CS6910: Fundamentals of Deep Learning

Assignment # 1

Team-8

Aditya Raj - ED21B006
Annangi Shashank Babu - EE21B021
Ata Karim Subhani - EE21B025
Juswanth T - EE21B063

March 25, 2024

Contents

List of Figures

List of Tables

1 Task 1

1.1 Aim

Compare various optimization methods for training a Multilayer Feedforward Neural Network (MLFFNN) on Image Dataset 1 with 5 classes for classification tasks. The primary objective is to evaluate and compare the performance of different weight update rules in terms of convergence speed and classification accuracy.

1.2 Model Configuration

- **Model Type:** MLFFNN with 2 hidden layers.
- **Activation Function:** Hyperbolic Tangent (Tanh) in hidden layers and Softmax in output layer.
- **Loss function:** Cross-Entropy.
- **Learning Mode:** Pattern mode.
- **Stopping Criterion:** Change in average error below a threshold. Threshold used was **0.002**.
- **Learning Rate Parameter:** $\eta = \mathbf{0.01}$ was used as learning rate parameter across all optimization techniques.
- **Initial Random Weights:** Xavier-Glorot initialization.

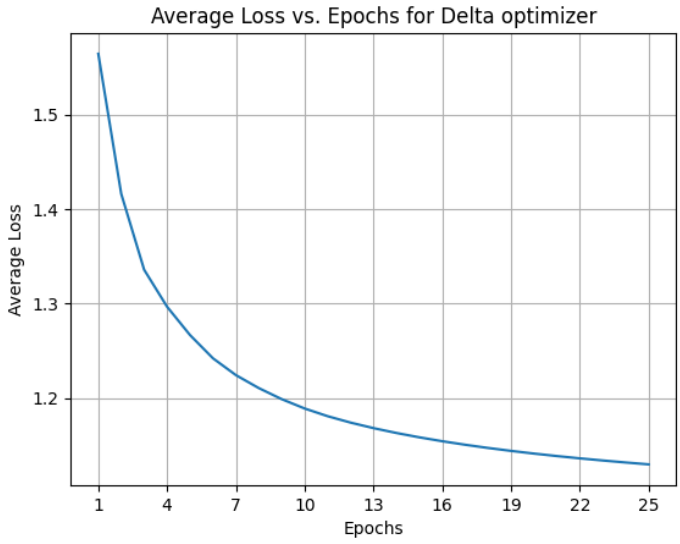
```
for layer in model.children():
  print(layer)

Linear(in_features=36, out_features=20, bias=True)
Linear(in_features=20, out_features=10, bias=True)
Linear(in_features=10, out_features=5, bias=True)
```

Figure 1: Model

1.3 Result and Analysis

- **<u>Delta Rule</u>**



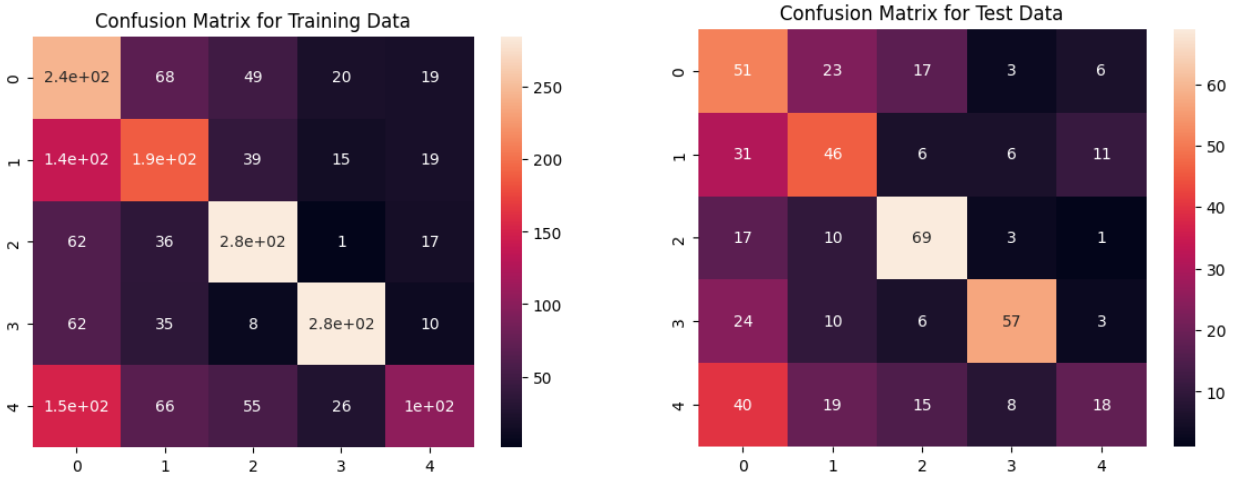
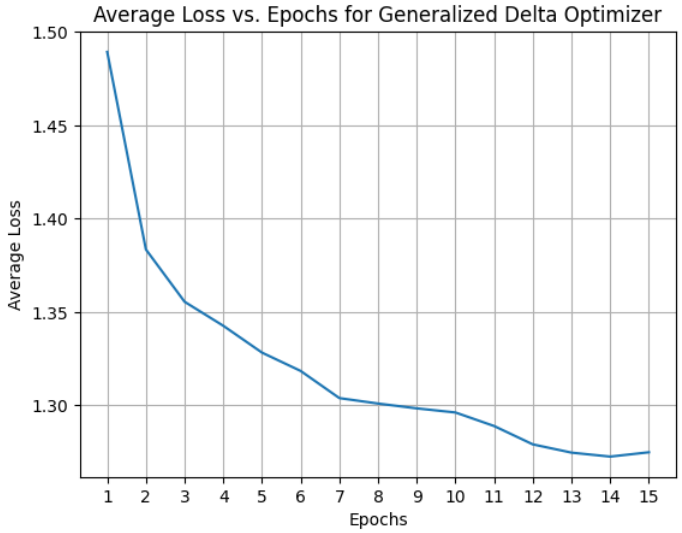


