Digit Recognition

1 Introduction

In this assignment, we have implemented a Multilayer Perceptron Neural Network and evaluated its performance in classifying handwritten digits. After completing this assignment we are able to understand how neural network works, How to setup a Machine Learning experiment on public data, How regularization, dropout plays a role in machine learning implementation and How to fine-tune a well-train model.

2 Dataset

To implement the multi-layer perceptron neural network, we used a public dataset i.e., MNIST dataset which consists of training set of 60000 examples which were divided into 50000 examples for training set and validation set of 10000 dataset. All the digits have been normalised and sent with a fixed image of 28x28 size. In the MNIST dataset each pixel in the image is represented by an integer between 0 and 255, where 0 is black, 255 is white and anything between represents different shade of Gray.

3 Implementation

Step-1: Loading the dataset and displaying the number of training and validating datasets.

```
Total training images 50000 Total testing images 10000
```

Fig-1: Showing number of datasets in training and testing.

Step-2: Number of images in each of the classes within the training dataset.

```
Number of images per class:
Class 5: 4506
Class 0: 4932
Class 4: 4859
Class 1: 5678
Class 9: 4988
Class 2: 4968
Class 3: 5101
Class 6: 4951
Class 7: 5175
Class 8: 4842
```

Fig-2: Showing number of images per class.

Step-3:

3.1 Created a customised CNN model with 8 layers.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2  (None, 13, 13, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 5, 5, 64)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 3, 3, 128)         73856

 flatten (Flatten)           (None, 1152)              0

 dense (Dense)               (None, 256)               295168

 dense_1 (Dense)             (None, 10)                2570

=================================================================
Total params: 390410 (1.49 MB)
Trainable params: 390410 (1.49 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Fig-3: Baseline CNN model summary

3.2 Trained the customised CNN model using 2 optimisers i.e., Adam, RMSprop.

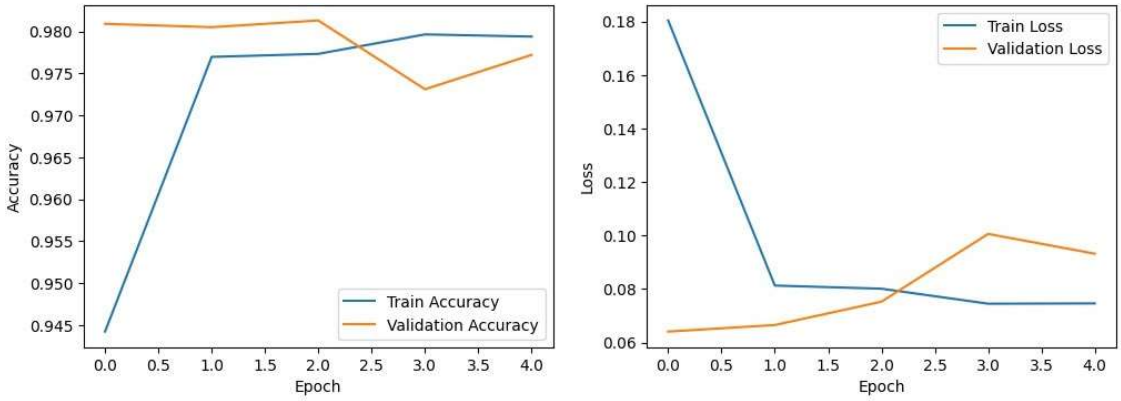

Fig-4: Adam Optimiser, Left: Train accuracy and Validation Accuracy. Right: Train loss and validation loss.

We have achieved an accuracy of 97% by using Adam optimiser to the CNN model.

