Applied AI Assignment-2

Karthik Upadhyayula 201626620

Surya Prakash Tirukami 201630546

The EMNIST dataset is a set of handwritten character digits derived from the NIST Special Database 19 and converted to a 28x28 pixel image format and dataset structure that directly matches the MNIST dataset.

Format of the EMNIST Dataset

There are six different splits provided in this dataset and each are provided in two formats:

1. Binary (see emnist_source_files.zip)
2. CSV (combined labels and images)
    - Each row is a separate image
    - 785 columns
    - First column = class_label (mappings.txt gives ASCII values for class label definitions)
    - Each column after represents one pixel value (784 total for a 28 x 28 image)

We used EMNIST balanced dataset split for this project.

Balanced dataset

The EMNIST Balanced dataset is meant to address the balance issues in the ByClass and ByMerge datasets. It is derived from the ByMerge dataset to reduce mis-classification errors due to capital and lower case letters and also has an equal number of samples per class. This dataset is meant to be the most applicable.

- train: 112,800
- test: 18,800
- total: 131,600
- classes: 47 (balanced)

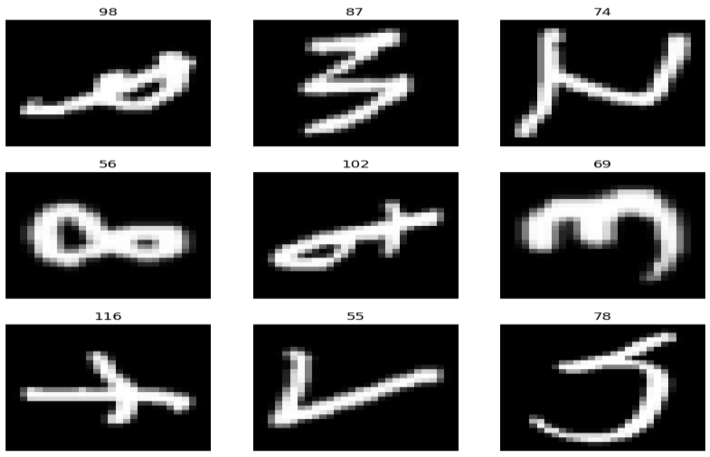
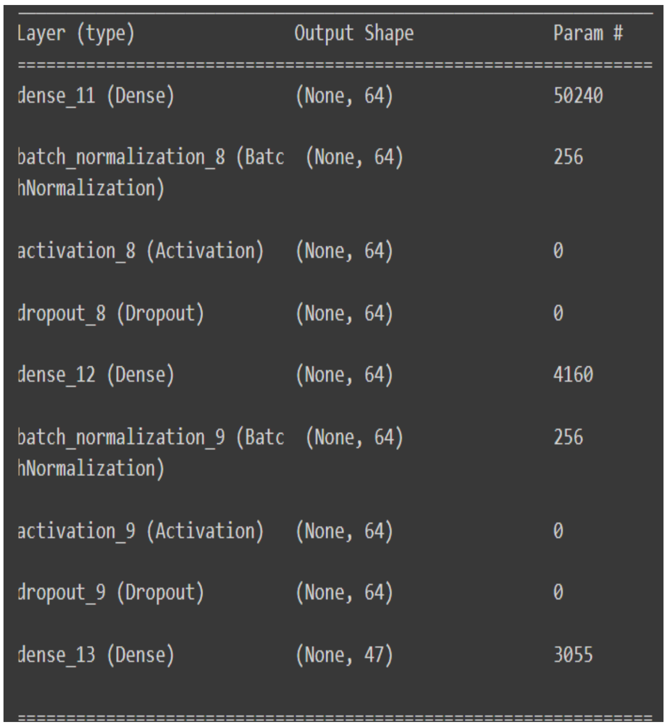


