**Group**

- Dubai CW PG Thursday Group 6

**Group Members**

- Mohamed Aman | Pratibha Yadubanshi | Faizan Watare | Fardeen Khan

# F21DL Coursework Part 4 – Neural Networks and CNN

In this report, we explored the performance of linear classifier and conducted experiments with Multilayer Perceptron (MLP). The objectives included assessing the linear classifier's generalization, experimenting with various MLP parameters, and drawing conclusions. Then we perform CNN on the dataset and repeat the same experiments and drew conclusions.

## Using Logistic Regression classifier on our dataset

We ran a Logistic Regression classifier on our dataset and noted its performance using both train-test split and 10-fold cross-validation. It was further tested on the testing dataset.

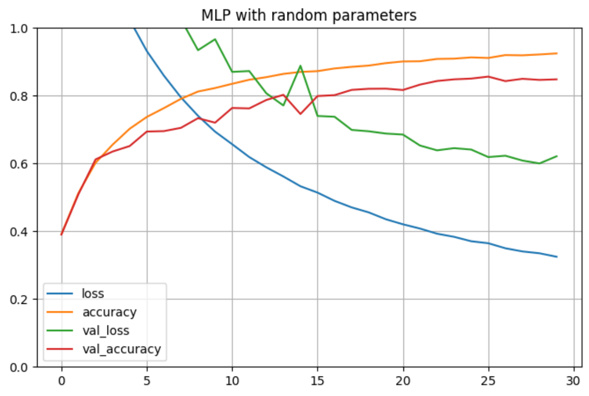
| | 10-Fold Cross Validation on training set | Train Test Split On training test | Testing dataset |
|---|---|---|---|
| Accuracy | 92.7% | 92.8% | 87.8% |
| Precision | 92.7% | 92.9% | 88.0% |
| Recall | 92.7% | 92.8% | 87.8% |
| F1 Score | 92.6% | 92.8% | 87.3% |

The logistic regression showed good performance on the training set, achieving high accuracy, precision, recall, and F1 Score. The model also generalized well to the testing dataset, though there was a slight drop in performance. Dataset appears to be linear separable; this may be due to certain features in images might contribute to the uniqueness of various classes.

## Using Multilayer Perceptron (MLP)

We conducted an initial assessment of Multilayer Perceptron (MLP) by running random parameters to establish a baseline accuracy. The following configuration was used, resulting in an accuracy of **86%:**

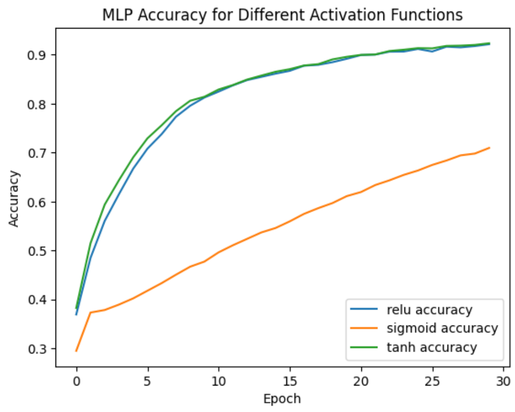
- Activation Function: tanh
- Optimizer: SGD
- Batch size = 50
- Epochs = 30
- Layers = 2
- Neurons = 100

