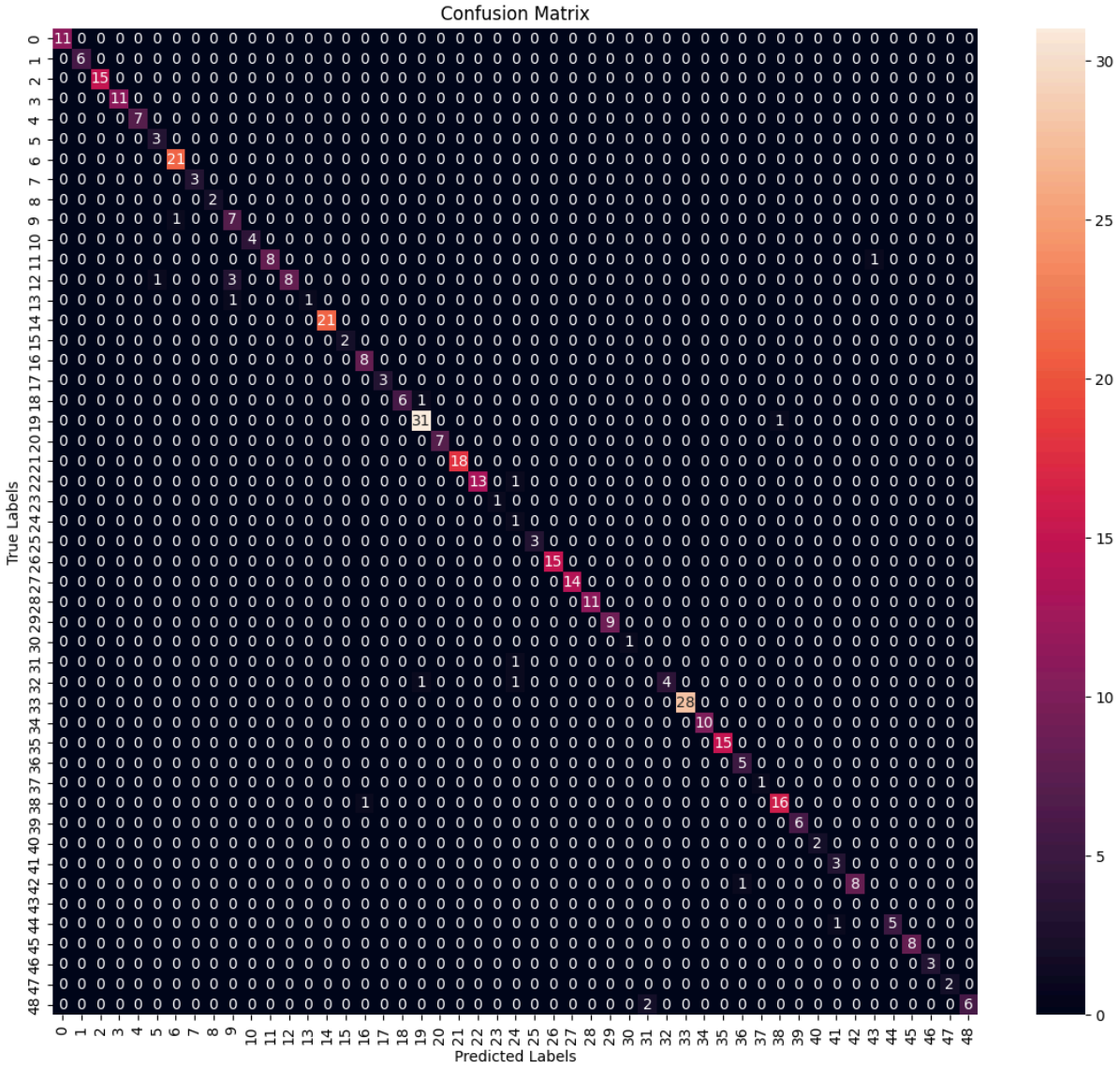
Since the fusion model has the highest testing accuracy from the above classifier models we are considering it to classify the detected objects which we obtain from the YOLO pretrained model.