Figure 2: Confusion Matrix for Delta rule,

- **Generalized Delta Rule**



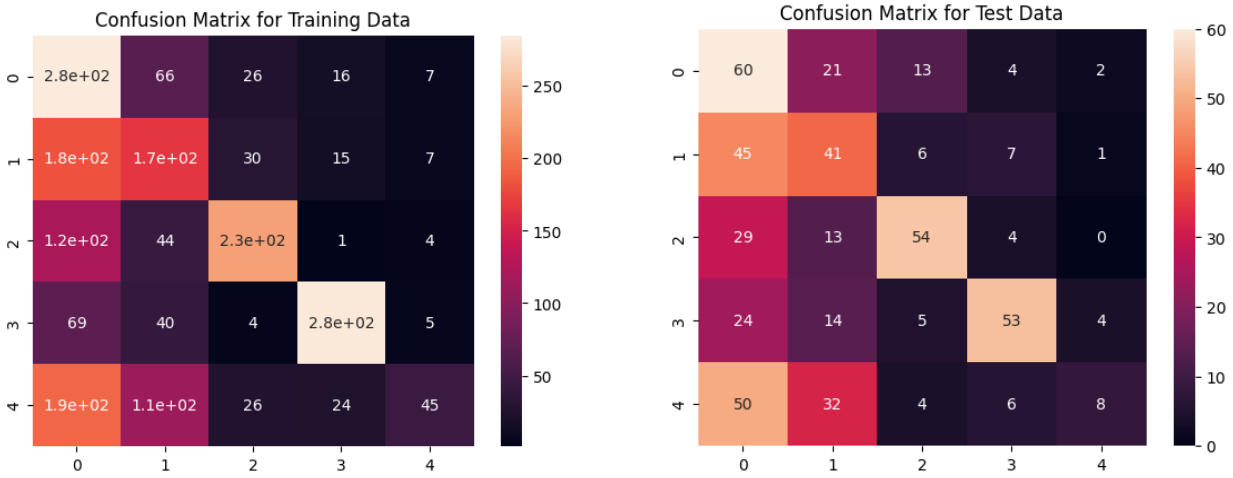


Figure 3: Confusion Matrix for Generalized Delta rule,

- **AdaGrad**

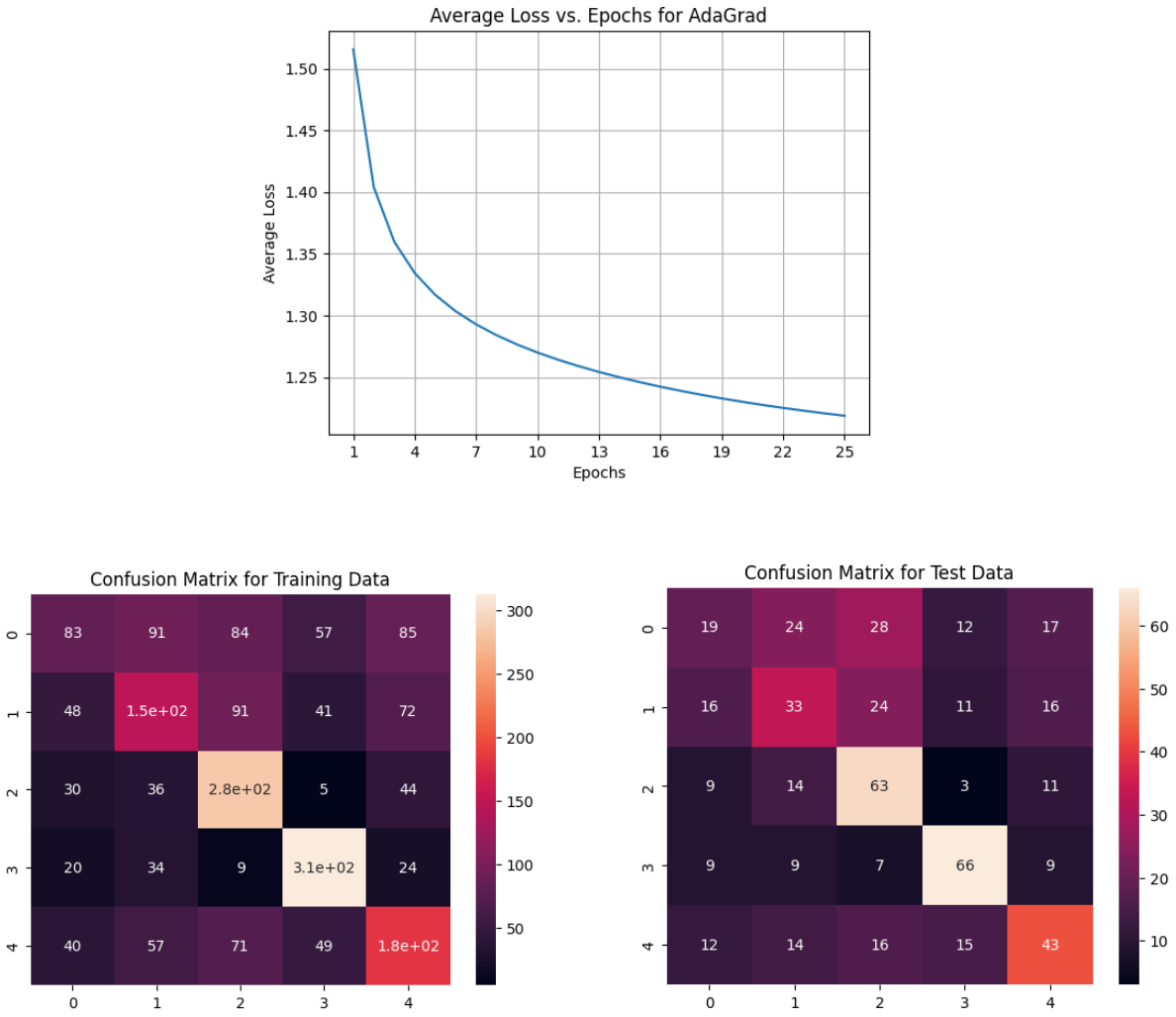
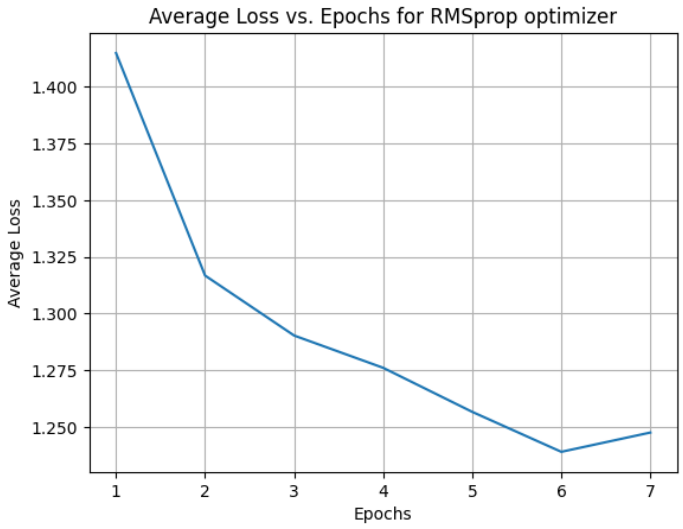