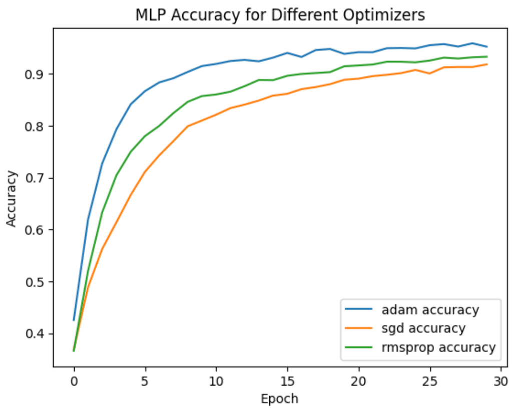


**Manual Hyperparameter Tuning in MLP**

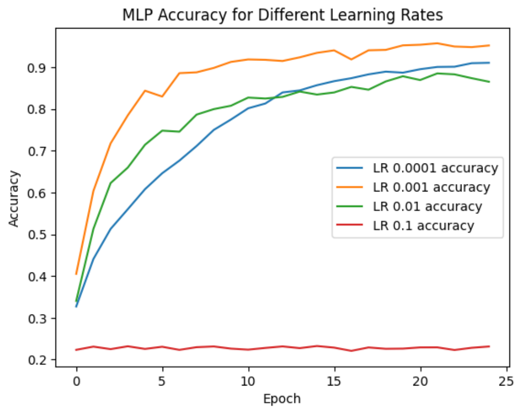
To enhance the Multilayer Perceptron's (MLP) performance, we experimented various hyperparameter tuning such as activation functions, optimizers, batch size and learning rates etc... Initially, we started with activation functions and optimizers. We used all the parameters similar to baseline  changing only activation functions and optimzers. Following activation function and optimizer performed well in our model.

- Activation Function: ReLu
- Optimizer: Adam



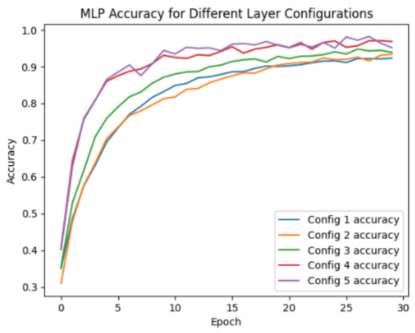


Then we checked the accuracy with varying batch sizes and found that batch sizes between 64 and 128 gives a higher accuracy compared to others. We selected a batch size of 100 for following experiments, recognizing it as an ideal balance.



| Batch size | Batch Size 32 |
|---|---|
| 32 | 88.3% |
| 64 | 87.2% |
| 128 | 88.5% |
| 256 | 85.9% |

Then we checked different layers and changing the number of neurons in each layer at the same time. Based on the configurations mentioned below and the resulting graph, we can see that the best performance is given by the model with 2 layers and 100 neurons for each.



| Configs | Neurons in Layer 1 | Neurons in Layer 2 | Neurons in Layer 3 |
|---|---|---|---|
| Config 1 | 25 | - | - |
| Config 2 | 50 | 25 | - |
| Config 3 | 25 | 50 | - |
| Config 4 | 100 | 100 | - |
| Config 5 | 200 | 200 | 200 |

Then We experimented the impact of different epochs on accuracy. The results indicated that accuracy peaked at Epoch 25, achieving an outstanding 90.06% accuracy.

| | Epoch 10 | Epoch 25 | Epoch 50 | Epoch 100 |
|---|---|---|---|---|
| Accuracy | 84.76% | 90.06% | 85.92% | 88.32% |

Based on the insights gained from each stage of manual hyperparameter tuning, we created a final model. The accuracy of this tuned model reached an impressive **89.4%,** indicating the effectiveness of systematic hyperparameter adjustments.

**Systematic tuning using Keras Tuner**

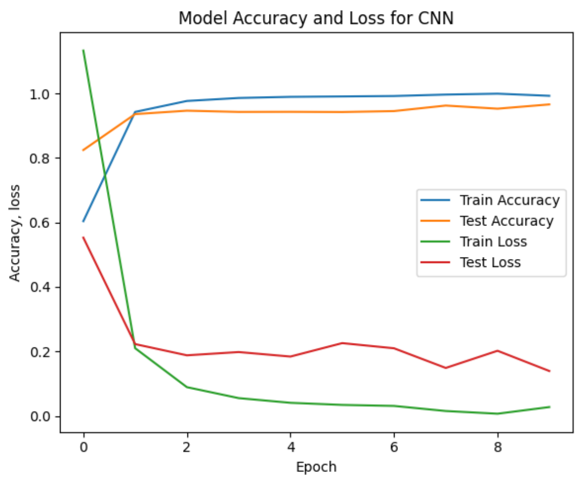
We systematically tuned our Multilayer Perceptron (MLP) model using Keras Tuner. The outcome of Trial 3, which generated the best hyperparameters, is summarized below:

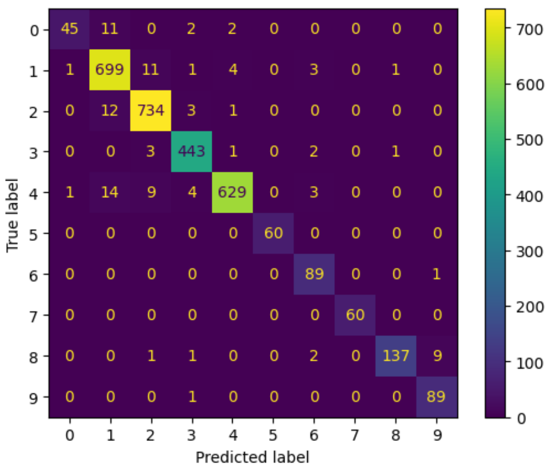
- Layers: 1
- Units: 96
- Score: 0.8895

With the identified best hyperparameters, we trained the model and achieved a remarkable accuracy of 88.71%. This underscores the effectiveness of systematic tuning using Keras Tuner in enhancing the model's performance.

**CNN**

As extra research step, we implemented Convolutional Neural Network (CNN). The CNN architecture consisted of two Conv2D layers with 32 and 64 filters, each followed by MaxPooling2D layers. The model included a dense layer with 128 neurons and an output layer with 10 neurons. We achieved an exceptional accuracy of nearly **97%**.





| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 45 | 11 | 0 | 2 | 2 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 699 | 11 | 1 | 4 | 0 | 3 | 0 | 1 | 0 |
| 2 | 0 | 12 | 734 | 3 | 1 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 3 | 443 | 1 | 0 | 2 | 0 | 1 | 0 |
| 4 | 1 | 14 | 9 | 4 | 629 | 0 | 3 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 60 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 89 | 0 | 0 | 1 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 60 | 0 | 0 |
| 8 | 0 | 0 | 1 | 1 | 0 | 0 | 2 | 0 | 137 | 9 |
| 9 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 89 |

**Conclusion of Part 4**

Logistic Regression showed good performance on the training set, but experienced a slight decrease in accuracy on the testing dataset. However, the high score indicates the linearity of the dataset, suggesting that distinct image features contribute to class separability. The Multilayer Perceptron (MLP) underwent manual hyperparameter tuning and the optimal configuration involved ReLU activation and Adam optimizer. Tuning with Keras-Tuner validated these results, achieving an accuracy of 88.71%.

Convolutional Neural Network (CNN) outperformed other models, achieving an exceptional accuracy of nearly **97%**. The CNN architecture, featuring two Conv2D layers, MaxPooling2D layers and a dense layer with 128 neurons showed the best performance, particularly well-suited for image classification tasks.

# F21DL Coursework Part 5 – Research Question

**Question**
Addressing imbalance in traffic sign image classification, a comprehensive exploration of balancing techniques and their impact on model performance.

**Solution**
Application of various balancing techniques like ROS, RUS, SMOTE, SMOTEENN, Augmentation etc on the training data using different classifiers like Random Forest, Decision Tree, Logistic Regression etc. Verifying how the balanced model generalizes to the unseen environments. Also, experimented with Augmentation using Keras ImageDataGenerator, PCA & CNN on balanced data.

**Performance Metrics**
Below table summarizes the results of Random Forest Classifier after application of various balancing techniques in both the training and test environment.

| | 10-Fold CV on training set | On Testing Dataset |
|---|---|---|
| **Initial dataset** | 97.8 % | 77.5% |
| **ROS** | 99.4 % | 77 % |
| **RUS** | 90.7% | 65.5% |
| **SMOTE** | 99.3% | 77.5% |
| **SMOTE + Under Sampling(50%)** | 98.2% | 75.8% |
| **SMOTEENN** | 99.5% | 75.3% |

**Conclusion**
Application of various balancing techniques on the traffic sign image dataset gave exemplary results in the training environment, but the model did not generalize well in the test environment. Possible reasons could be the complexity of the image data which was our scope of research. SMOTE technique gave us the best results. Also, implementation of CNN using augmented samples demonstrated remarkable accuracy.

- Github Link: https://github.com/amannuhman/F21DL-CW