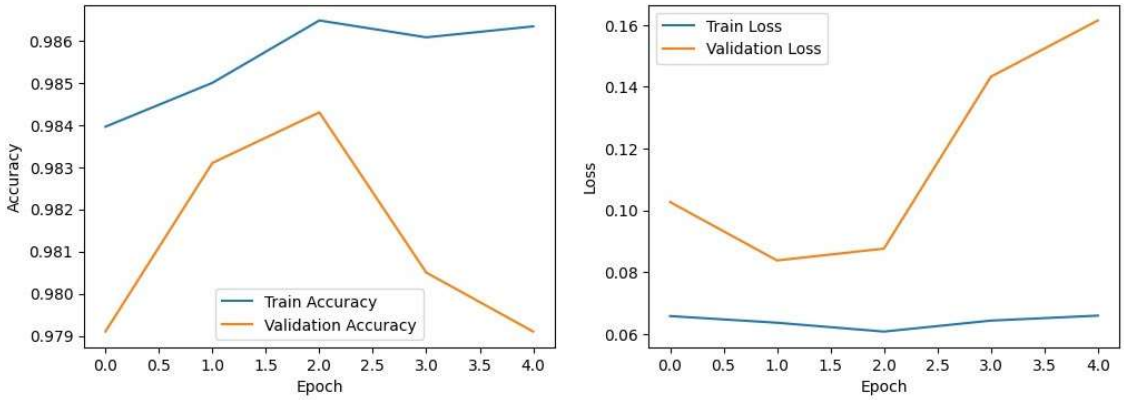

Fig-5: RMSprop Optimiser, Left: Train accuracy and Validation Accuracy. Right: Train loss and validation loss.

We have achieved an accuracy of 97.9% by using RMSprop optimiser to the CNN model.

Optimisers	Train accuracy	Train loss	Val accuracy	Val loss	Test accuracy	Test loss
Adam	97.7%	0.81	96.8%	0.122	96.68%	0.12
RMSprop	98%	0.075	94%	0.275	93.9%	0.28

Table-1: Showing the comparison between Adam and RMSprop.

3.3 Evaluation of all CNN models on different regularization methods.

The table below is a comparative analysis of 3 regularization methods i.e., l1, l2, dropout and baseline.

Regularization methods	Train Epochs	Train accuracy	Train loss	Val accuracy	Val loss	Test accuracy	Test loss
L1	5	97%	0.79	96.1%	0.78	96.9%	0.75
L2	5	94.3%	0.37	94.7%	0.33	95%	0.32
Dropout	5	96.3%	0.13	96.9%	0.129	97.54%	0.14
Baseline	5	97.8%	0.085	96.3%	0.121	96.6%	0.127

Table-2: Showing the comparison between various regularization methods.

L1 regularization:

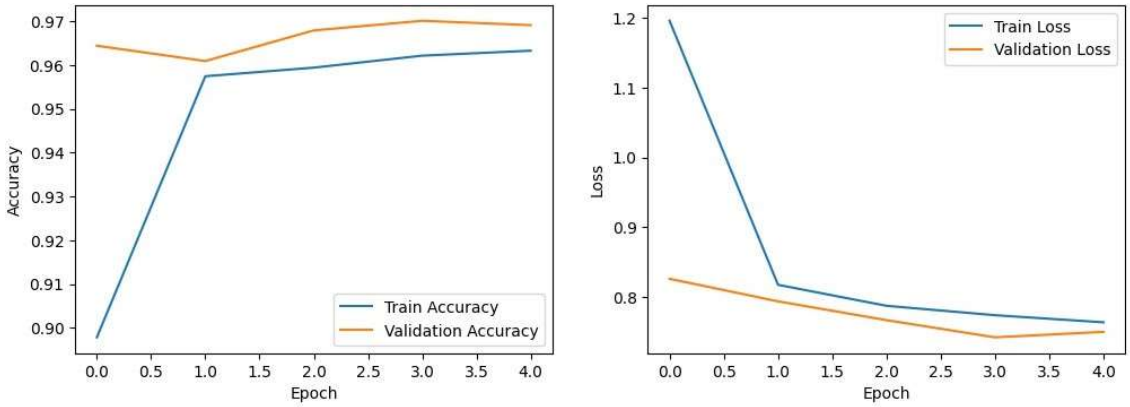

Fig-6: Train accuracy vs Validation accuracy and Train loss vs Validation Loss for L1 regularization

L2 regularization:

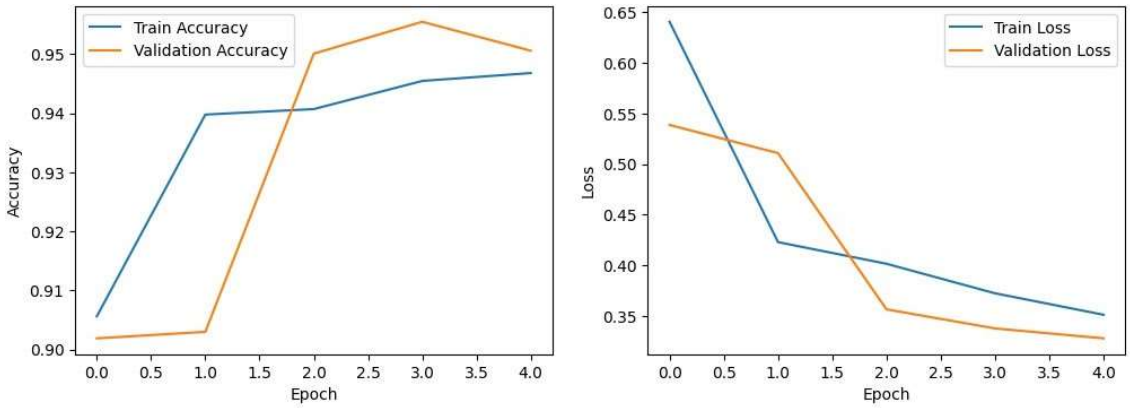

Fig-7: Train accuracy vs Validation accuracy and Train loss vs Validation Loss for L2 regularization

Dropout Regularization:

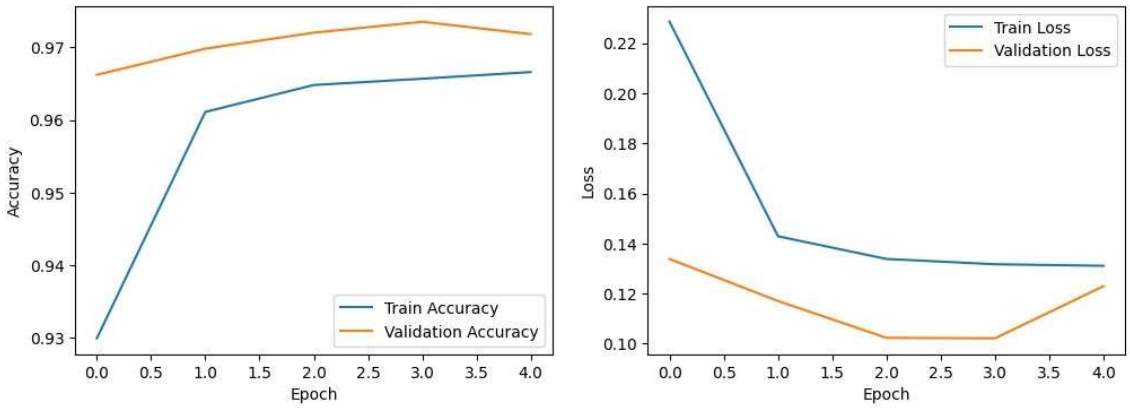

Fig-8: Train accuracy vs Validation accuracy and Train loss vs Validation Loss for dropout regularization.

Baseline:

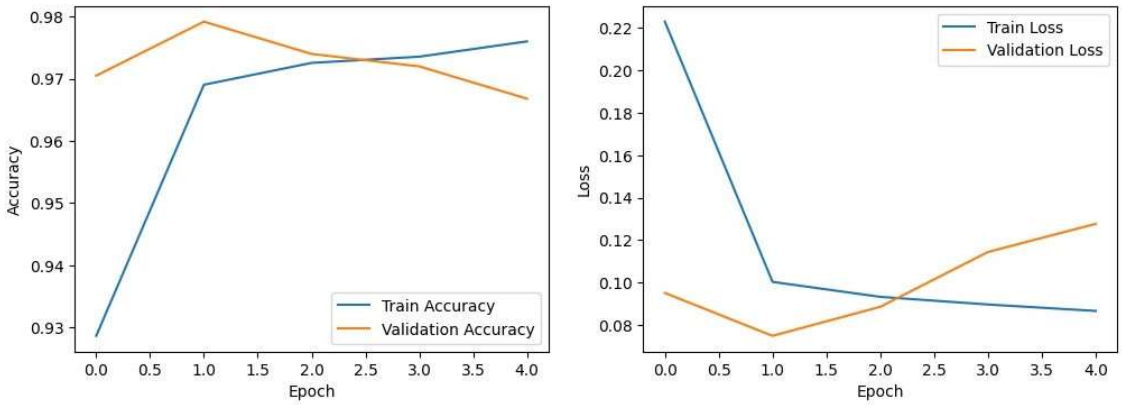

Fig-9: Train accuracy vs Validation accuracy and Train loss vs Validation Loss for baseline.