Figure 4: Confusion Matrix for AdaGrad

- **RMSProp**



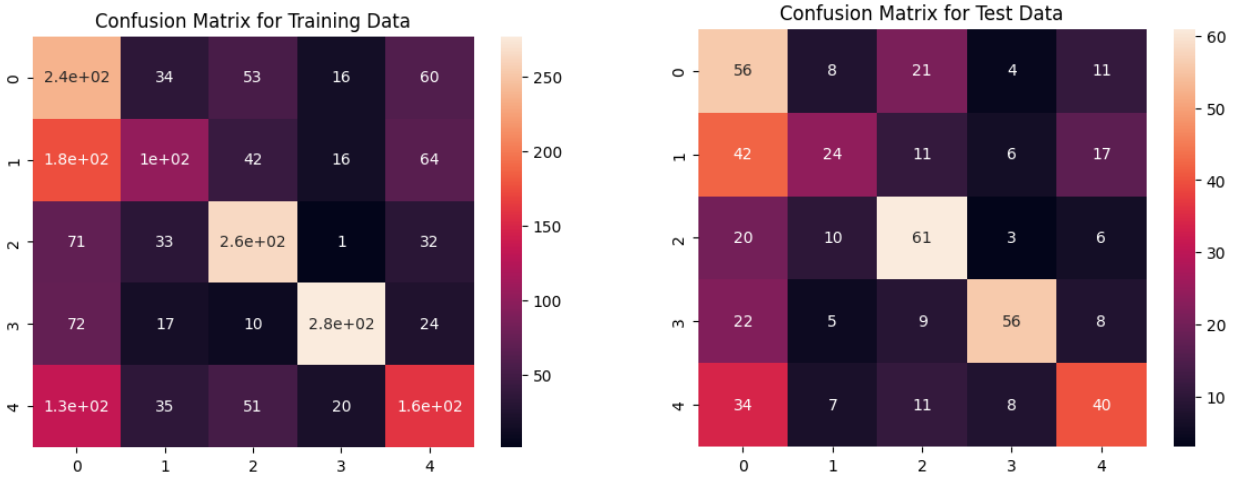
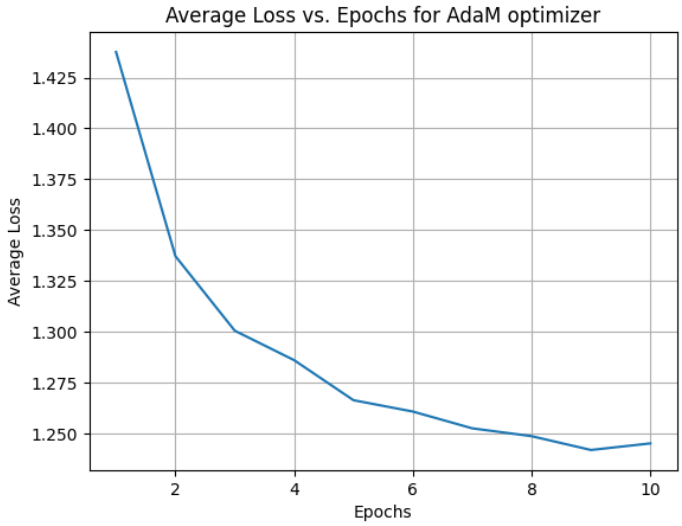


