COMP534 – Applied Artificial Intelligence
MLP & CNNs– Report.

Bharathidasan Ilango – 201670038

Introduction

Neural networks have attracted substantial interest in recent years because to their outstanding performance in numerous tasks, such as image classification, natural language processing, and voice recognition. In this study, we aim to investigate and evaluate the efficacy of two famous neural network models, namely the multilayer perceptron (MLP) and convolutional neural networks (CNNs). We applied several strategies, such as adaptive learning rate, activation functions, optimizers, batch normalization, L1&L2 regularization, and dropout, to train these models and attain the best possible performance. By using these strategies, we intend to attack frequent issues in training neural networks, such as overfitting, underfitting, and unstable gradient. Through this analysis, we aim to get a greater knowledge of these strategies and their effect on neural network performance.
To construct our model, we will utilise Python programming language and many tools for creating and training deep learning models. The two deep learning algorithms employed are multilayer perceptron (MLP) and convolutional neural networks (CNNs). The development method encompasses the following phases, Data loading and Visualization, Data pre-processing, Model training and hyperparameter tuning and Model assessment.

The libraries imported are:

- Tensorflow and Keras for building and training deep learning models
- PyTorch and Torchvision for machine learning tasks
- Numpy and Pandas for data manipulation and analysis
- Matplotlib for data visualization
- Scikit-learn for machine learning tasks, including model evaluation and tuning.

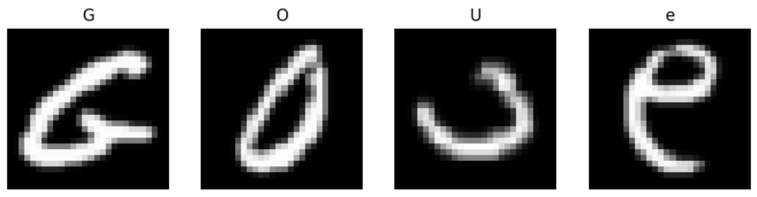
WE have imported various layers and utilities from the TensorFlow library, including Dense, Flatten, Conv2D, MaxPooling2D, Dropout, BatchNormalization, and Activation. Additionally, we have imported other utilities such as LearningRateScheduler and EarlyStopping from tensorflow.keras.callbacks.
The EMNIST (Extended MNIST) is an enhancement of the original MNIST However, The EMNIST Balanced dataset, comprises both handwritten numbers and handwritten uppercase and lowercase letters, resulting in a more diversified and complex dataset. The EMNIST Balanced dataset incorporates a total of 131,600 photographs, comprising 112,800 images for training and 18,800 images for testing. The remaining photos are separated into proportionate subsets of uppercase letters, lowercase letters, and numerals. The dataset is balanced, meaning that each class has nearly the same number of samples, which makes it acceptable for training and assessing machine learning models in a multi-class classification problem.
The images in the EMNIST Balanced dataset are 28x28 pixels in size, and each pixel is represented by an integer number ranging from 0 to 255, reflecting the grayscale intensity. The dataset is frequently used for tasks such as character identification, digit recognition, and letter recognition, and functions as a baseline for assessing the performance of various machine learning techniques and models.

MLP MODEL STRUCTURE:

The MLP (Multi-Layer Perceptron) model is developed using the Keras programme in Python, with hyperparameters adjusted using a RandomSearch tuning. It begins with a 'Flatten' layer to transform the input photos from 28x28 pixels to a flat vector. Then, add dense (completely linked) layers to the model. The number of units in each dense stratum is specified by the 'units' hyperparameter. 'relu' activation function gets used in these dense layers, and 'BatchNormalization' and 'Dropout' layers are added after each dense layer to enhance model performance and prevent overfitting. Finally, a dense layer with 'softmax' activation function is inserted as the output layer with 47 units.

CNN MODEL STRUCTURE:
The CNNs include numerous convolutional and pooling layers, followed by dense layers for classification. The CNNs commence with a convolutional layer using ‘relu’ activation function, followed by batch normalization and max pooling to extract features from the input pictures. The number of filters in the convolutional layers is defined by the hyperparameter. The CNNs may comprise 2 to 5 convolutional layers, specified by the hyperparameter. Each convolutional layer is followed by batch normalization and max pooling. Additionally, a dropout layer with a dropout rate given by the hyperparameter and it is inserted after each convolutional layer to avoid overfitting. After the convolutional layers, the feature maps are normalised and transmitted through dense layers using ‘relu’ activation function. The number of units in the dense layers is defined by the hyperparameter. Another dropout layer with a dropout rate set by the hyperparameter and it is added before the output dense layer for regularization.