Step-4: Implementation of various pre-trained CNN models

1. LeNet5: LeNet-5, a well-known option for MNIST, is renowned for its efficiency in digit identification tasks. In comparison to more contemporary models, it has a relatively shallow design, making it a useful place to start for straightforward applications like MNIST.

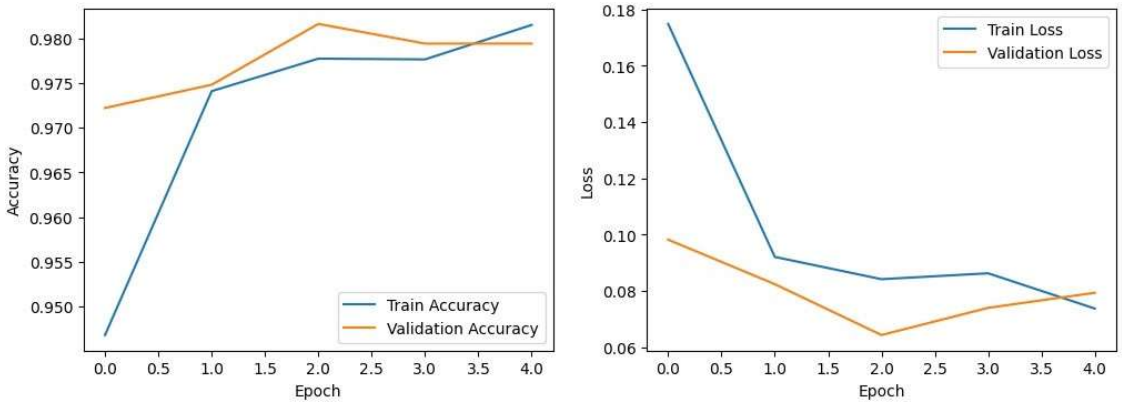

Fig-10: Train accuracy vs Validation accuracy and Train loss vs Validation Loss Using Adam optimizer for LeNet5 model.

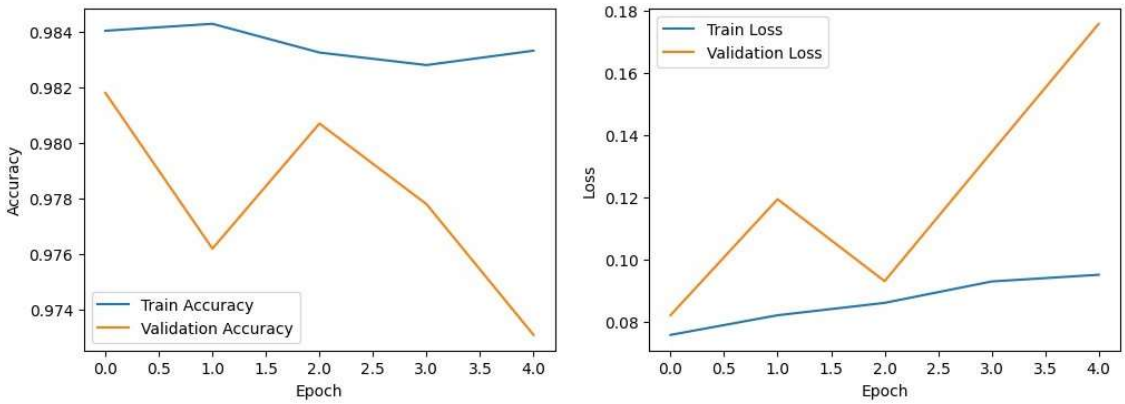

Fig-11: Train accuracy vs Validation accuracy and Train loss vs Validation Loss Using RMSprop optimizer for LeNet5 model.