Figure 5: Confusion Matrix for RMSProp

- **AdaM**



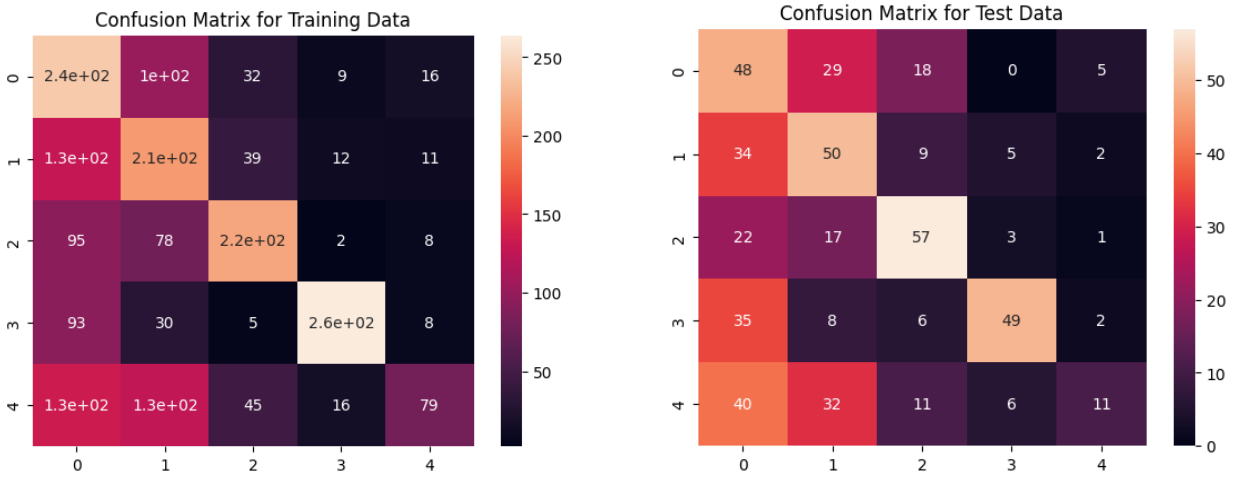


Figure 6: Confusion Matrix for AdaM

1.4 Results

Table 1: Finding of task 1

Optimizer	Number of Epoch to converge	Accuracy on test data
Delta	25	48.20%
Generalized Delta	15	43.20%
AdaGrad	25	44.80%
RMSProp	7	47.40%
AdaM	10	43.00%

1.5 Hyperparameter Tuning

- We tuned the value of threshold for convergence. With very low value of threshold we were getting better training accuracy but validation curve was crossing the optimal point. This behaviour was expected because of overfitting. To fix that we increase threshold value.
- We also played with changing number of neurons of in the model but we were not getting satisfactory result. With higher number of neurons model get overfitted and validation error loss very high.

1.6 Conclusion

- AdaM might be a good initial choice due to its high accuracy, but it might require more iterations.
- RMSProp seems to converge faster. Data may have many features with near-zero gradients most of the time (sparse gradients), RMSProp can be advantageous.
- Delta method is giving higher accuracy because it is taking more number of epochs.
- In conclusion, while RMSProp has some advantages in specific scenarios, Adam is often the preferred choice due to its overall effectiveness and robustness. However, experimentation is crucial to determine the optimal optimizer for your particular machine learning task.

2 Task 2

2.1 Aim

To investigate the impact of batch normalization (post-activation) and no normalization on the performance of a Multi-Layer Feedforward Neural Network (MLFFNN).

2.2 Model Configuration

- **Model Type:** MLFFNN with 2 hidden layers.
- **Activation Function:** Hyperbolic Tangent (Tanh) in hidden layers.
- **Loss function:** Cross-Entropy.
- **Learning Mode:** Mini-batch mode.
- **Stopping Criterion:** Change in average error below a threshold.
- **Weight update rule:** AdaM