Figure 1 Sample Images of the EMNIST dataset



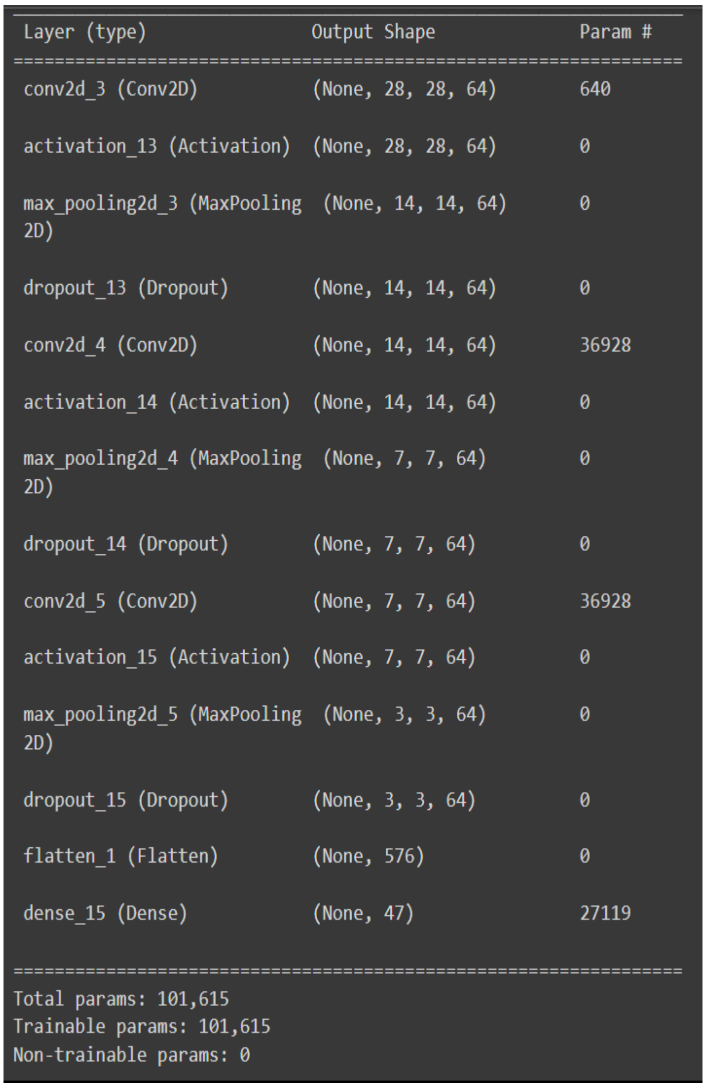
```
Layer (type)                Output Shape              Param #
=================================================================
dense_11 (Dense)            (None, 64)                50240

batch_normalization_8 (Batc  (None, 64)               256
hNormalization)

activation_8 (Activation)   (None, 64)                0

dropout_8 (Dropout)         (None, 64)                0

dense_12 (Dense)            (None, 64)                4160

batch_normalization_9 (Batc  (None, 64)               256
hNormalization)

activation_9 (Activation)   (None, 64)                0

dropout_9 (Dropout)         (None, 64)                0

dense_13 (Dense)            (None, 47)                3055

=================================================================
```

Figure 2 Structure of MLP

It has three dense layers (dense_11, dense_12, and dense_13) for classification. The first dense layer has 64 neurons and is followed by a batch normalization layer (batch_normalization_8) that normalizes the output of the previous layer. The output of the batch normalization layer is then passed through an activation function (activation_8) and a dropout layer (dropout_8) that randomly sets a fraction of the inputs to 0 during training to prevent overfitting.

The second dense layer also has 64 neurons and is followed by another batch normalization layer (batch_normalization_9), activation function (activation_9), and dropout layer (dropout_9).

Finally, the output of the last dense layer is passed through a third dense layer with 47 neurons, which produces the final output for classification.



```
Layer (type)                Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)           (None, 28, 28, 64)        640

activation_13 (Activation)  (None, 28, 28, 64)        0

max_pooling2d_3 (MaxPooling  (None, 14, 14, 64)       0
2D)

dropout_13 (Dropout)        (None, 14, 14, 64)        0

conv2d_4 (Conv2D)           (None, 14, 14, 64)        36928

activation_14 (Activation)  (None, 14, 14, 64)        0

max_pooling2d_4 (MaxPooling  (None, 7, 7, 64)         0
2D)

dropout_14 (Dropout)        (None, 7, 7, 64)          0

conv2d_5 (Conv2D)           (None, 7, 7, 64)          36928

activation_15 (Activation)  (None, 7, 7, 64)          0

max_pooling2d_5 (MaxPooling  (None, 3, 3, 64)         0
2D)

dropout_15 (Dropout)        (None, 3, 3, 64)          0

flatten_1 (Flatten)         (None, 576)               0

dense_15 (Dense)            (None, 47)                27119

=================================================================
Total params: 101,615
Trainable params: 101,615
Non-trainable params: 0
```

Figure 3 Structure of CNN

The structure of the CNN has the following,

Three convolutional layers—conv2d_3, conv2d_4, and conv2d_5—followed by activation functions—activation_13, activation_14, and activation—and max pooling layers—max_pooling2d_3, max_pooling2d_4, and max_pooling2d_5—reduce the spatial dimensions of the feature maps that up the CNN.