Mapping and Visualization:
The file 'emnist-balanced-mapping.txt' is read and built a dictionary named 'mapping' from the contents of the file. The file is read line-by-line, with each line holding two integers separated by a space. The first number represents the numerical index of a class in the EMNIST dataset, and the subsequent number is the associated ASCII code of the character denoted by that class.
We used a dictionary comprehension to generate the 'mapping' dictionary. Each key-value combination in the dictionary corresponds to an index-ASCII code pair from the input file. To show a few examples from the EMNIST dataset, we made use of matplotlib to construct a 4x1 grid of pictures. We then obtained the appropriate picture and label from the dataset. To make the labels more visible, we utilised the mapping dictionary to turn the label integer into its equivalent ASCII character.



Hyperparameters and Techniques that are used to tune and explore:
In this project, we studied numerous strategies for training neural network models, including adaptive learning rate, activation functions, optimizers, batch normalization, L1/L2 regularization, and dropout. We set up TensorFlow Keras library for devising these strategies.

Adaptive Learning Rate: (Step decay, Exponential decay): Two varieties of learning rate schedulers that may be employed are step decay and exponential decay. The step decay approach decreases the learning rate by a factor after a particular number of epochs, whereas the exponential decay method decreases the learning rate drastically after each epoch. This will aid in obtaining greater efficacy and speedier convergence.
Optimizers: (SGD, ADAM, RMSprop): The model was trained using three distinct optimizers: SGD, Adam, and RMSprop. The training efficacy of each optimizer was evaluated on the training data. Adam optimizer got the greatest training accuracy of 82%, followed by RMSprop at 80% and SGD at 72%.

Activation function: (ReLU, Sigmoid, Tanh): The model was trained using three activation functions: ReLU, sigmoid, and tanh. The validation accuracy for every single function was 82.44%, 79.69%, and 82.34%, respectively. The ReLU activation function obtained the highest validation accuracy. The associated validation loss for each of the functions was 0.5922, 0.6336, and 0.5436, respectively. The lowest validation loss was attained using the tanh activation function.
Batch Normalization: To investigate the influence of Batch Normalization on neural network models, two models have been trained - one with Batch Normalization and one without. The model with Batch Normalization obtained a validation loss of 0.5887 and a validation accuracy of 0.8211, whereas the model without Batch Normalization had a validation loss of 0.5977 and a validation accuracy of 0.8166. The findings demonstrate that Batch Normalization may enhance the efficacy of neural network models.
L1, L2 Regularization: To investigate the impacts of L1 and L2 regularization, three choices were tested: without regularization, with L1 regularization, and with L2 regularization. The model without regularization obtained a validation loss of 0.6167 and validation accuracy of 0.8183. The model with L1 regularization generated a markedly increased validation loss of 4.5103 and validation accuracy of just 0.0197. Finally, the model using L2 regularization obtained a validation loss of 1.2835 and validation accuracy of 0.7510.
Dropout: Two models were devised to evaluate the impact of utilising Dropout in a neural network. The model with Dropout had a higher validation accuracy of 82.44% and a validation loss of 0.5922. Instead, the model lacking Dropout layer had an inferior validation accuracy of 79.69% and a greater validation loss of 0.6336. Dropout regularization may assist to avoid overfitting by arbitrarily leaving out neurons during training, which drives the model to acquire more robust characteristics and minimises the dependency on any neuron.

Hyperparameters Used: -
The hyperparameters are selected for the hyperparameter search (RandomSearch) are learning rate, dropout rate, number of layers, and units. Learning rate regulates the step size in modifying the weights during training, and differing learning rates may influence the convergence speed and accuracy of the model. This helps discover the optimum collection of hyperparameters that enhance the model's performance on the validation data.
Learning rate scheduler: An exponential decay learning rate scheduler is utilised, which reduces the learning rate over time during training. This enables the model to fine-tune its weights more effectively during successive epochs, perhaps leading to higher convergence and generalization performance.
Activation function: RELU owing to its capacity to alleviate the vanishing gradient issue and enhance the training of complex models. ReLU introduces non-linearity into the model, which enables the network to learn sophisticated and nonlinear interactions between inputs and outputs. ReLU is computationally efficient and helps mitigate the vanishing gradient problem, which occurs in other activation functions like sigmoid or tanh.
Optimization algorithm: Adam modifies the learning rate for each parameter based on the preceding gradient information, which enables it to autonomously alter the learning rate throughout training. This may assist expedite convergence and enhance optimization performance, particularly in problems with sparse or chaotic gradients. Adam is reasonably resilient to the choice of hyperparameters compared to other optimization algorithms, such as learning rate and momentum, which makes it simpler to use and less susceptible to hyperparameter tuning.
Early stopping: EarlyStopping callback is used to monitor the validation loss during training and terminate training early if the validation loss does not progress for a given number of epochs (in this example, 5 epochs). This helps avoid overfitting and saves training time.

