TRAFFIC SIGN RECOGNITION

Pritish Arora – mailto:pritisharora55@gmail.com

Introduction:

Traffic sign recognition is the process of identifying and classifying traffic signs in real-time using computer vision techniques. It is a crucial task in the field of intelligent transportation systems, as it enables vehicles to understand and respond to the visual cues present in the environment. This can assist in navigation, decision-making, and safety for both human-driven and autonomous vehicles.

Traffic signs serve as important visual cues for drivers, providing information about the rules and regulations of the road, as well as potential hazards and directions. Some common types of traffic signs include stop signs, speed limit signs, and turn restriction signs. Accurate and timely recognition of these signs is essential for safe and efficient navigation.

In this project, we propose the use of convolutional neural networks (CNNs) and transfer learning techniques for traffic sign recognition. Convolutional neural networks (CNNs) are a type of deep learning model that are well-suited for image recognition tasks, including traffic sign recognition. CNNs are able to learn hierarchical representations of spatial features in an input image, allowing them to make accurate predictions. Transfer learning is a technique that can be used to accelerate the training of a CNN for a specific task, by leveraging the knowledge learned by a pre-trained model on a large dataset.

Preliminaries:

The dataset used in this project consists of images of 42 different traffic signs, including various types of stop signs, yield signs, speed limit signs, and turn restriction signs. The dataset includes a large number of images for training the model, as well as a smaller number of images for testing the model's performance. There are

a total of over 27,000 training images and 12,630 testing images. The images have been resized to 30x30 pixels to reduce the computation.

The testing data includes a varying number of images per class, with a minimum of 60 and a maximum of 800 images per class. Furthermore, each class in the dataset has a minimum of 450 images, with some classes possessing as many as 1,000 images.

In this project, we will compare the performance of our own convolutional neural network (CNN) architectures with transfer-learning CNN architectures using VGG-16 and ResNet-50. We will evaluate the models using the accuracy metric. We will also apply regularization techniques such as dropout layers, model checkpointing, and batch normalization to improve the performance of the models on the traffic sign recognition task. By comparing the performance of these techniques, we aim to identify the most effective methods for improving the accuracy of CNN's on this task.

The problem setup involves training the CNN to recognize the different classes of traffic signs in the dataset. The network structure includes three convolutional layers with 64 and 128 filters respectively, followed by two fully-connected layers with 256 units respectively. The output layer has 42 units, corresponding to the 42 classes of traffic signs in the dataset. The network will be trained using categorical cross-entropy loss and evaluated using the accuracy and loss metrics.

In total, we have designed and implemented four distinct neural networks. Two of these networks were created entirely by us, while the other two were developed using transfer learning techniques. This approach allowed us to use a pre-trained model as the foundation for our own models, allowing us to train them more efficiently and with improved performance.The model will be implemented using the Tensorflow and keras frameworks and executed on an M1 pro apple silicon environment.The model will be built with the TensorFlow and Keras frameworks and run on an Apple M1 Pro silicon environment.

Results:

Model 1:

The CNN will be trained using the Adam optimization algorithm with a learning rate of 0.001 and a batch size of 64 for 25 epochs. During the training process, out of 27,000 images, we divided the images into training

and validation data. After training the model, we used our testing data, which is made up of 12,630 traffic signs, to test it.

Adam is an optimization algorithm that can be used instead of the traditional stochastic gradient descent method to change the weights of a network based on training data in an iterative way. Adam doesn't just change the learning rates of the parameters based on the average first moment (the mean), like RMSProp does. Instead, it takes into account the average of the second moments of the gradients (the uncentered variance). In particular, the technique calculates an exponential moving average of the gradient and the squared gradient, and the parameters beta1 and beta2 control how fast both moving averages lose their accuracy.