A portion of the input units are randomly set to 0 during training in the dropout_13, dropout_14, and dropout_15 layers to avoid overfitting.

The final output for classification is created by flattening the output of the last pooling layer using the flatten_1 layer and then passing it through a fully connected (Dense) layer of 47 neurons.

II. The project involved using Random Search Cv as a technique for getting the best hyperparameters of both the models.

At the beginning of the project GridSearchCV was used. Due to the limited computational resources, and as the project involved tuning a number of Hyperparameters, RandomSearch CV was preferred.

Multi-Layer Perceptron

The hyperparameter that was chosen for the MLP model were using, Batch normalization layer, changing the regularizer, Changing the type of Optimizer being used, number of neurons in each layer, changing the number of hidden layers, changing the activation function and learning rate of the optimizers.

The best hyperparameters of the MLP model was found by using Random Search CV

Best Hyperparameters: {'use_batchnorm': True, 'regularizer': 'l1', 'optimizer': 'RMSprop', 'num_neurons': 64, 'num_layers': 2, 'learning_rate': 0.01, 'dropout_rate': 0, 'activation': 'relu'}

Batch normalization is a technique that normalizes the output of each layer by subtracting the mean and dividing by the standard deviation of the batch. This helps to reduce the effect of internal covariate shift and can make the model more stable during training.

L1 regularizer is a technique used to add a penalty to the model's loss function based on the absolute values of the model's weights. This encourages the model to learn sparse representations and can help to prevent overfitting by reducing the model's complexity.

RMSprop optimizer is an optimization algorithm that uses a moving average of the squared gradient to scale the learning rate of each weight. This can help the model converge faster and more reliably to the optimal solution, particularly in the presence of sparse gradients or noisy data.

ReLU activation function is a non-linear function that replaces negative input values with zero and leaves positive input values unchanged. This can help the model to learn more complex and expressive representations, particularly when used in deep neural networks, Overall, using a combination of these techniques can help to increase the performance of a model by improving its stability, reducing overfitting, and allowing it to learn more complex and expressive representations.

Convolutional Neural Nets

Best Hyperparameters: {'use_batchnorm': True, 'regularizer': 'l2', 'optimizer': 'RMSprop', 'num_layers': 3, 'num_filters': 64, 'learning_rate': 0.001, 'kernal_size': (3, 3), 'dropout_rate': 0, 'activation': 'relu'}

Kernel size determines the receptive field of the CNN and the amount of spatial information that the model can capture from the input image. Using larger kernel sizes can help the model to capture more global features, while smaller kernel sizes can help to capture more local features.

The weight decay method, commonly known as the L2 regularizer, penalises the loss function of the model by adding a penalty depending on the square of the weights. By making the model less complex, this encourages the model to learn lower weights and may help minimise overfitting.

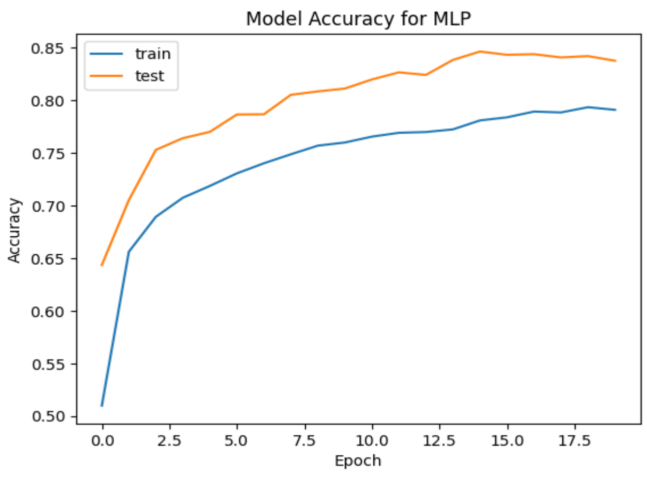
Another significant hyperparameter that may be adjusted to improve the performance of a CNN model is the number of layers. The model may be able to learn more complicated representations and capture more abstract characteristics from the input image by using additional layers. But too many layers can result in overfitting and poor performance.