2.3 No Normalization

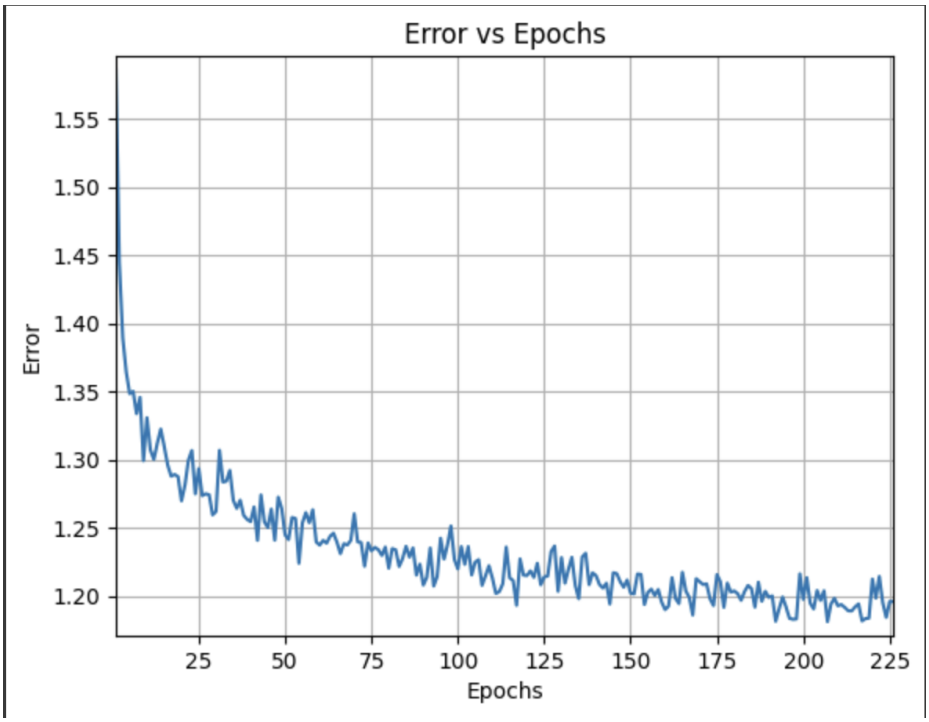


Figure 7: Average Loss Curve

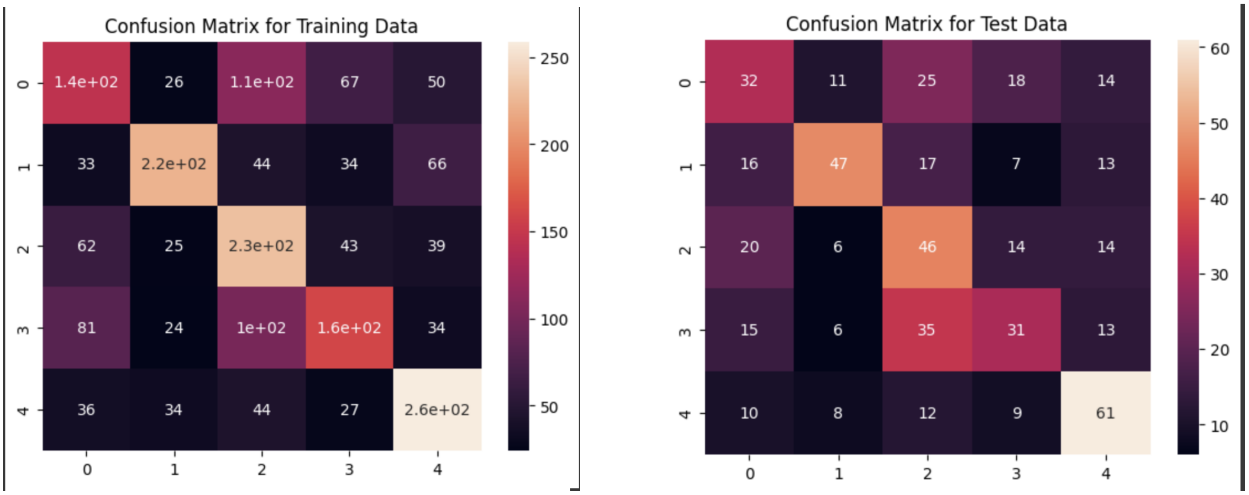


Figure 8: Confusion Matrix for **No normalization**

2.4 Batch Normalization

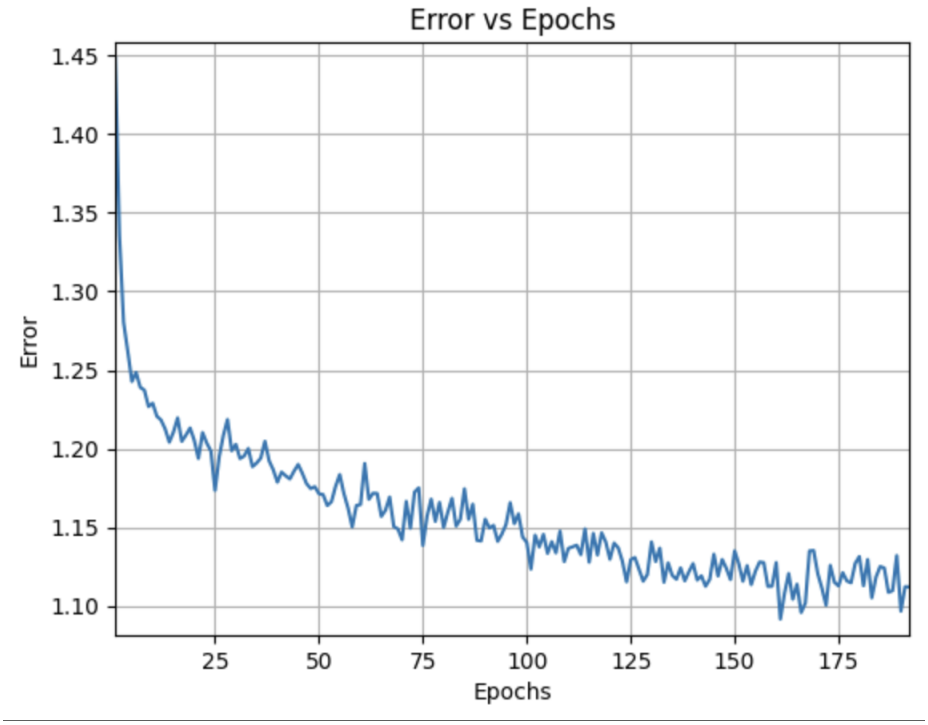


Figure 9: Average Loss Curve

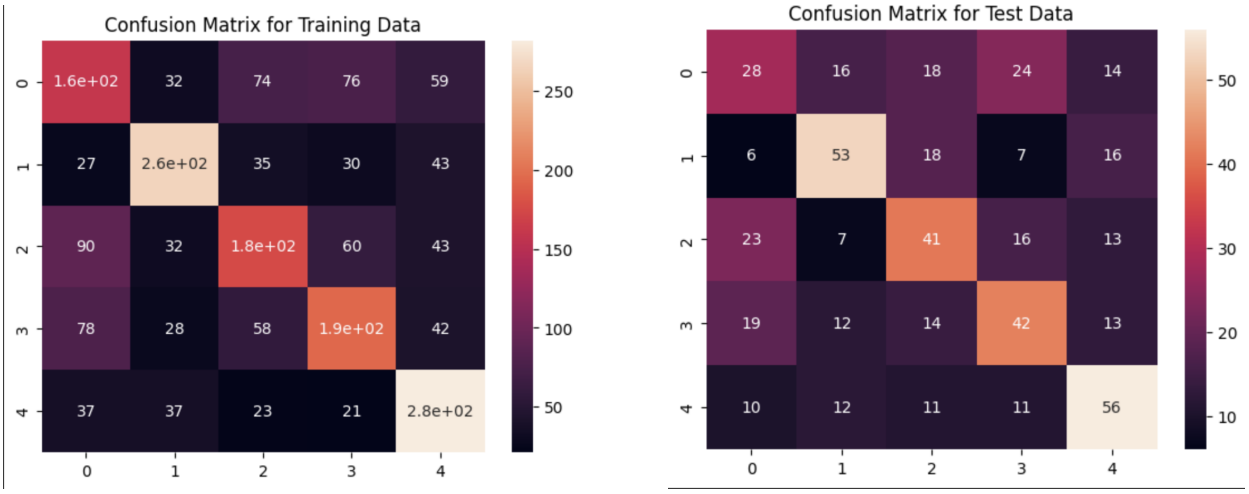


Figure 10: Confusion Matrix for **Batch normalization**

2.5 Results

- **Batch Size :** 100
- **Dropout Probability** : 0.55
- **Hidden Layer 1 :** 25
- **Hidden Layer 2 :** 300
- **Learning Rate :** 0.01
- **L2 Regularization** (λ) : 0