2. ResNet-like: The ResNet design, which added residual connections to prevent vanishing gradients, is a scaled-down version of the ResNet architecture. ResNet-18 is appropriate for MNIST because it strikes a fair balance between model complexity and accuracy.

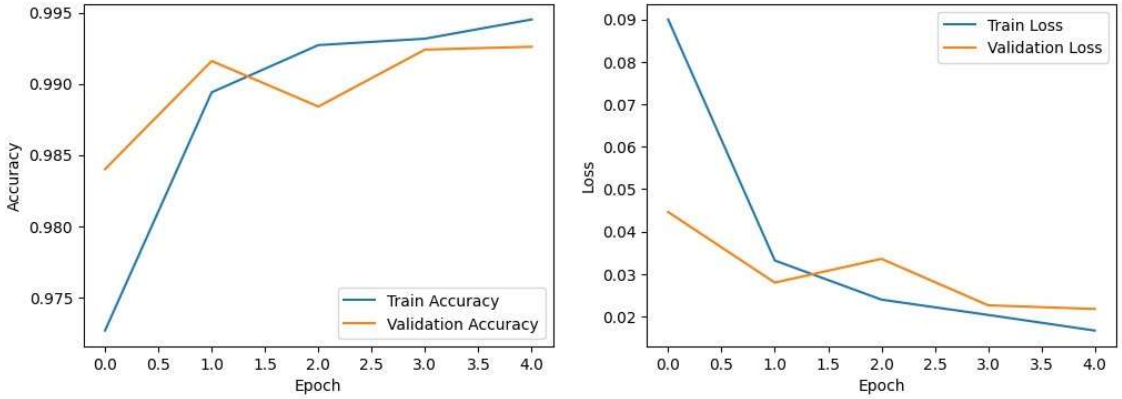

Fig-12: Train accuracy vs Validation accuracy and Train loss vs Validation Loss Using Adam optimizer for ResNet-like model.

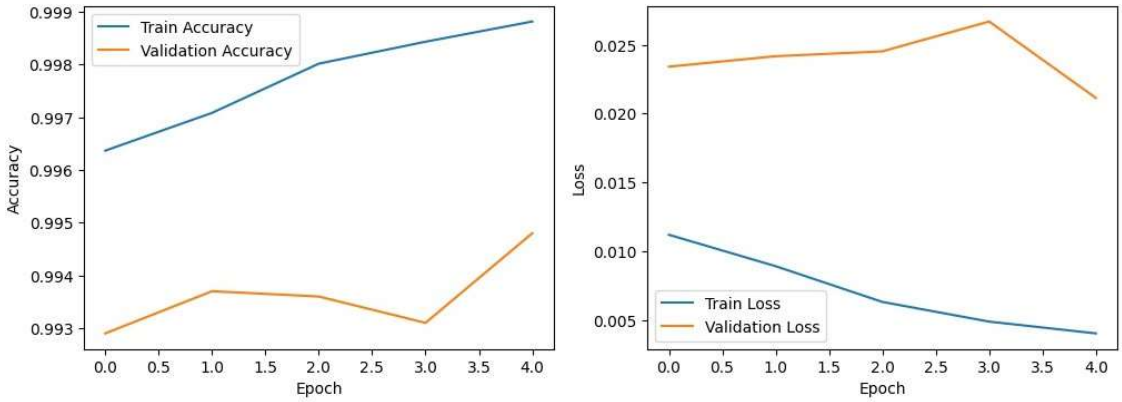

Fig-13: Train accuracy vs Validation accuracy and Train loss vs Validation Loss Using RMSprop optimizer for ResNet-like model.

3. VGG16: The VGG models, especially VGG-16, have demonstrated outstanding performance in a variety of image classification applications. Despite being more complex than LeNet-5, they are still reasonably easy to implement and, with the right training, can offer high accuracy on MNIST.