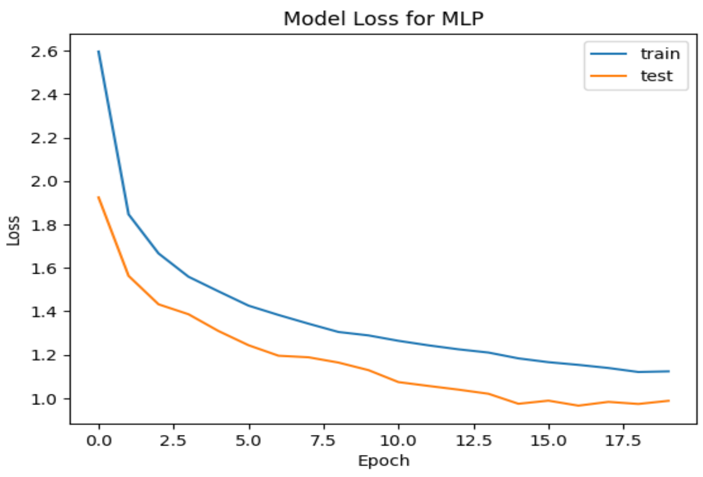
No, we did not face the problem of Overfitting or Underfitting as both the models Training and Testing accuracies were almost similar as we used lots of regularization techniques such as dropouts, Tuning the number of hidden layers and also by using the L1/L2 regularizer

Training Loss is An indicator of a machine learning model's performance during the training stage. Typically, it is determined as the difference between the model's anticipated output and the actual output of the training data**.**

While testing a machine learning model, testing loss is a metric that assesses how well the model does on a collection of untried data. The same formula used to compute the training loss is often used to calculate it for the testing data. The testing loss is a helpful indicator for assessing how effectively the model generalises to fresh data and for spotting any overfitting problems.

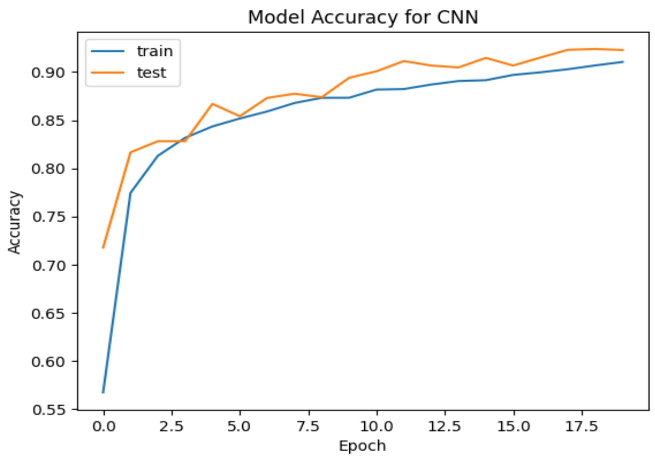
Loss and Accuracies of MLP model

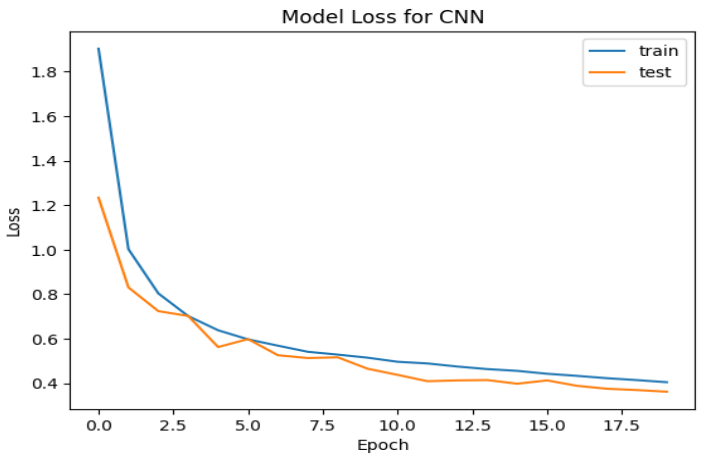




The Testing Accuracy was found to be 0.83, As there is very little difference between the Training and Testing Accuracy the performance of the model seems to be good. 0.86, 0.83,0.83 are the Precision, Recall and F1 score of the Model.