By the 1st epoch,
Model Accuracy: 0.70
Model Loss: 1.70

By the 25th epoch,
Model Accuracy: 0.997
Model Loss: 0.0093

Testing Results,
Model Accuracy: 94.46%

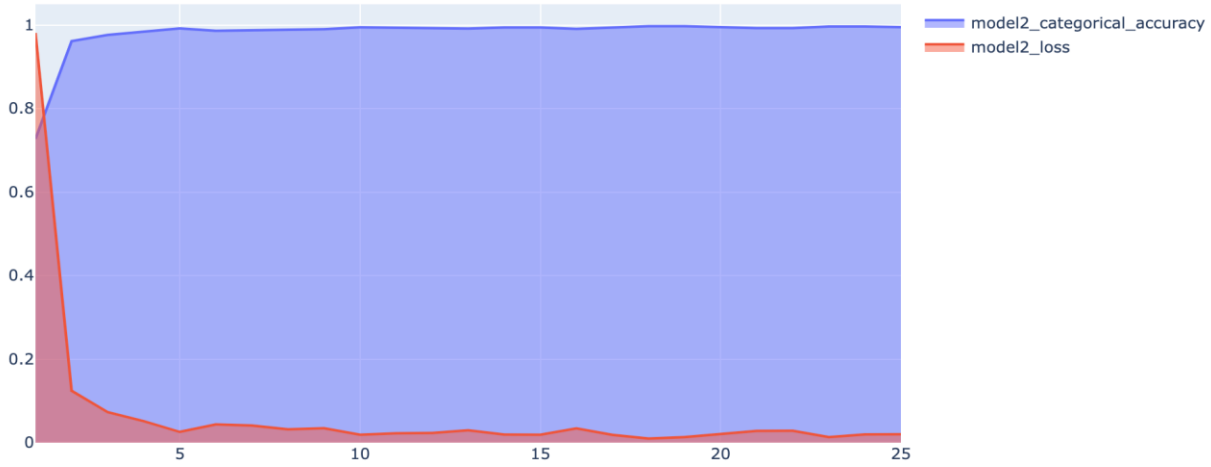

Figure (i) **Model Training Accuracy and Training Loss during Training**

The above graph demonstrates the relationship between the number of epochs used to train a model and its performance. As the number of epochs increases, the model loss decreases, indicating that the model is becoming more effective at minimizing the error in its predictions. Additionally, the model's accuracy also improves as the number of epochs increases, suggesting that the model is becoming more accurate in its

predictions. Overall, this graph shows that increasing the number of epochs can lead to better performance for a machine learning model.

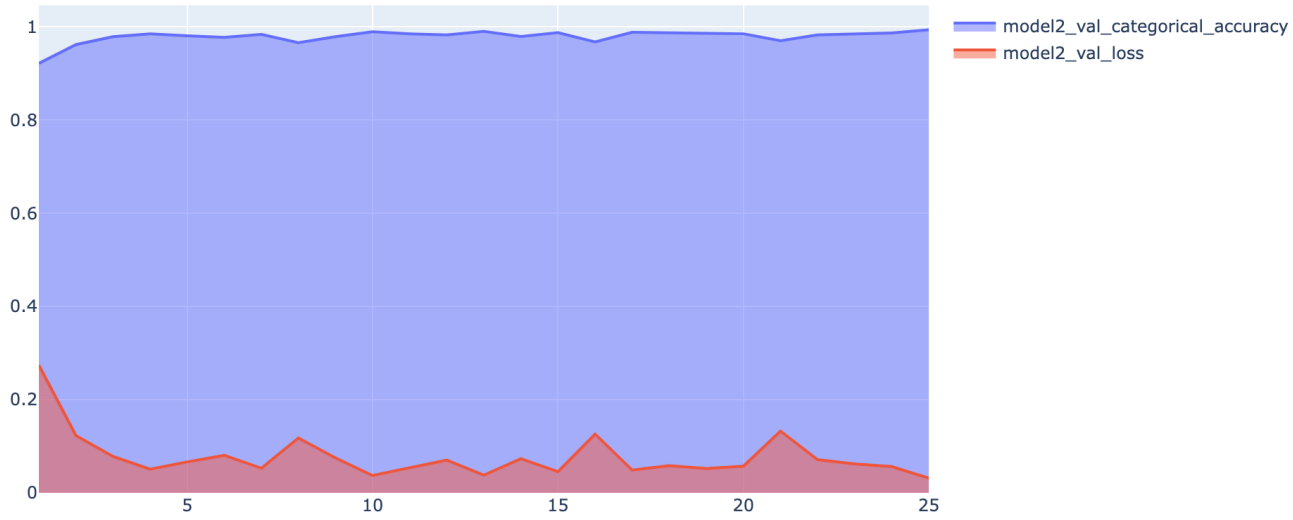

Figure (ii): **Model Validation Accuracy and Validation Loss during Training**

As shown by the above graph, there is a positive relationship between the number of epochs used to train a model and its performance. Specifically, as the number of epochs increases, the validation loss of the model decreases, indicating that the model is learning and improving. At the same time, the model's validation accuracy also improves as the number of epochs increases, suggesting that the model is becoming more accurate in its predictions. Overall, this graph indicates that increasing the number of epochs can lead to better performance for a machine learning model.

