A Comparative Analysis in Classifying Traffic Sign Images through Convolutional Neural Networks and Multilayer Perceptrons

Nidhi Joshi
nidhi.joshi@city.ac.uk

Abstract

This paper conducts a comparative analysis between Convolutional Neural Networks (CNN) and Feedforward Multilayer Perceptron (MLP) models for the task of traffic sign classification using the German Traffic Sign Recognition Benchmark (GTSRB) dataset from Kaggle[1]. Both models are trained by varying respective hyperparameters and further evaluated, to compare their performance in terms of test accuracy, confusion matrices, classification reports and training time. The study concluded CNN to be more effective model for this specific classification task.

1. Introduction

Traffic Sign Recognition (TSR) is a technology that helps drivers by automatically identifying traffic signs, providing crucial road information. It's commonly found in modern vehicles equipped with cameras and assists drivers through image processing. TSR serves as a critical component, especially in self-driving systems like Advanced Driver Assistance Systems (ADAS). The ability to detect and classify traffic signs accurately in such systems, is essential in order to ensure safety and efficiency on the roads[2]. While deep learning techniques like Convolutional Neural Networks (CNNs) are considered ideal for image classification tasks, simpler models like Multilayer Perceptron (MLPs) remain popular for their ease of implementation. The aim of this paper is to critically analyse and compare these two implemented models on a German Traffic Sign Recognition dataset comprising of 43 different classes with over 50,000 images. While section 2 provides with insights into the dataset, section 3 explains the approach towards the implementations. Furthermore, models are compared and critically evaluated as per their respective outcomes in the section 4 with section 5 providing the final inferences.

1.1. Convolutional Neural Network (CNN)

Convolutional neural networks (CNNs) comprise of three fundamental layers: Convolutional, Pooling & Fully-connected. The convolutional layer initiates by processing input data with filters to generate feature maps. These feature maps undergo Rectified Linear Unit (ReLU) activation, introducing nonlinearity to enhance learning. Following this, the pooling layer is to reduce input dimensionality, enhance efficiency and mitigate overfitting risks. Additional convolutional or pooling layers may be added to deepen network complexity. Ultimately, the fully-connected layer performs classification tasks, leveraging extracted features from previous layers to recognize patterns and make predictions accordingly[3].

1.2. Multilayer Perceptron (MLP)

The multilayer perceptron (MLP) is a forward-propagating neural network comprising numerous layers of interconnected artificial neurons, or perceptrons. These layers typically include an input layer, hidden layers (can be one or more), and an output layer. Within each layer, individual neurons receive input signals, conduct computations based on assigned weights and biases, and employ activation functions to yield outputs[4]. During the initial forward pass, output is computed by altering the hidden unit weights input values are altered by the weights linked to hidden units to compute the output. This output is matched against the target output, generating an error signal that is then propagated backwards to adjust the connection weights. Once the

neural network has undergone training, it can generate an output number encoded as a class label corresponding to the input[5].

	CNN[3]	MLP[5]
Pros	◦ Effective at capturing spatial hierarchies in images. ◦ Automatically learns relevant features from input data.	◦ Simple and flexible architecture. ◦ Easy to implement and interpret.
Cons	◦ Can be computationally expensive and require large amounts of training data.	◦ May struggle with capturing spatial dependencies in images. ◦ Prone to overfitting, especially with high-dimensional data.

Table 1: Pros and Cons of CNN and MLP models

2. Dataset

The concerned German Traffic Sign Benchmark, presented at the International Joint Conference on Neural Networks (IJCNN) 2011, offers a significant dataset for assessing single-image classification models. This benchmark comprises over 50,000 images, covering more than 40 different classes of traffic signs, such as speed limits, stop signs, yield signs etc. It serves as a valuable resource due to its extensive and realistic database[1]. On initially analysing the meta data through correlations, it was noticed that even though there are minimal relationships between features, 'ClassId' and 'ColorId' fields indicate some degree of association. Train data matrix, however, shows that there are strong relationships between all features except ClassId.