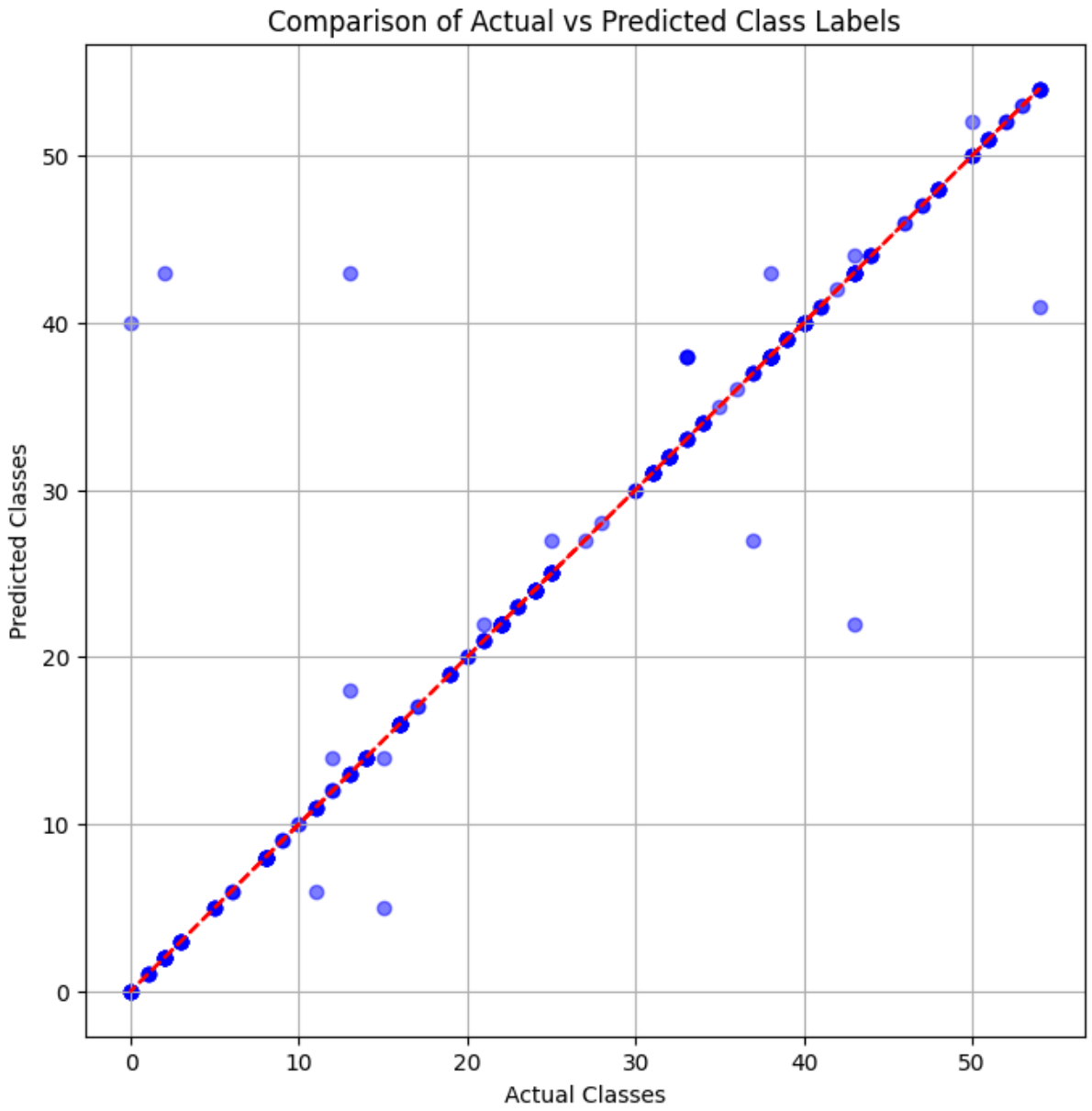
Why VitResNet Fusion model:

After experimenting with the above models, the VitResNet Fusion model performs better than the other models, such as VGG-13, ResNet34, and Vision Transformer. Consequently, the choice has been made to use the VitResNet Fusion model for image classification after the YOLO model has been used for object recognition. This calculated decision is supported by strong evidence of the VitResNet Fusion model's effectiveness and ability to produce dependable and accurate classification results, which is in perfect alignment with our goals.

Results for the VitResNet Fusion Model:

```
Performance Metrics:
Accuracy      : 93.19 %
Precision     : [          1          1          1          1          1       0.75    0.95455          1          1
0.63636          1          1          1          1          1          1    0.88889          1          1    0.93939
1          1          1          1       0.25          1
          1          1          1          1          1          0          1          1          1          1     0.
83333          1    0.94118          1          1       0.75          1          0          1          1          1
1          1]
Recall        : [          1          1          1          1          1          1          1          1          1
0.875          1    0.88889    0.66667        0.5          1          1          1          1    0.85714    0.96875
1          1    0.92857          1          1          1
          1          1          1          1          1          0    0.66667          1          1          1
1          1    0.94118          1          1          1    0.88889          0    0.83333          1          1
1       0.75]
F1 Score      : [          1          1          1          1          1    0.85714    0.97674          1          1
0.73684          1    0.94118        0.8    0.66667          1          1    0.94118          1    0.92308    0.95385
1          1    0.96296          1        0.4          1
          1          1          1          1          1          0        0.8          1          1          1     0.
90909          1    0.94118          1          1    0.85714    0.94118          0    0.90909          1          1
1    0.85714]
```