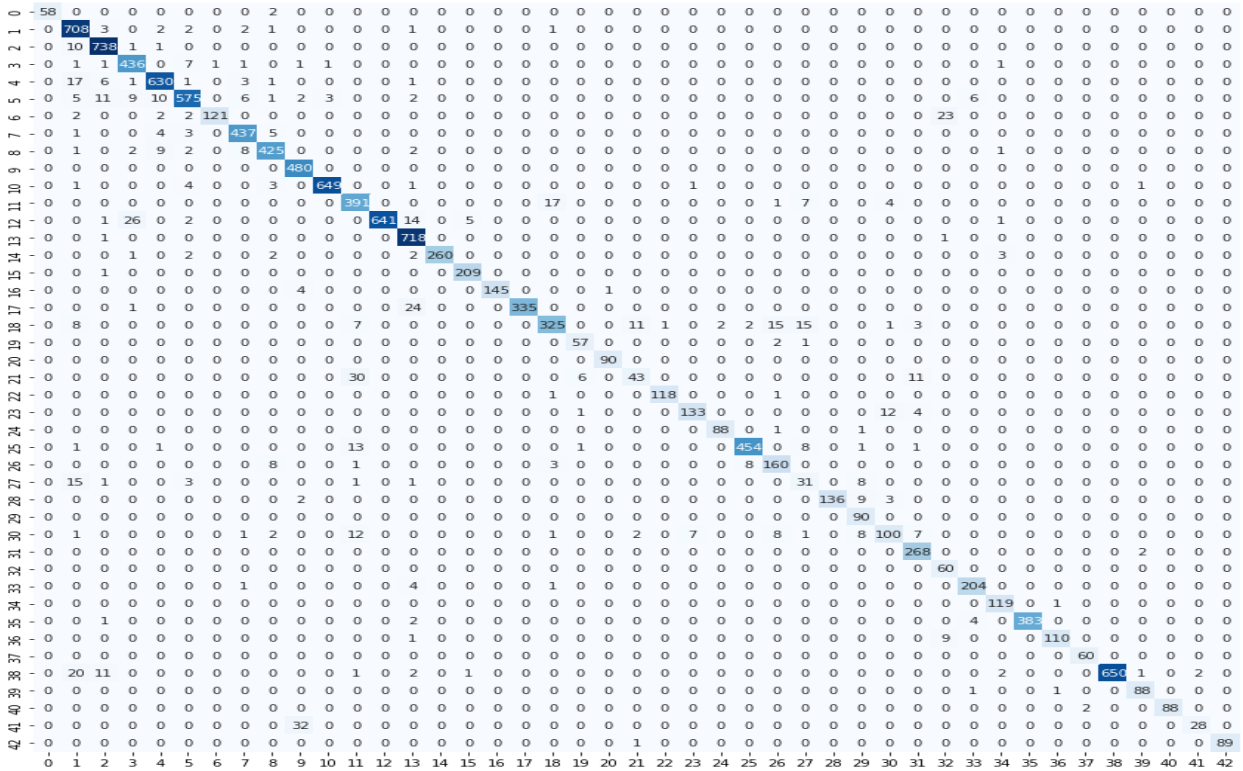

Figure (iii): **Confusion Matrix**

Model 2:

In our implementation of the convolutional neural network (CNN), we have used the Stochastic Gradient Descent (SGD) optimization algorithm with a learning rate of 0.001 and a batch size of 64. The model will be trained for 25 epochs using a dataset of 27,000 images, which will be divided into training and validation data. After the training process is complete, we will use a separate set of testing data, consisting of 12,630 traffic signs, to evaluate the performance of the trained model.

SGD updates the model's parameters in the direction of the negative gradient of the loss function, with the goal of minimizing the loss and improving the model's performance. Unlike other optimization algorithms, SGD uses a small batch of data to compute the gradient, which makes it more computationally efficient and scalable to large datasets.

By the 1st epoch,
Model Accuracy: 0.12
Model Loss: 3.50

By the 25th epoch,
Model Accuracy: 0.97
Model Loss: 0.086

Testing Results,
Model Accuracy: 93.19%

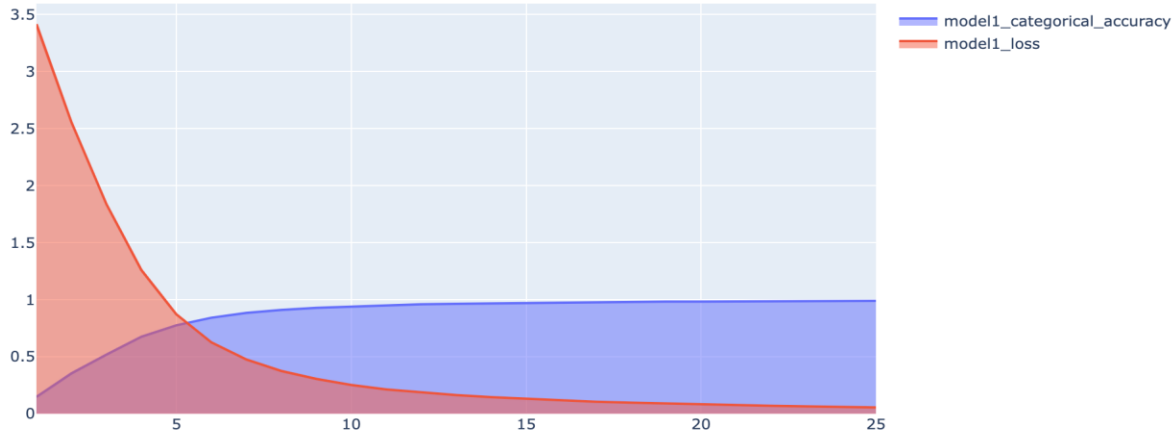

Figure (i) **Model Training Accuracy and Training Loss during Training**

From the above graph, we can say that as the number of epochs increases, the model loss decreases. At the same time, the model's accuracy increases as the number of epochs increases.