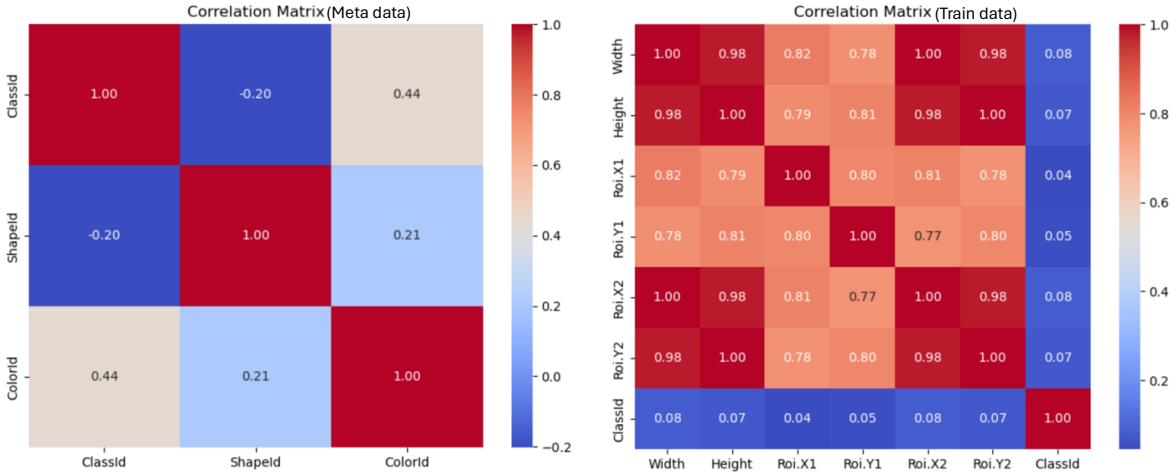

Figure 1: Correlation matrices for Meta data and Train data resp.

2.1. Initial Data Analysis

Apart from plotting pair-wise comparison histograms in the datasets, an image count plot was generated to visualize number of images in each class label. This plot indicated that while 8 classes had more than 2000 images to train, even the other classes had enough images to train our models sufficiently. Also, on examining the dataset further we could see that there was

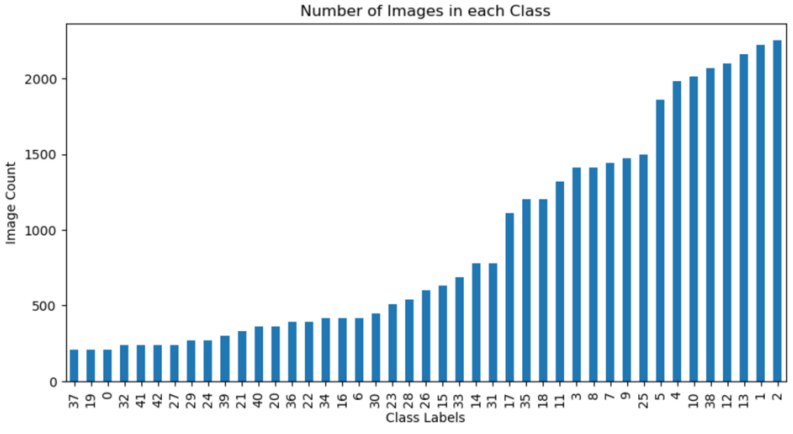

Figure 2: Image count in each Class label

sufficient variance in images available for each class such as noise, lighting conditions, blur etc. So, although the data was robust in itself, data augmentation had to be implemented to convert images into a format which was compatible with pyTorch tensors for further steps. Along with that, RandomHorizontalFlip and Resize transformations were performed as well to enhance the generalization ability of the model. This was done to make it more capable of handling variations within the data and sustaining performance even on the unseen data[6].

3. Methods

This section explains in detail the architecture used to build CNN and MLP models along with the specified hyperparameters. And then how we trained and tested through the given dataset.