Loss and Accuracies of CNN model

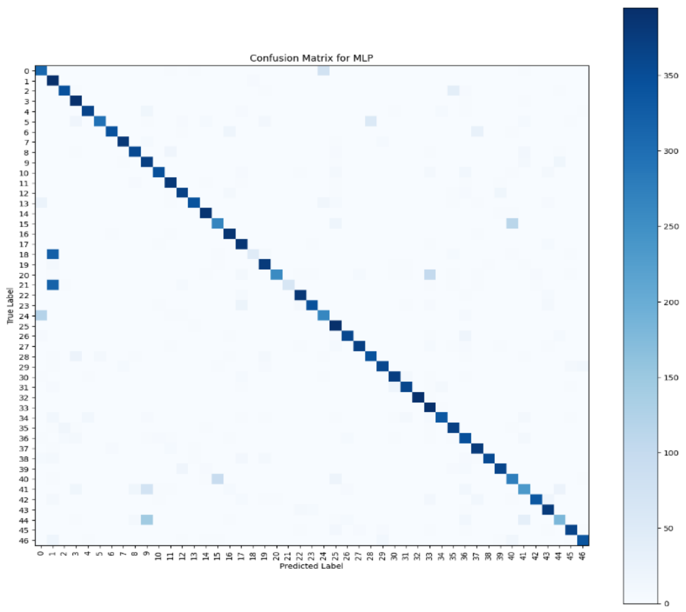


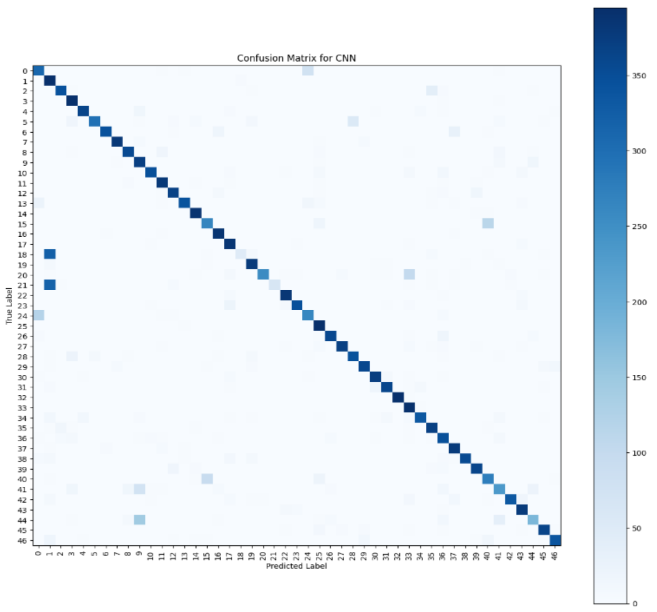


The Testing Accuracy was found to be 0.92, As there is very little difference between the Training and Testing Accuracy the performance of the model seems to be good. 0.929, 0.92,0.92 are the Precision, Recall and F1 score of the Model.

Confusion Matrix

The diagonal elements in the confusion matrix of both the models seems to be very high, it indicates that the model is making confident and accurate predictions for each class.





Results

From the Above results we can see that CNN has outperformed MLP, as the testing accuracy of CNN is much higher than the testing Accuracy of MLP.

This is Because the spatial associations between pixels in an image are crucial for image classification tasks, and CNNs can manage this spatial information. By utilising convolutional layers, which scan the input image and discover key features for the classification task, CNNs are built to handle this spatial information.

CNNs are also built to learn hierarchical characteristics from the input image, they can recognise intricate patterns and relationships between pixels.

Because MLPs require a fixed-size input, they are problematic for applications requiring the categorization of images, because the size of the images can vary. MLPs are sensitive to the input size. In contrast, CNNs use pooling and convolutional layers to handle input images of various sizes.

Overall, because of their capacity for handling spatial information and handling input images of various sizes, CNNs are more suitable for image classification tasks than MLPs.

Conclusion

Through this research, we have learned how deep neural networks function and how to easily and accurately classify multiple classes using a variety of strategies. We now have a better understanding of more sophisticated deep learning techniques for computer vision-based picture classification. When employing the backpropagation technique to calculate and update all of the weights, we encountered a lack of quick computational capacity. Perhaps we can improve on tuning additional hypermeters in both MLP and CNN, which will lengthen the computation time but result in a model that is better optimised.