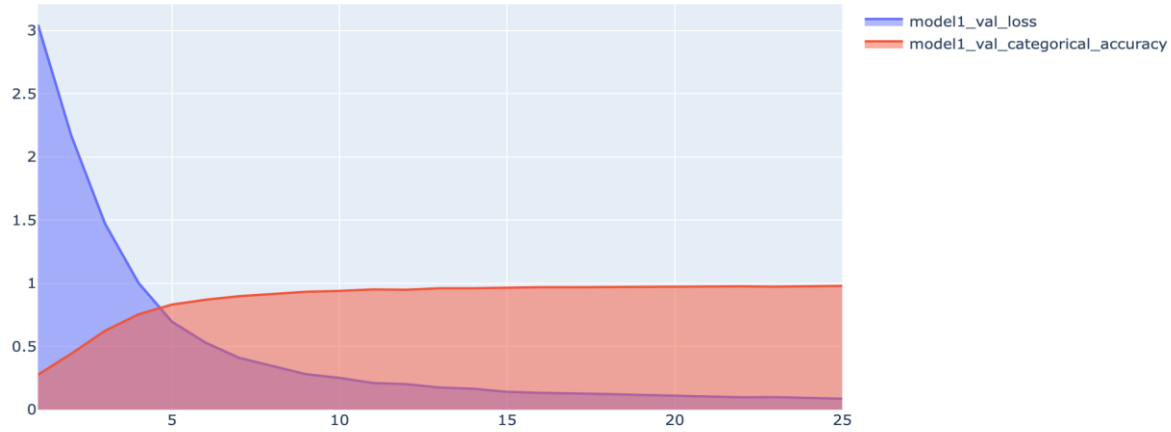

Figure (ii): **Model Validation Accuracy and Validation Loss during Training**

The above graph shows that as the number of epochs goes up, the validation loss of the model goes down. At the same time, the models' validation accuracy increases as the number of epochs increases.

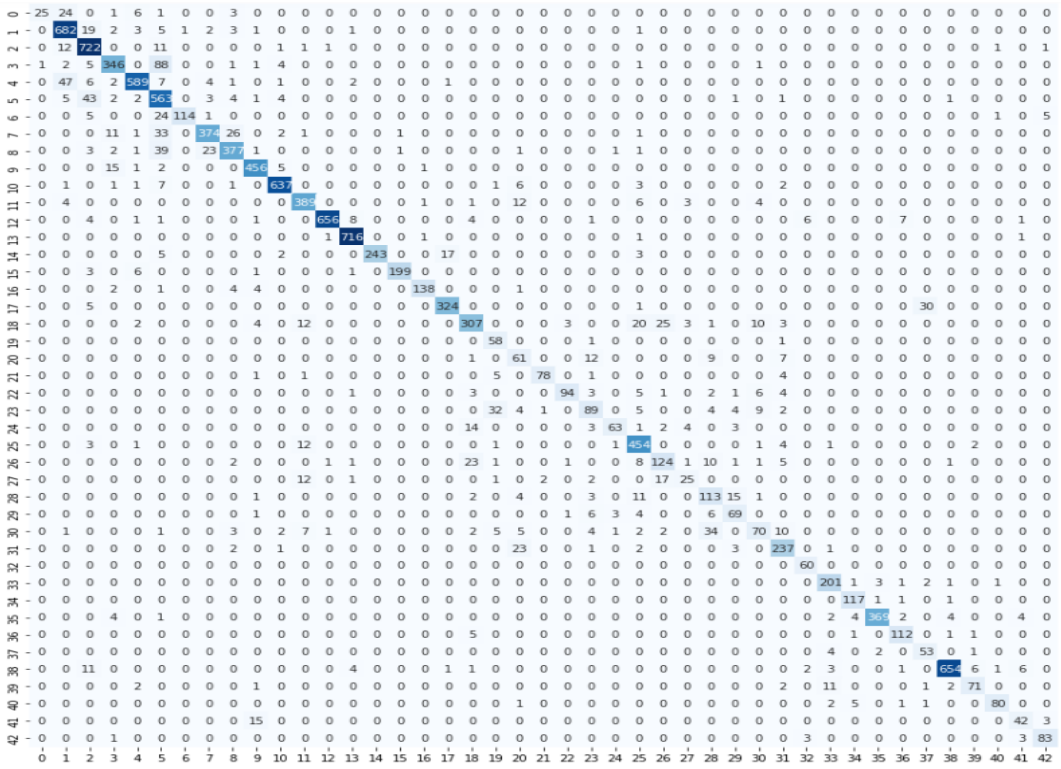

Figure (iii): **Confusion Matrix**

Model 3:

In this model, we have used the VGG-16 transfer learning technique. Transfer Learning in our case refers to using the feature learning layers of a trained CNN to classify a problem other than the one it was designed for. In other words, we leverage the patterns that the neural network discovered to be beneficial for classifying photographs of a particular problem to categorize images of an entirely new problem without retraining that portion of the network.

The architecture of VGG16 consists of a series of convolutional and pooling layers, followed by a few fully-connected layers. The convolutional layers are responsible for extracting features from the input images, while the fully connected layers use those features to make predictions. The architecture of VGG16 is relatively simple, but it has been shown to be effective for a wide range of image recognition tasks.
The convolutional layers in VGG16 use small filters and zero padding to learn local spatial hierarchies of features, while the pooling layers use a 2x2 filter and stride of 2 pixels to reduce the spatial dimensions of the feature maps while preserving important information. The fully-connected layers at the end of the network use the extracted features to make predictions. This simple but effective architecture has been widely used for image recognition tasks.