3.1. Methodology Analysis

The methodology begins with setting aside 20% of the entire data as a testing set to be evaluated with the best trained CNN and MLP models in the end. Rest of the dataset is partitioned into training and validation sets in a 80:20 ratio to train both our models in the main file and then compare the outcomes after evaluation. A custom DataLoader class facilitates loading and preprocessing of the dataset. Data transformations are applied and then class folders are sorted and indices are taken into account before loading the images. Images are converted into 'RGB' format before being returned along with the respective class labels. Both the datasets are split into batches of 32 for efficient computation resulting in 981 train batches and 246 test batches. Furthermore, there are common functions defined for both the models to be trained (train_model()), evaluate the models (evaluate_model()), images to be displayed to check the evaluated model (disp_images()) and plot the confusion matrices and loss graphs (plot_confusion_matrix() and plot_loss_graphs()). For both the CNN and MLP models, we utilize the Adam optimizer with a learning rate of 0.001 without any weight decay. The best hyperparameter combination was the result of partially performed grid search initially (due to the fact that the dataset is huge, it was taking a very long time to perform grid search and in interest of time, it was interrupted in between) but it was noticed that applying the recommended values (till interruption) of weight decay along with the learning rate (0.001) decreased the test accuracy for both the models somehow. Even manually checking for different weight decay values (0.0001, 0.001, 0.01) didn't yield good results, hence dropped. It is understood that CNN as a model demands larger datasets to train and so weight decay might be limiting its capacity to get trained properly.

The training process involves iterating over the train data loader in batches, computing the model's output. Loss is calculated using the Cross Entropy Loss function, and models' parameters are updated accordingly using backpropagation. This process is repeated for 5 epochs each to ensure convergence. To evaluate the models, we employ a similar procedure for iterating over the test data loader, computing the model's output, and calculating the loss. Additionally, we compute the accuracy of the models on the testing dataset to assess their performance.

3.2. CNN Model Analysis - Architecture and Parameters

This CNN model architecture consists of ***two convolutional layers*** followed by ***max pooling layers***, and then, ***two fully connected layers***. Designed for image classification tasks, it is expecting input images with 3 channels ('RGB'). The first Convolutional layer (self.conv1) is passed 3 input channels, 16 output channels, a 3*3 kernel size with both stride and padding as 1. With 16 filters applied, 16 feature maps are obtained which are then passed on to the second convolutional layer (self.conv2) with the same kernel size, stride and padding as the previous layer. This layer now produces 32 output channels with 32 filters. The next Max pooling layer (self.pool) reduces the spatial dimensions of the feature maps after each convolutional layer to

extract important features. A kernel size of 2*2 is used with a stride of 2 for down sampling. Now, for the fully connected layers, feature maps are flattened and passed. The first fully connected layer has the flattened 32*56*56 input features and 128 output features. This layer applies linear transformation to the input data. The second and the final fully connected layer takes 128 output features from the previous layer to produce the final output with number of output classes equal to number of classes which is passed as a parameter (num_classes). Rectified Linear Unit (ReLU) activation function was used after each layer to introduce non-linearity to the model and help learning complex patterns in the data.[7]

3.3. MLP model Analysis - Architecture and Parameters

The built MLP (Multilayer Perceptron) model is a feedforward neural network architecture consisting of an ***input layer*** that is flattening the input images, ***multiple hidden layers*** with ReLU activation functions, and an ***output layer***. The input layer size consists of 3-channel image ('RGB') with a 224*224 pixel resolution. The nn.Flatten() is reshaping the input tensor into a single dimension before passing on to the hidden layers. Next are two hidden layers with 256 and 128 neurons respectively. The nn.ModuleList() is used to dynamically create a list of linear layers (nn.Linear()) based on the specified hidden layer sizes. Each linear layer represents a fully connected (dense) layer in the neural network. ReLU activation functions are applied to each hidden layer to introduce non-linearity. Finally, the output layer is a linear layer responsible for producing the output logits. These are then passed through a softmax activation function to determine probable classes.[8][9]

4. Results, Findings & Evaluation

Our investigation into the performances of Convolutional Neural Networks (CNNs) and Multilayer Perceptrons (MLPs) for the traffic sign image classification task yielded insightful findings. Here, we present a comprehensive analysis of the model selection process, algorithm comparison, and evaluation metrics.