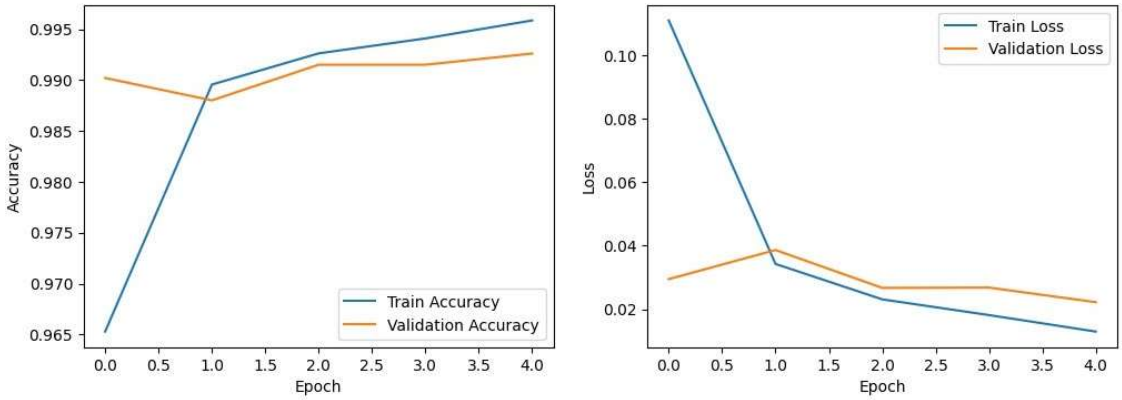

Fig-14: Train accuracy vs Validation accuracy and Train loss vs Validation Loss Using Adam optimizer for VGG16 model.

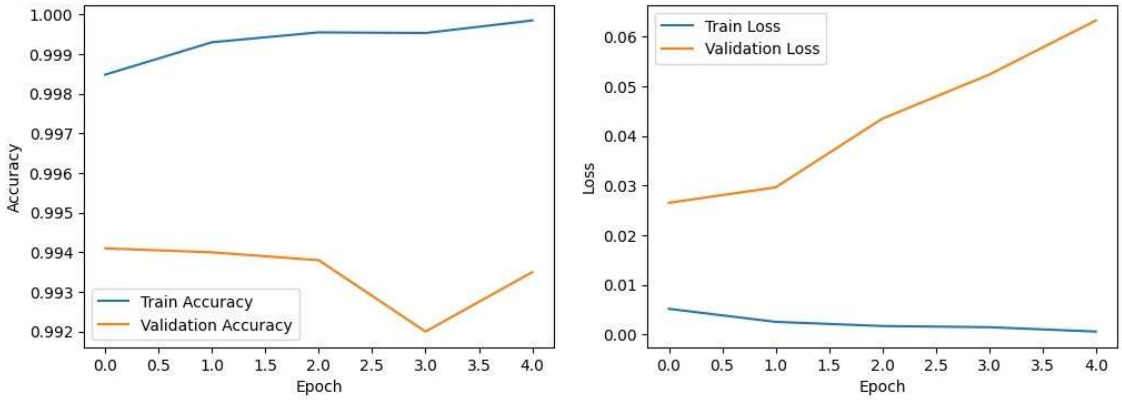

Fig-15: Train accuracy vs Validation accuracy and Train loss vs Validation Loss Using RMSprop optimizer for VGG16 model.

Comparison between different models using Adam optimizer.

Models	Train accuracy	Train loss	Val accuracy	Val loss	Test accuracy	Test loss
LeNet5	98.2%	0.07	97.7%	0.08	97.9%	0.07
ResNet-like	99.4%	0.01	99.2%	0.02	99.2%	0.021
VGG16	99.5%	0.01	99.3%	0.023	99.26%	0.022

Table-3

Comparison between different models using RMSprop optimizer.

Models	Train accuracy	Train loss	Val accuracy	Val loss	Test accuracy	Test loss
LeNet5	98.3%	0.09	97.3%	0.18	97.3%	0.175
ResNet-like	99.9%	0.004	99.5%	0.021	99.48%	0.02
VGG16	100%	0	99.3%	0.06	99.3%	0.063

Table-4

4 Conclusion:

Coming to the initial CNN models, the models having Adam optimizer proved better results compared to RMSprop optimizer. And, by comparing L1, L2, Dropout regularizations and Baseline, Dropout regularization using Adam optimizer showed better accuracy than other methods.

In the case of the pre-trained models i.e., VGG16, LeNet5, RestNet-like VGG16 showed good accuracy using Adam optimizer whereas ResNet-like showed the good accuracy for RMSprop optimizer.