By the 1st epoch,
Model Accuracy: 0.40
Model Loss: 6.68

By the 25th epoch,
Model Accuracy: 0.91
Model Loss: 0.270

Testing Results,
Model Accuracy: 61.69%

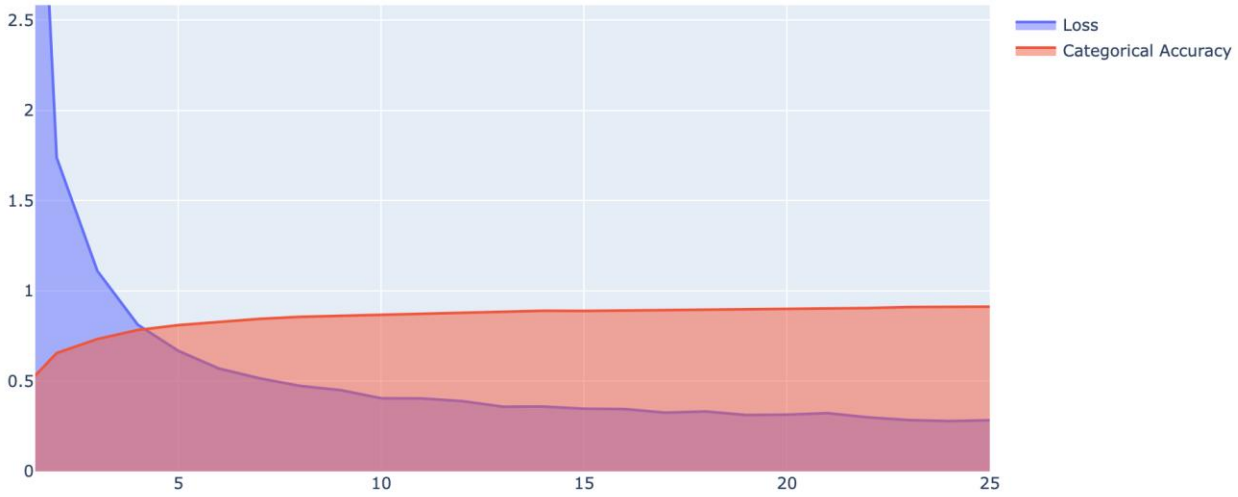

Figure (i) **Model Training Accuracy and Training Loss during Training**

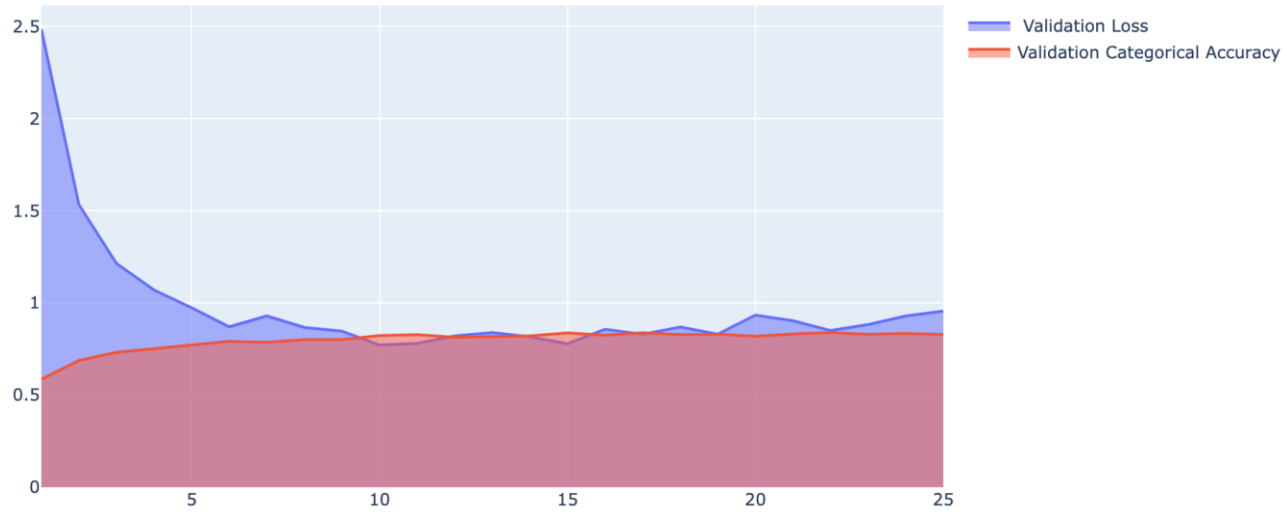

Figure (ii): **Model Validation Accuracy and Validation Loss during Training**

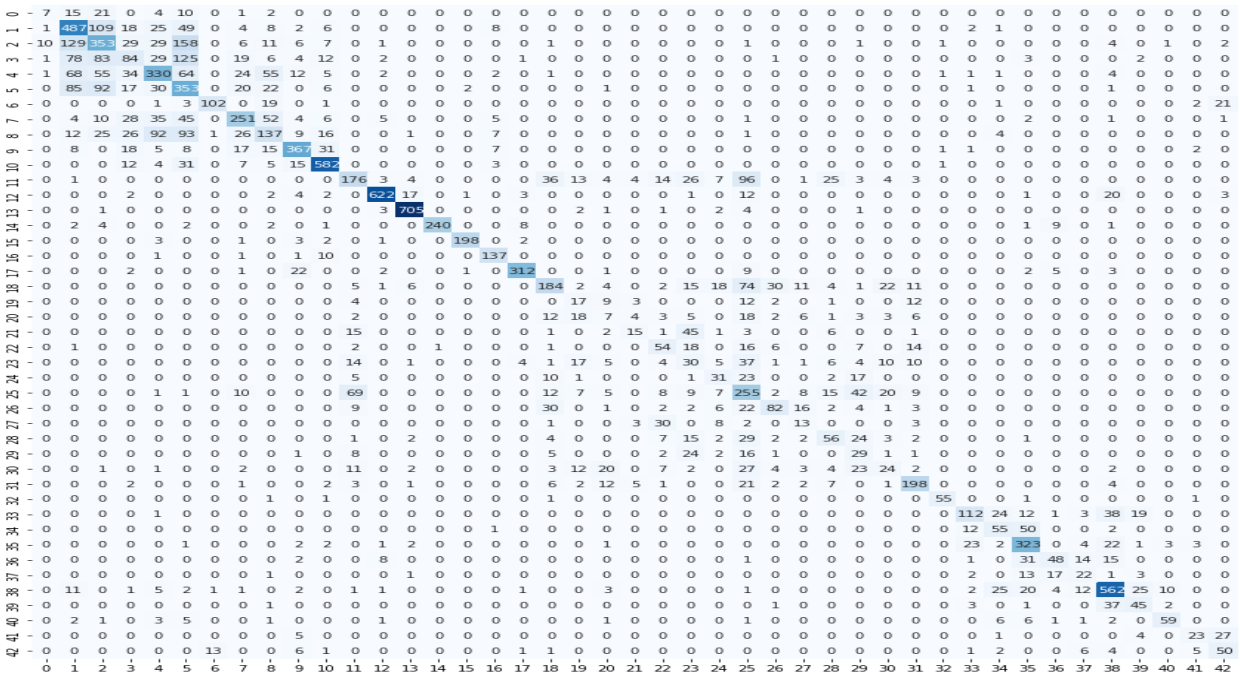

Figure (iii): **Confusion Matrix**

Model 4:

In this model, we have used the ResNet-50 transfer learning technique.

The fundamental unit of the 50-layer ResNet is a bottleneck architecture. A bottleneck residual block uses 1*1 convolutions, which is how it got its name. This is done to limit the number of parameters and matrix multiplications. Due to this, training time for each layer is drastically reduced. Instead of the standard two layers, it employs a three-layer stack.