4.1. Model Selection

After selecting the optimum hyperparameters, epoch-wise training loss graphs for both models depict a steady decline, indicating effective learning during training. Notably, the CNN model demonstrated a test loss reduction from 1.3227 to 0.1714 over five epochs, with a corresponding increase in test accuracy from 88.88% to 88.94%. In contrast, the MLP model exhibited slower convergence, with a final test loss of 0.7889 and a test accuracy of 75.64%.

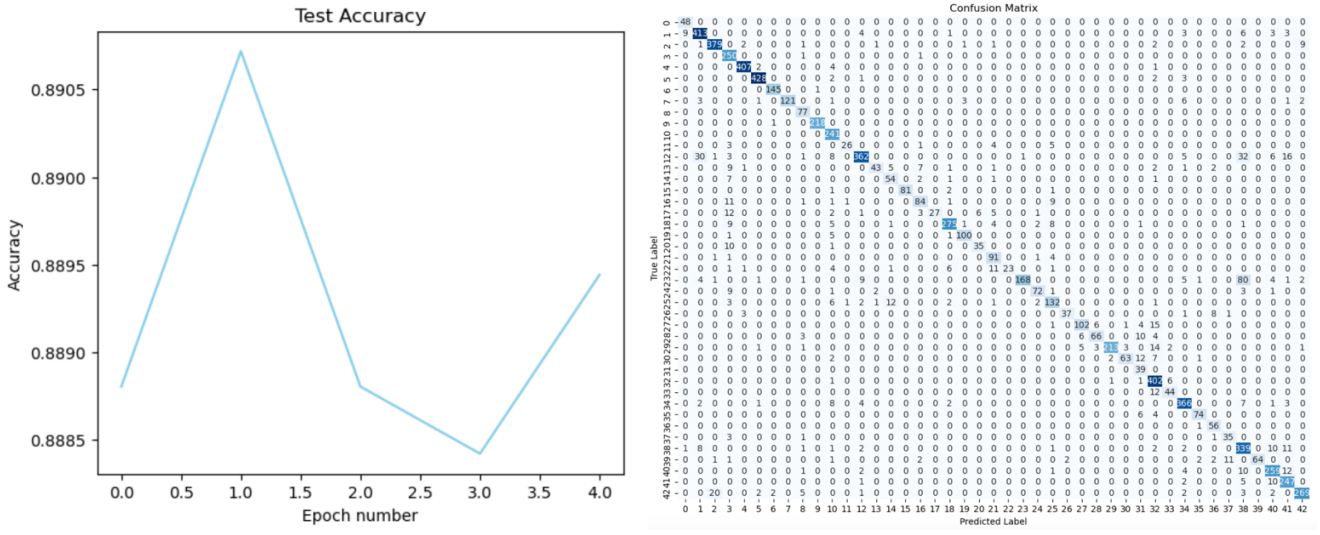

Figure 3a: Test Accuracy and Confusion Matrix plots for CNN model

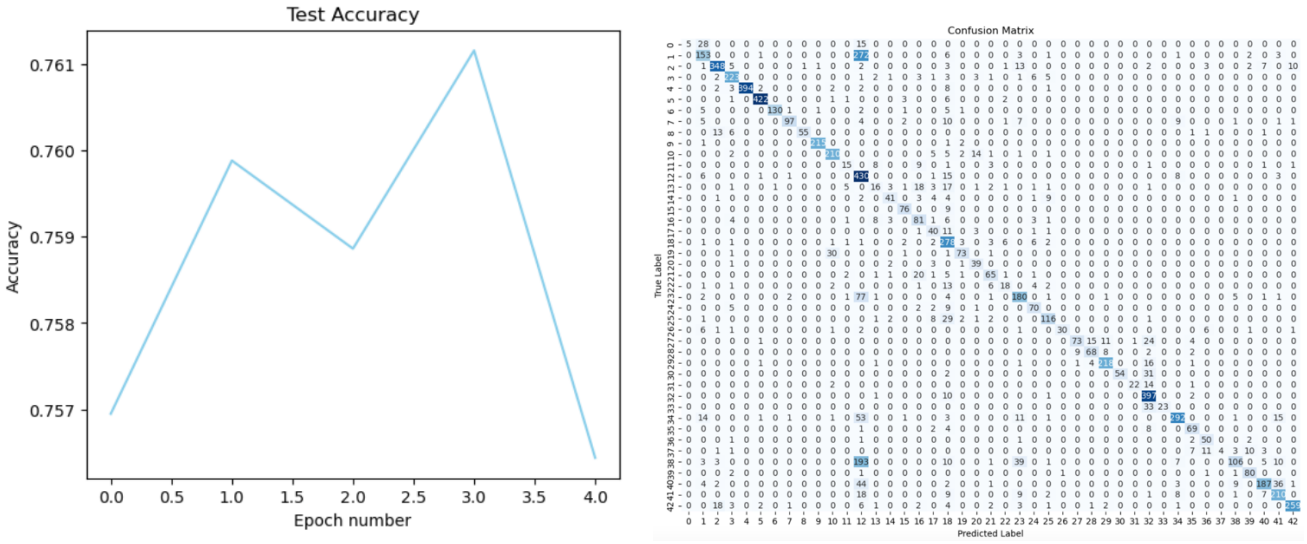

Figure 3b: Test Accuracy and Confusion Matrix plots for MLP model

4.2. Algorithm Comparison

For comparing the performance of CNN and MLP trained models, confusion matrices and classification reports provided detailed insights. Both models displayed a strong diagonal dominance in their confusion matrices, indicative of robust classification capabilities. However, epoch-wise training loss graphs revealed significant differences in training dynamics. The CNN model showcased a rapid reduction in training loss, converging to a lower value compared to the MLP model, suggesting superior learning efficiency. Furthermore, the classification reports highlighted the CNN model's superior precision, recall, and F1-score metrics across all classes, reaffirming its efficacy in traffic sign classification tasks.

4.2. Evaluation Metrics

To delve deeper into the models' performance, we utilized the evaluate_model() function to compute additional evaluation metrics, for example images linked with their predicted and true labels were returned after evaluation. Apart from the other important metrics, the images were displayed in a sample grid of 