Normalization	Number of Epoch to converge	Train Accuracy	Test Accuracy
No Normalization	225	52.6%	46.5%
Batch Normalization	190	54.5%	48.7%

2.6 Hyperparameter Tuning

Batch Size	Dropout Probability	Hidden Layer size 1	Hidden Layer size 2	Learning Rate	L2 (λ) Regularization	Test Accuracy
447	0.5	100	76	0.0001	0.01	48.8
10	0.745	28	76	0.1	0.01	48.2
447	0.35	52	52	0.01	0.01	47.4
447	0.5	27	272	0.01	0.001	48.0
30	0.8	22	515	0.001	0.01	50.0
447	0.65	22	757	0.001	0.01	48.0
186	0.65	22	1000	0.01	0.01	48.0
30	0.8	27	757	0.01	0.01	48.0

All these tuning is done for batch normalization. We used regularization techniques such as L2 regularization and Drop out regularization. For few hyper parameter values we even got a better accuracy, but still we didn't consider them as the best since the same parameters gave poor accuracy in no normalization case. So there is a trade off, and hence

we used the parameters which gave good accuracy in both the ways, i.e., with and without batch normalization.

We also used Data augmentation regularization technique by adding little noise to the image but unfortunately we did not get better results with it . With the same parameters as above but with data augmentation we got accuracy (on test data) of roughly 4% - 6% less than what we got without it.

2.7 Conclusion

From these results, it can be concluded that the inclusion of batch normalization led to an improvement in both the convergence time and the model's performance. Batch normalization helps stabilize and accelerate the training process by normalizing the input to each layer, which can result in better overall performance. However, it's worth noting that even with batch normalization, the achieved accuracies might still not be satisfactory.

3 Task 3

3.1 Aim

The objective is to compare the performance of the DFNN trained solely on labeled data with that of a DFNN pre-trained using unlabeled data with stacked autoencoders and fine-tuned with labeled data

3.2 Model Configuration

```
for layer in model.children():
  print(layer)
```

```
Linear(in_features=36, out_features=30, bias=True)
Linear(in_features=30, out_features=24, bias=True)
Linear(in_features=24, out_features=20, bias=True)
Linear(in_features=20, out_features=16, bias=True)
Linear(in_features=16, out_features=12, bias=True)
Linear(in_features=12, out_features=8, bias=True)
Linear(in_features=8, out_features=5, bias=True)
```

(a) DFNN trained using labelled data

```
for layer in model.children():
  print(layer)
```

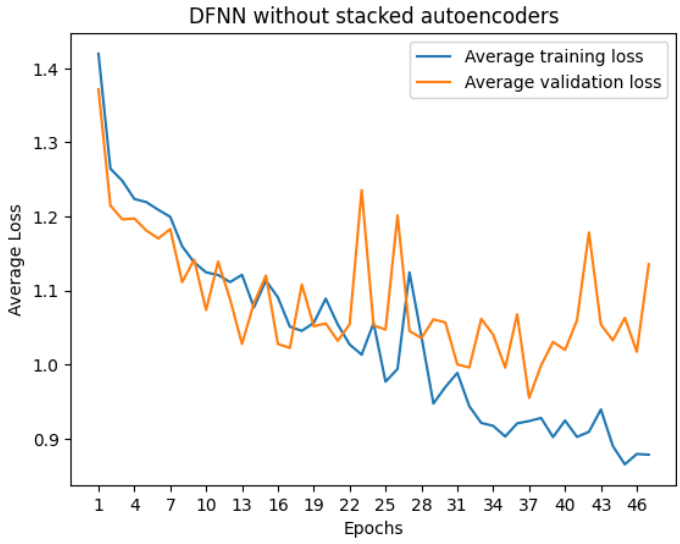
```
AANN(
  (hl1): Linear(in_features=36, out_features=30, bias=True)
  (hl2): Linear(in_features=30, out_features=24, bias=True)
  (hl3): Linear(in_features=24, out_features=30, bias=True)
  (ol): Linear(in_features=30, out_features=36, bias=True)
)
AANN(
  (hl1): Linear(in_features=24, out_features=20, bias=True)
  (hl2): Linear(in_features=20, out_features=16, bias=True)
  (hl3): Linear(in_features=16, out_features=20, bias=True)
  (ol): Linear(in_features=20, out_features=24, bias=True)
)
AANN(
  (hl1): Linear(in_features=16, out_features=12, bias=True)
  (hl2): Linear(in_features=12, out_features=8, bias=True)
  (hl3): Linear(in_features=8, out_features=12, bias=True)
  (ol): Linear(in_features=12, out_features=16, bias=True)
)
Linear(in_features=8, out_features=5, bias=True)
```

(b) DFNN using stacked autoencoder

Figure 11: Model architecture

- **Mode of Learning:** Mini batch mode of learning
- **Stopping Criteria:** Training stops when the change in average error falls below a predefined threshold.
- **Weight update rule:** The AdaM (Adaptive Moment Estimation) algorithm is used for weight updates during training.
- **Learning rate parameter:** $\eta = 0.01$ was used across all training process.