Issues While Training the Models: -

Overfitting and underfitting are major challenges in training neural network models. Overfitting happens when the model understands the training data too effectively but fails to apply to new data. On the other side, underfitting happens when the model is too simplistic and fails to reflect the complexity of the data. To cope with overfitting, we applied strategies like as regularization (L1/L2, Dropout), early halting, or data augmentation. These strategies serve to avoid the model from remembering the training data and increase generalization to fresh input. To cope with underfitting, we enhanced the model complexity (adding additional layers, increasing the number of neurons), and improving the hyperparameters (learning rate, optimizer, activation function). We kept track of the training and validation loss/accuracy during model training to discover and fix any overfitting or underfitting concerns.

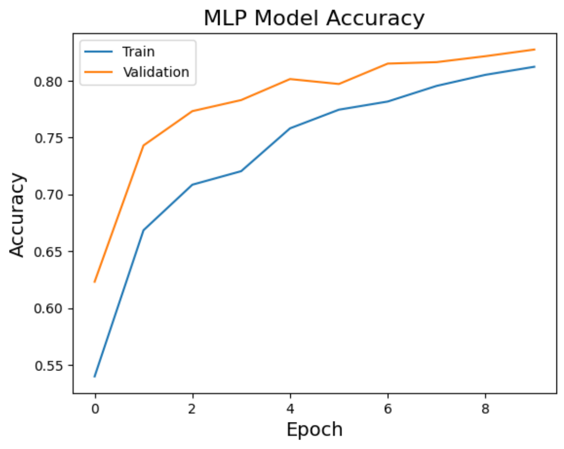
After investigating each technique separately, the most effective methods in each technique are:

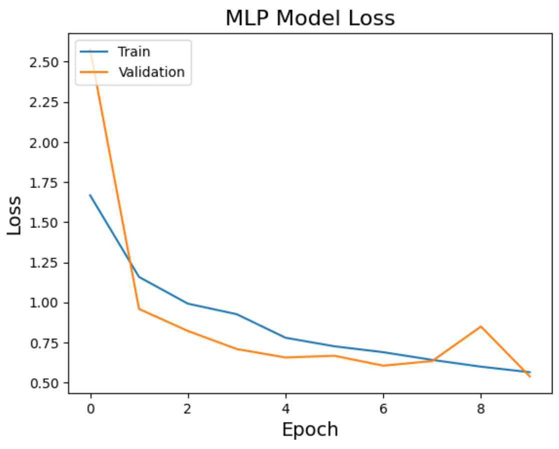
- Adaptive Learning Rate: In learning rate schedulers exponential decay.
- Activation function: ReLU
- Optimizers: ADAM
- Batch Normalization: With Batch normalization
- L1 & L2: With L2 regularization
- Dropout: With Dropout, these techniques are used to build the final models.

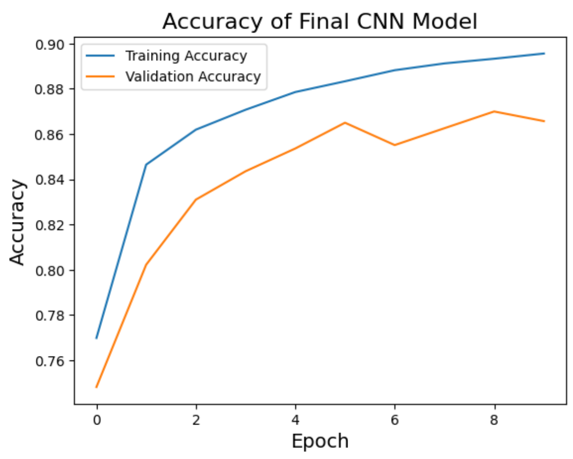
The training loss, Testing loss and Testing accuracy of MLP: -

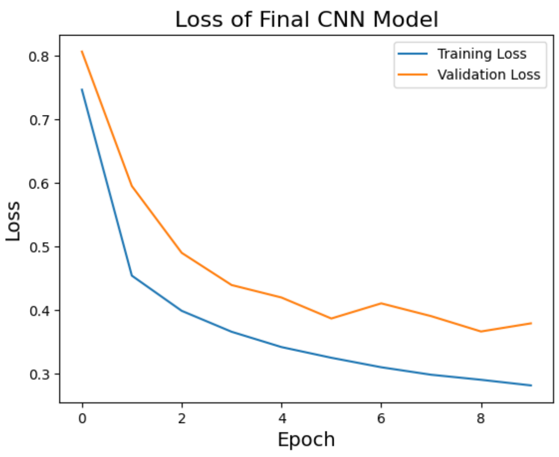
The training loss of my MLP (Multi-Layer Perceptron) model is 0.6129, while the training accuracy is 0.8056. The testing loss is around 0.612917, and the testing accuracy is around 0.80558.