5*5 to analyse how the models fared as compared to each other. When CNN exhibited a Precision of 0.90, Recall of 0.89 and F1 score of 0.89, it overweighed the fact that CNN took more than double the time to train as compared to MLP which was evaluated with 0.80 Precision, 0.76 Recall and 0.75 F1 score. The CNN model demonstrated consistent improvements in accuracy over epochs, accompanied by a steady decline in test loss. Conversely, the MLP model exhibited slower convergence, with less significant improvements in accuracy. Additionally, the CNN model's higher test accuracy and lower test loss underscore its superior performance compared to the MLP model.

	CNN	MLP
Precision	0.90	0.80
Recall	0.89	0.76
F1 Score	0.89	0.75

Table 2: Performance Metrics of CNN and MLP models

5. Conclusion

Our analysis underscores the efficacy of CNNs over MLPs in case of traffic sign image classification. Despite the much longer training time, CNNs exhibited superior performance, characterized by

higher accuracy and faster convergence. These findings emphasize the importance of leveraging CNNs, for complex image classification tasks like traffic sign recognition to contribute further towards road safety developments. In cases like ours, where we had a huge dataset to train our models, CNN proved to be very efficient because of its ability to automatically extract features required to learn the patterns and efficiently classify them even from a very complex dataset.[10]
Moving forward, further optimizations to the CNN architecture could potentially enhance performance metrics even more. Additionally, exploring novel techniques such as transfer learning and data augmentation may offer avenues for improving model generalization and robustness in real-world scenarios.