3.3 Training DFNN with labelled data



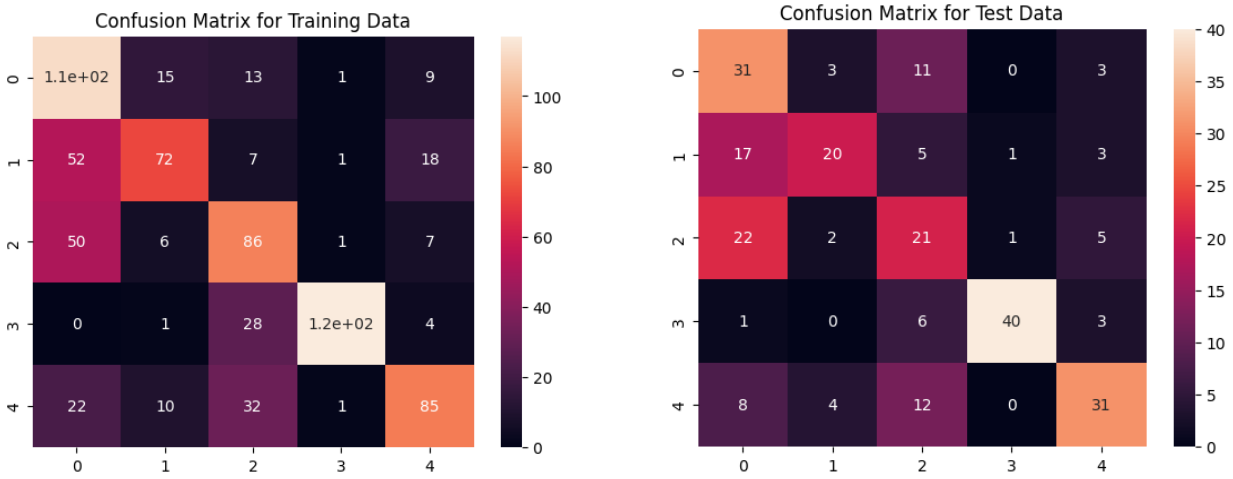


Figure 12: Confusion Matrix for DFNN trained using only labeled data,

3.4 DFNN using stacked autoencoder using unlabeled data

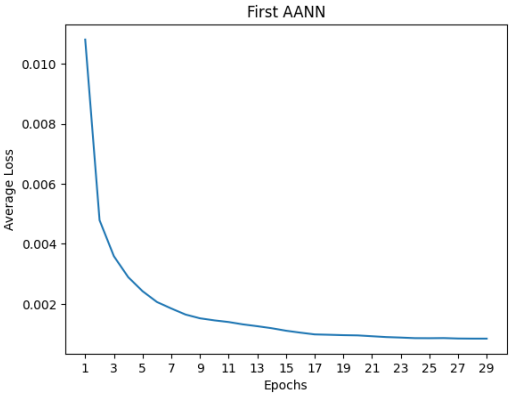


Figure 13: Loss curve for 1st autoencoder

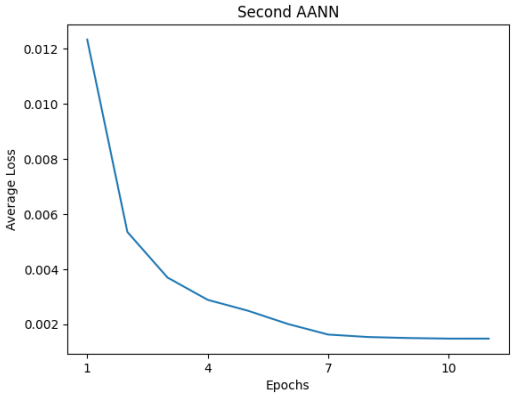


Figure 14: Loss curve for 2nd autoencoder

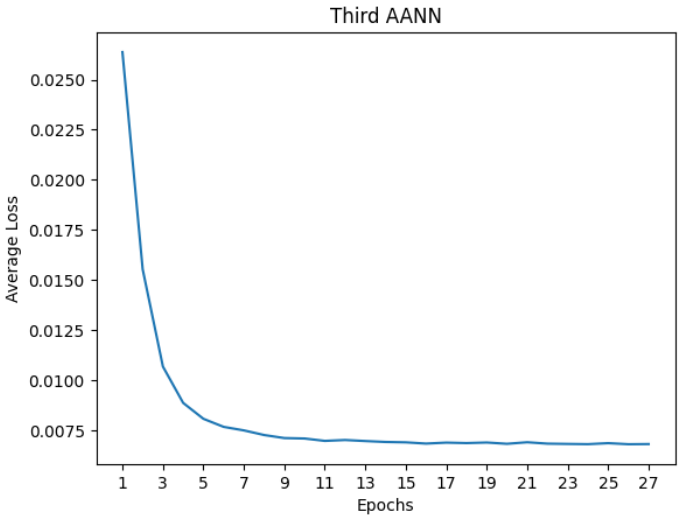


Figure 15: Loss curve for 3rd autoencoder

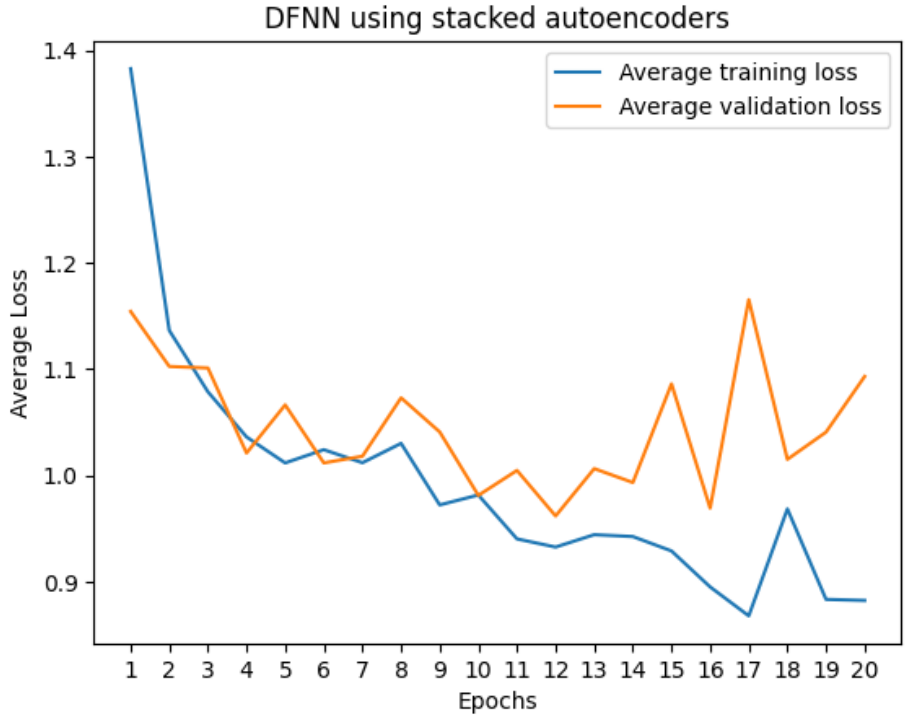


Figure 16: Average Loss Curve for stacked autoencoder

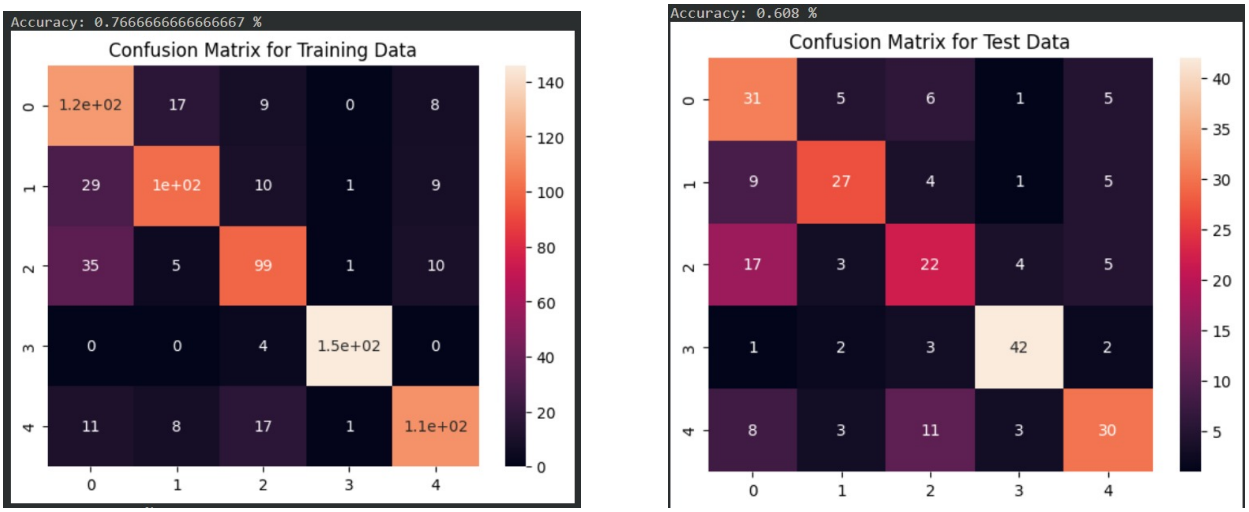


Figure 17: Confusion Matrix for DFNN trained using stacked autoencoder

3.5 Results

Table 2: Finding of Task 3

Training Method	Number of Epoch to converge	Train Accuracy	Test Accuracy
DFNN using labeled data	47	62.93%	57.20%
Stacked autoencoder	20	76.67%	60.80%

3.6 Hyperparameter Tuning

We used some sensible hit and trial method for picking value of threshold, learning rate parameter and model configuration (number of neurons in each layer).

- Initially we pick large number of neurons in hidden layer but loss was too high on validation data set. Because of large number of parameters model get overfitted. To tune that that we decreased number of neurons in hidden layers.
- We tuned learning rate after looking at the loss curve. For large value of learning rate like 0.05 or 0.1 loss curve was not smooth. For very low values like 0.001 it was converging very late. we fixed $\eta = 0.01$ as final learning rate parameter.
- We fix very low threshold value for training each of the autoencoder for stacked autoencoder. For training DFNN (both versions) we used little larger value of threshold.

  We mostly played with learning rate parameter and threshold values to tune our model.

3.7 Conclusion and Observations

....

The performance of the DFNN pre-trained with stacked autoencoders and fine-tuned with labeled data is expected to show improvements compared to the DFNN trained solely on labeled data. This technique leverages the unlabeled data to initialize the weights of the DFNN, enabling it to learn more efficiently from the labeled data. The confusion matrices provide insights into the classification performance of both models, aiding in the evaluation and comparison of their effectiveness.

We can also infer from their loss curve that the stacked encoder is showing less average loss with respect to DFNN trained on labeled data. It is happening because stacked encoder is pretrained while in the first case we are training the complete model. So it is taking more number of epochs to reach the same loss.

Since the stacked autoencoder was learnt during pre training phase it gets overfitted in lesser number of epoch than in DFNN we trained in first part. DFNN trained with labeled data is getting overfitted at around 28 epoch while stacked autoencoder is getting overfitted at around 12 epochs.