By the 1st epoch,
Model Accuracy: 0.91
Model Loss: 0.26

By the 25th epoch,
Model Accuracy: 0.93
Model Loss: 0.19

Testing Results,
Model Accuracy: 61.71%

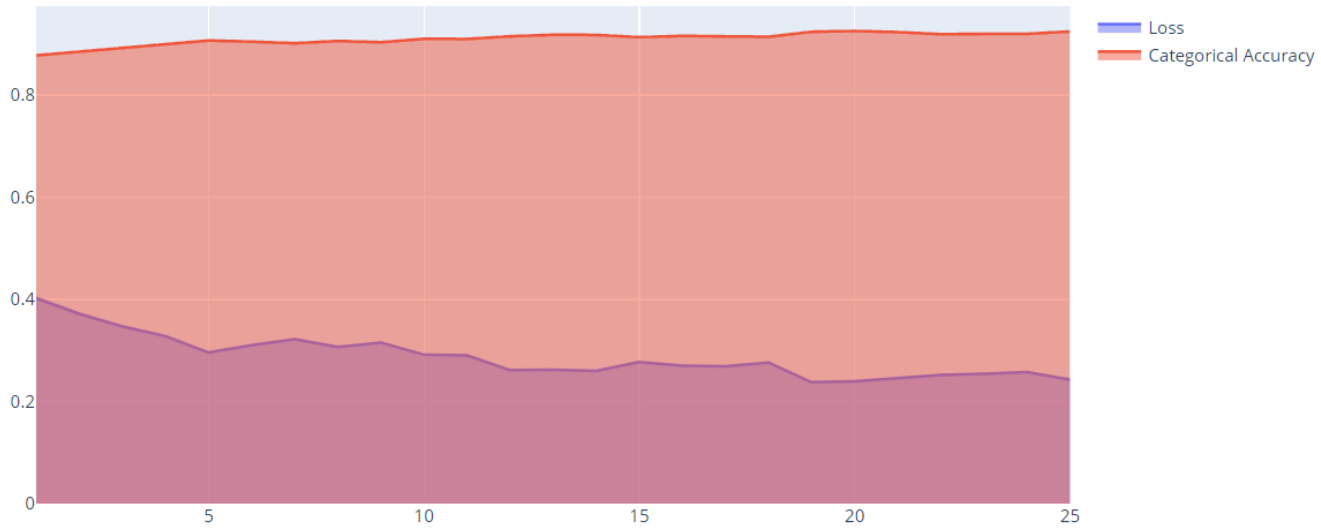

Figure (i) **Model Training Accuracy and Training Loss during Training**