6. Lessons learnt and Future work

CNNs demonstrated rapid convergence and lower final training losses compared to MLPs, highlighting their efficacy in capturing complex image features. While CNNs offer superior performance, they also entail longer training times due to their deeper architectures and convolutional operations when compared to the feedforward MLP model. Investigating ensemble learning techniques, such as model averaging or boosting, could potentially combine the strengths of multiple models (e.g., CNNs and MLPs) to achieve superior performance. Ensemble methods may mitigate the weaknesses of individual models and enhance overall prediction accuracy. Furthermore, more advanced architectures such as ResNet and DenseNet can also be explored to improve the model performance.[11]

6. References

[1] Mykola (2018) *GTSRB - German Traffic Sign Recognition Benchmark*, *Kaggle*. Available at: https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign (Accessed: 17 April 2024).
[2] Safe, S. (2023) *What you need to know about traffic sign recognition*. Available at: https://www.newsmartsafe.com/industry-news/traffic-sign-recognition (Accessed: 17 April 2024).
[3] Torres, L.F. (2023) *Convolutional neural network from scratch*, *Medium*. Available at: https://medium.com/latinxinai/convolutional-neural-network-from-scratch-6b1c856e1c07 (Accessed: 17 April 2024).
[4] Everton Gomede, P. (2024) *Unraveling the multilayer perceptron: A journey into neural network excellence*, *Medium*. Available at: https://medium.com/aimonks/unraveling-the-multilayer-perceptron-a-journey-into-neural-network-excellence-db64ea4bc6c5 (Accessed: 17 April 2024).
[5] Bishop, C.M. (1996) *Neural networks for pattern recognition - Christopher M. Bishop - Oxford University press*, *Neural Networks for Pattern Recognition*. Available at: https://global.oup.com/academic/product/neural-networks-for-pattern-recognition-9780198538646?cc=us&lang=en& (Accessed: 17 April 2024).
[6] Mumuni, A. and Mumuni, F. (2022) *Data augmentation: A comprehensive survey of modern approaches*, *Array*. Available at: https://www.sciencedirect.com/science/article/pii/S2590005622000911 (Accessed: 17 April 2024).
[7] cs231n (no date) *Convolutional Neural Networks for Visual Recognition*, *Github*. Available at: https://cs231n.github.io/convolutional-networks/ (Accessed: 17 April 2024).
[8] Brownlee, J. (2019) *How to configure the number of layers and nodes in a neural network*, *MachineLearningMastery.com*. Available at: https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/ (Accessed: 17 April 2024).
[9] AIML (2024) *What is a multilayer perceptron (MLP) or a Feedforward Neural Network (FNN)?*, *AIML.com*. Available at: https://aiml.com/what-is-a-multilayer-perceptron-mlp/ (Accessed: 17 April 2024).
[10] Naresh, R.K. (2023) *Why CNN is better than SVM for Image Classification and which is better for image classification machine learning or Deep learning? | Researchgate*, *ResearchGate*. Available at: https://www.researchgate.net/post/Why_CNN_is_better_than_SVM_for_image_classification_and_which_is_better_for_image_classification_machine_learning_or_deep_learning (Accessed: 17 April 2024).
[11] Alzubaidi, L. *et al.* (2021) *Review of Deep Learning: Concepts, CNN Architectures, challenges, applications, future directions - Journal of Big Data*, *SpringerOpen*. Available at: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00444-8 (Accessed: 17 April 2024).