Loss is a measure of how well the model is doing during training, with lower values indicating greater performance. The training loss of 0.6129 implies that on typical, the model's predictions in training were out by around 0.6129 units compared to the actual target values. Similarly, the testing loss of 0.612917 shows that the model's predictions during testing were wrong by the same amount. **Accuracy** evaluates the proportion of true predictions provided by the model. The training accuracy of 0.8056 suggests that during training, the model accurately predicted the target values for about 80.56% of the data. Similarly, the testing accuracy of 0.80558 implies that during testing, the model accurately predicted the target values for the same proportion of samples.









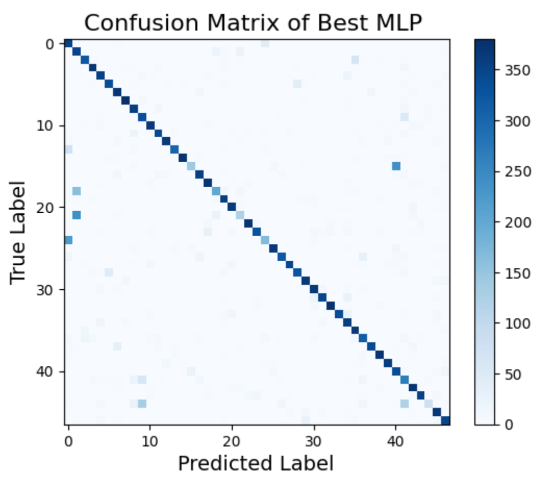
The training loss, Testing loss and Testing accuracy of CNN: -

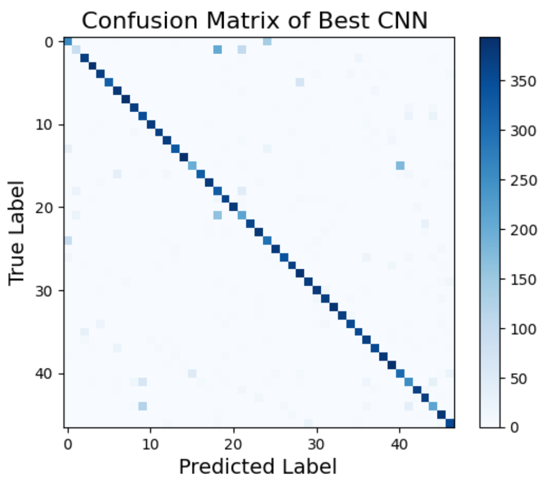
The training loss of our CNN (Convolutional Neural Network) model is 0.4026, whereas the training accuracy is 0.8632. The testing loss is 0.40258, and the testing accuracy is 0.86324. Overall, our CNN model appears to have achieved an excellent performance in terms of accuracy, with both training and testing accuracies around 86%. Additionally, the low loss values of 0.4026 and 0.40258342027664185 imply that the model's predictions were quite near to the actual target values throughout both training and testing.

Pros and Cons of MLP:
MLP is a basic and easy-to-understand neural network design. MLP is excellent for short datasets or issues that do not need extensive feature extraction. MLP may be used for a broad variety of applications such as classification, regression, and even time-series prediction.

MLP lacks the capacity to capture spatial and local patterns in data, which might be significant in applications like as image recognition. MLP tends to overfit readily, particularly with big datasets, because to its enormous number of trainable parameters. MLP may not function well on complicated data with several levels of abstraction.

Pros and Cons of CNN:
CNNs are developed for image identification and other applications that entail the extraction of spatial and local patterns. CNNs employ convolutional and pooling layers that can automatically acquire essential features from data, eradicating the need for human feature engineering. CNNs have fewer trainable parameters compared to MLP, lowering the hazard of overfitting.
CNNs may be more sophisticated and difficult to comprehend compared to MLP. CNNs may need larger datasets to train effectively, particularly when dealing with deep architectures.





CONCLUSION:

According to our analysis we can conclude that CNN model is better than MLP model because, CNNs are often better appropriate for image recognition or other applications with spatial and local pattern recognition needs because to their distinctive architectural design. CNNs employ convolutional layers that are capable of automatically learning local patterns or features from the input data, such as edges, corners, and textures, using convolution operations. Additionally, pooling layers in CNNs assist to minimise spatial dimensions while keeping critical information, enabling the network to record hierarchical representations of the input data.