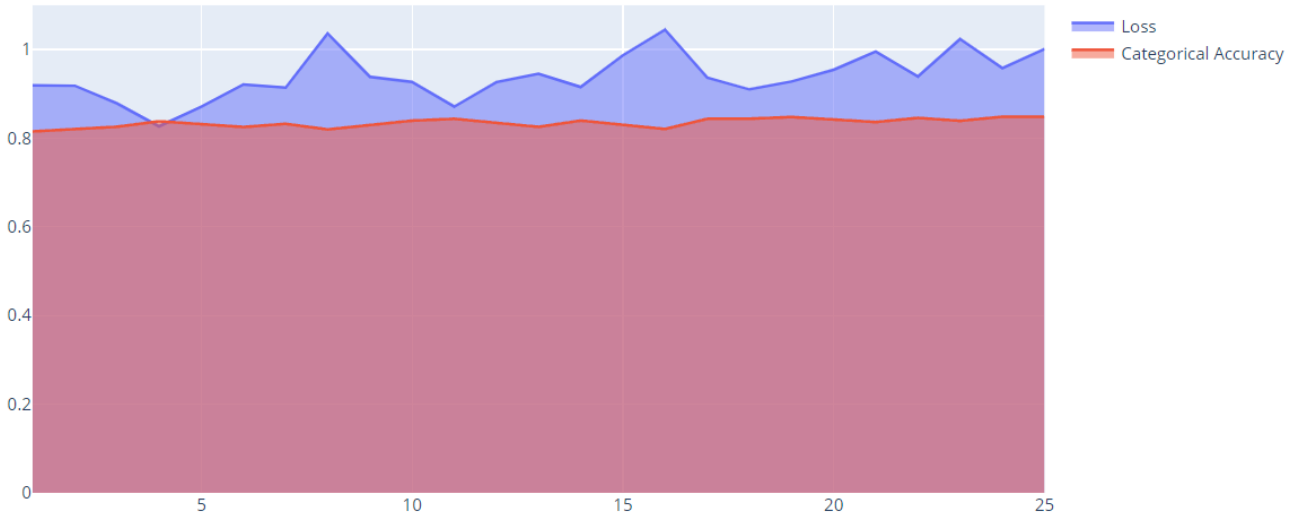

Figure (ii): **Model Validation Accuracy and Validation Loss during Training**

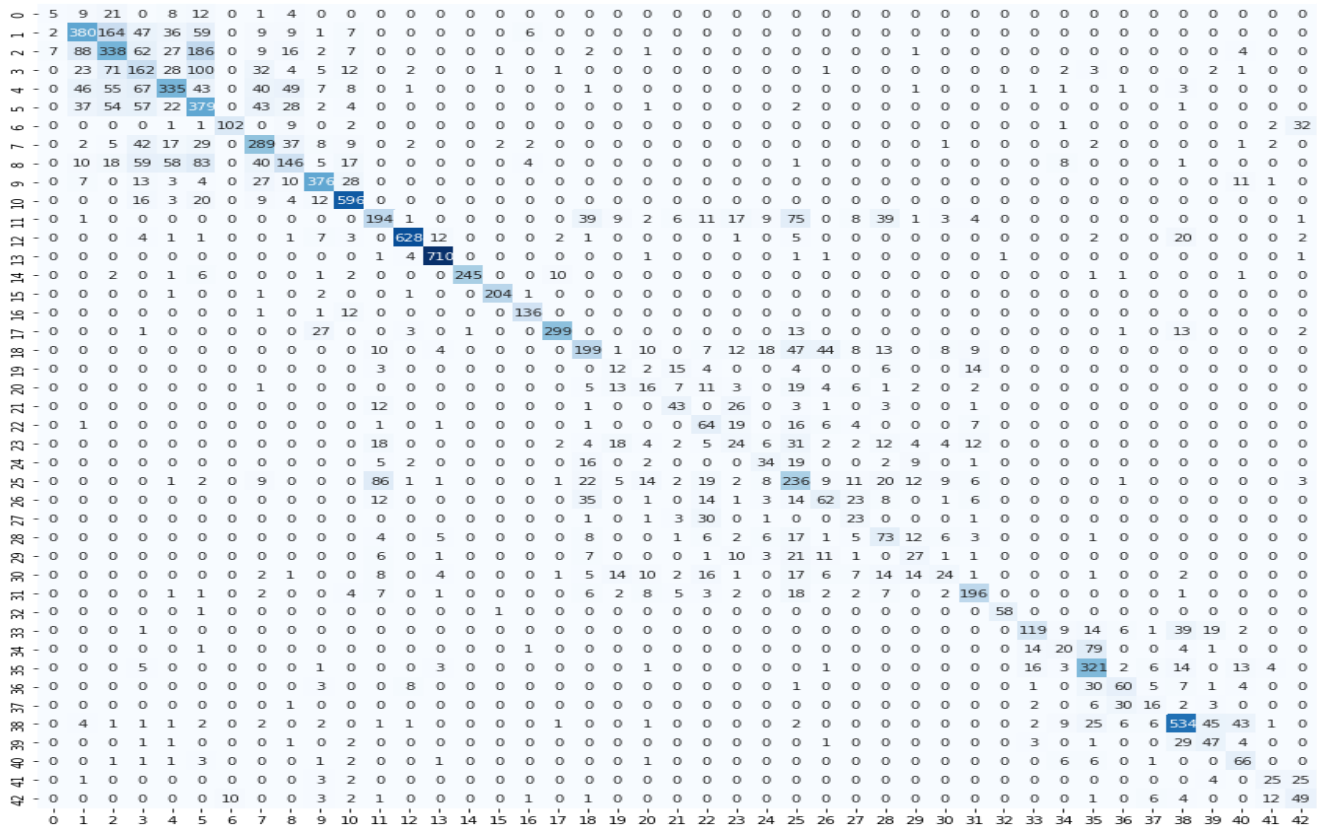

Figure (iii): **Confusion Matrix**

Before getting better accuracy, we trained models with different architectures.