Appendix 1 – Glossary

- ***Convolutional Neural Networks (CNN)*** - A type of artificial neural network used primarily for image recognition and processing, due to its ability to recognize patterns in images. (Source)
- ***Feedforward Multilayer Perceptron (MLP)*** - A type of feedforward neural network consisting of fully connected neurons with a nonlinear kind of activation function. (Source)
- ***Confusion matrix*** - A specific table layout that allows visualization of the performance of an algorithm (Source)
- ***Classification report*** - The classification report shows a representation of the main classification metrics on a per-class basis. (Source)
- ***Traffic Sign Recognition (TSR)*** - A technology that helps drivers by automatically identifying traffic signs, providing crucial road information. It's commonly found in modern vehicles equipped with cameras and assists drivers through image processing. (Source)
- ***Advanced Driver Assistance Systems (ADAS)*** - Technologies that assist drivers with the safe operation of a vehicle through a human-machine interface (Source)
- ***Rectified Linear Unit (ReLU)*** - The rectifier activation function introduces the property of nonlinearity to a deep learning model and solves the vanishing gradients issue. (Source)
- ***Adam optimizer*** - Short for “Adaptive Moment Estimation,” it is an iterative optimization algorithm used to minimize the loss function during the training of neural networks. (Source)
- ***Learning rate*** - A tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function. (Source)
- ***Weight Decay*** - Also known as L2 regularization, it helps prevent overfitting in machine learning models by adding a penalty term to the loss function. (Source)
- ***Cross Entropy Loss function*** - Also known as logarithmic loss or log loss, it is a popular loss function used in machine learning to measure the performance of a classification model. (Source)
- ***Softmax activation function*** - The function transforms the raw outputs of the neural network into a vector of probabilities, essentially a probability distribution over the input classes. (Source)
- ***Accuracy*** - Refers to how close a measurement is to the true or accepted value. (Source)
- ***Precision*** - Refers to how close measurements of the same item are to each other. (Source)
- ***Recall*** - The measure of our model correctly identifying True Positives. (Source)
- ***F1 Score*** - Measures a model's accuracy by combining the precision and recall scores of a model. (Source)
- ***ResNet*** - A residual neural network (also referred to as a residual network or ResNet) is a seminal deep learning model in which the weight layers learn residual functions with reference to the layer inputs. (Source)
- ***DenseNet*** - A type of convolutional neural network that utilises dense connections between layers, through Dense Blocks, where we connect all layers (with matching feature-map sizes) directly with each other. (Source)

Appendix 2 –

Implementation details

- Dataset Preparation
  The German Traffic Sign Recognition Benchmark dataset which included more than 50,000 images distributed across 43 classes of traffic signs was analysed and then, split into training and testing sets to facilitate model training and further evaluation.
- Model Architectures
  Two primary deep learning architectures used in the project: Convolutional Neural Networks (CNNs) and Multilayer Perceptrons (MLPs).
  - **CNNs:** The chosen CNN architecture consists of two convolutional layers followed by max-pooling layers for downsampling, and then two fully connected layers for classification in the end. Initially, we tried a base model with just one convolutional layer but adding another layer significantly improved the performance with some adjustments to Kernel size and stride. The input features were flattened and increased to pass on higher output features to the next fully connected layer.
  - **MLPs:** Initially only the output size was passed to the model to obtain base results. Then, input size was added along with introducing multiple hidden layers which led to better results. Input tensor was flattened in the beginning itself to enable multidimensional input processing.
  - **Common enhancements:** Two transformations were added to the image data - Random horizontal flipping to address overfitting and resizing them to a fixed 224*224 size for consistency. Images were also converted to 'RGB' format to maintain consistent color channels across. Learning rate of 0.01 and 0.1 were applied initially to the optimiser but 0.001 proved to be the best learning rate.

Trials and Negative Results

- SVC model was also explored to be trained on the aforementioned dataset with 'rbf' kernel and different values of 'C' but the obtained test accuracy proved to be very low, hence instead, MLP was explored with multiple hidden layers.
- HOG (Histogram of Oriented Gradients) feature was implemented while training CNN model to further improve the performance, but it increased the training time by a lot to reproduce results, hence dropped.
- Different values of Weight decay were applied to the optimiser to further improve the models' performances but even with different weight decay values (0.0001, 0.001, 0.01) paired with different combinations of learning rates and batch sizes, it had an opposite effect on the test accuracies. So, the parameter was skipped.