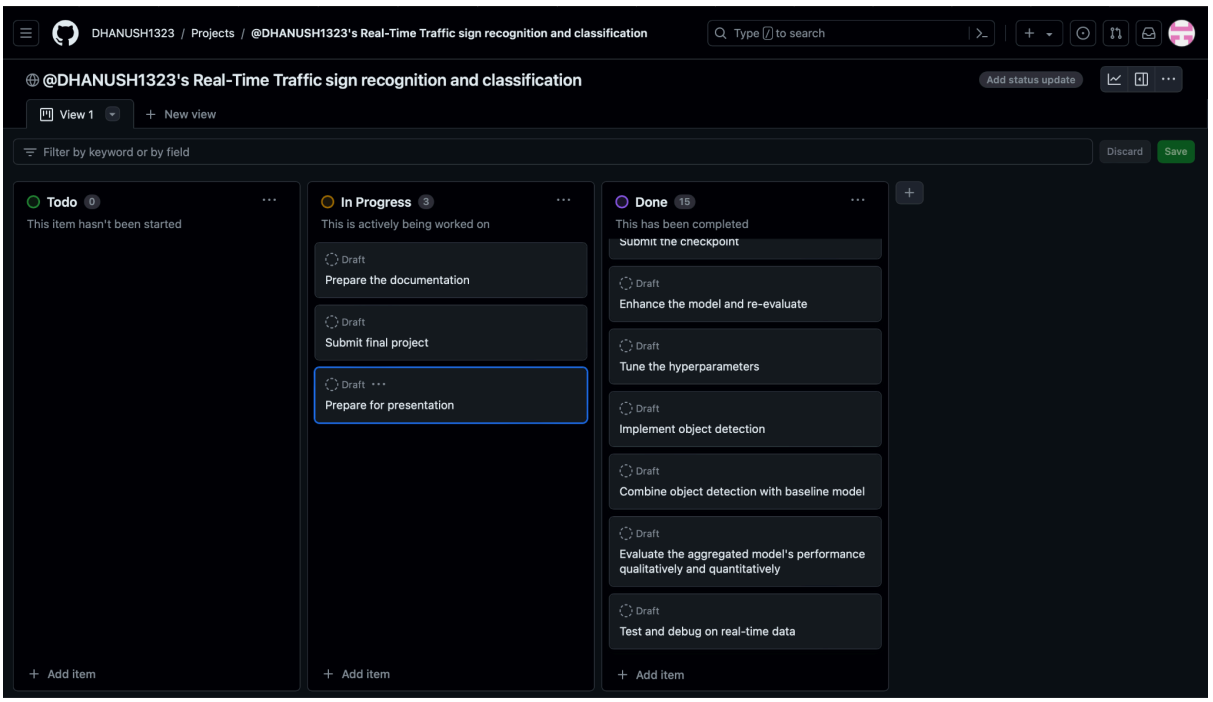
The overall accuracy for the model which uses YOLOv8 for object detection and VitResNet38 Fusion for object classification is 93.19%.

FUTURE WORK

With our developed deep learning model capable of detecting and classifying traffic signs from images, future work aims to enhance its suitability for integration into autonomous vehicles. Expanding the model to encompass a broader range of classes beyond conventional traffic signs, such as pedestrian crossings, road markings, and traffic lights, will bolster its applicability in real-world driving scenarios. Moreover, improving the model's accuracy on an even larger dataset by leveraging advanced data augmentation techniques, ensemble learning approaches, and robust training methodologies will ensure reliable performance across diverse environmental conditions. By continually refining the model's capabilities and scalability, we can lay the foundation for its seamless

integration into autonomous vehicles, contributing to safer and more efficient transportation systems.

GitHub Tool



REFERENCES

- CSE 574 - Introduction to Machine Learning Assignment 2 from Sneha Talapala, Dhanush Babu Ramadoss, Kishore Nithin Sridhar
- CSE 676 - Deep Learning Assignment 0, 1, 2 from Sneha Talapala, Dhanush Babu Ramadoss, Kishore Nithin Sridhar
- YoloV8 - https://docs.ultralytics.com/
- Vision Transformer - https://arxiv.org/abs/2010.11929