Model 5:

Before training the model with CNNs, we trained the model with Artificial Neural Networks. To train the models, we first resized the images to 28*28 and then trained the model. But due to the high number of trainable parameters, we trained it for fewer epochs, and didn't achieve much accuracy.

Improvements:

1. Used Convolutional Neural Networks to improve the model's accuracy.

Model 6:

In this model, we trained a model with a lower number of epochs, layers, and neurons. During the training process, we had high accuracy on the training data, but the model performed very poorly while testing the data.

Improvements:

To increase the model's accuracy, we did the following

1. Added reguralisation techniques.
2. Increase the number of neurons and layers.
3. Using Transfer learning techniques

By increasing the number of layers and neurons, we increased the model's accuracy. Also, regularization techniques like dropout and batch normalization were added to the CNN to make the models more accurate by more than 5%.

Discussions:

Model Comparison:

Name	Training Accuracy	Testing Accuracy	No.of. Trainable Parameters
Model 1	0.99	0.95	496811
Model 2	0.97	0.93	1807531
Model 3	0.91	0.61	22059
Model 4	0.93	0.68	25878

Out of all the 4 models developed, the first model (i.e., Model 1) has the best training and testing accuracy of all the models, followed by Model 2. To begin, we trained a smaller neural network for a smaller number of epochs. From there on, we kept on increasing the layers and adding regularization techniques discussed in the class (such as initial weight initialization, Dropout, Batch Normalisation, and Model Checkpoint). After

we saw a significant change in the accuracy, we trained the model with different optimization algorithms, such as SGDprop, Adam, and RMSprop. Adam and SGDprop performed the best.

For the Model 3 and Model 4, we have used transfer learning techniques, VGG-16 and ResNet-50, respectively. These models seem to perform perfectly well on the training data, but they perform very poorly on the testing data. The model seems to overfit the training data, as there is a huge gap between the accuracy of the training and testing data. It's possible that the VGG16 model is not a good fit for this problem or that there are differences between the training data and the testing data that are causing the model to do poorly on the testing data. Transfer learning is a powerful tool, but it's important to make sure that the model you're using is well-suited to your particular problem.
One way to address this issue is to use a more robust evaluation method, such as cross-validation, to get a better estimate of your model's performance on unseen data. By using a different pre-trained model or fine-tuning the transfer learning model more to make it more closely match your data. It's also worth making sure that we have enough data and that it is of high quality, as this can have a big impact on the performance of our model.

When it comes to the Model 5 and Model 6, we first looked at how the model would perform with fewer neurons and fewer epochs. We incorporated multiple filter sizes for convolutional layers, Max Pooling Layers, and regularization such as Dropout and Batch Normalization to generalize our model.

Conclusion:

For our project, we classified a dataset of Traffic sign images to their respective labels using Convolutional neural networks and transfer learning with a resnet50 model and a VGG16 model. Out of all the models developed and trained, the model that performed the best was the model 1 which resulted in an Training accuracy of 0.997 and Testing accuracy 0.944.

References:

[1]“Traffic-Sign Recognition Using Deep Learning - Zhongbing Qin” https://cerv.aut.ac.nz/wp-content/uploads/2021/12/Traffic-Sign-Recognition-Using-Deep-Learning-1.pdf
[2] K. S. Boujemaa, I. Berrada, A. Bouhoute and K. Boubouh, "Traffic sign recognition using convolutional neural networks," 2017 International Conference on Wireless Networks and Mobile Communications (WINCOM),2017https://ieeexplore.ieee.org/abstract/document/8238205?casa_token=9A9SCG3koHsAAAAA:eV2VaxxEam2Odx82mBihf9gAupLabkR0_vr7BuWyaHWH941uZgwsTpcZOdZgDnl_SW8txiKANu0
[3]M. M. Lau, K. H. Lim and A. A. Gopalai, "Malaysia traffic sign recognition with convolutional neural network," 2015 IEEE International Conference on Digital Signal Processing (DSP), 2015, pp. 1006-1010, doi:10.1109/ICDSP.2015.7252029.https://ieeexplore.ieee.org/abstract/document/7252029?casa_token=wvorFehfCX0AAAAA:1B0TtmktT9v1xY3XVtffGoALgLqJtMyGLuB-O7VwZhY_BzpbRRhmJHfQtJhyJCuZFe-7cR07fpU
[4]Traffic Signs Recognition using CNN and Keras in Python by Shikha Gupta
https://www.analyticsvidhya.com/blog/2021/12/traffic-signs-recognition-using-cnn-and-keras-in-python/