CS 599: Deep Learning - Lab 3

Neha Nagaraja
School of Informatics, Computing, and Cyber Systems
Northern Arizona University
`nn454@nau.edu`

Link to the Github: https://github.com/Neha-Nagaraja/CS599-DL/tree/master/Assignment4

1 Problem 1: Implementing various Normalization using basic ten- sorflow op

1.1 Objective

The objective of this assignment is to implement and compare Batch Normalization (BN), Weight Normalization (WN), and Layer Normalization (LN) using TensorFlow 2. The goal is to analyze their impact on the performance of a CNN trained on the Fashion-MNIST dataset.

1.2 Implementation

Implementation: We designed a Convolutional Neural Network (CNN) in TensorFlow 2 to evaluate the effects of different normalization techniques on model performance using the Fashion-MNIST dataset.

The architecture consists of a convolution layer (5×5 filters), followed by an optional normalization layer (BatchNorm, LayerNorm, or WeightNorm), ReLU activation, max pooling, a fully connected hidden layer, and a final output layer.

We implemented three normalization methods from scratch:

Batch Normalization (BN): Normalizes each feature across the batch, followed by learnable scale (γ) and shift (β) parameters.

Layer Normalization (LN): Normalizes across features within each sample, followed by learnable γ and β parameters.

Weight Normalization (WN): Reparameterizes weights using a norm-scaled vector:

$$w = \left(\frac{g}{\|v\|}\right) \times v$$

where g is a learnable scalar parameter and v is the weight vector.

Six training modes were tested: No Normalization, Custom BN, TensorFlow BN, Custom WN, Custom LN, TensorFlow LN

Training was done using momentum-based SGD (learning rate = 0.01, momentum = 0.9) for 4 epochs with a batch size of 100. Accuracy and loss were recorded per epoch for both training and test sets. Custom implementations were validated against TensorFlow's built-in layers to ensure correctness. Results were visualized using line and bar plots for comparison.

1.3 Batch Normalization

Batch Normalization (BN) helps in stabilizing training and improving convergence speed by normalizing activations across mini-batches.

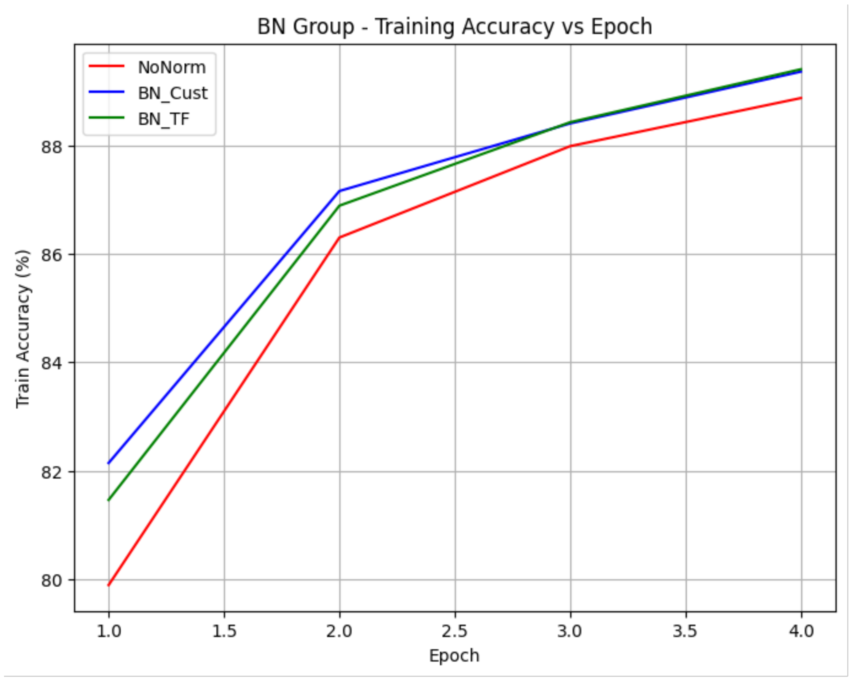

Figure 1: Batch Norm - Training Accuracy vs Epoch

In our experiments, the custom Batch Normalization showed better performance compared to the model without normalization. However, TensorFlow's built-in BN achieved test accuracy slightly lower than the baseline without normalization. This small variation can be due to factors like floating-point precision, randomness, and TensorFlow's internal handling of running statistics.

Table 1: Final Test Accuracy Comparison

Mode	**Final Test Accuracy (Batch Normalization)**
No Normalization	88.25%
Custom BN	88.55%
TensorFlow BN	88.14%

Observations:

- BN improved training accuracy and helped the model converge faster, as seen in Figure 1.
- Custom BN achieved the highest final test accuracy among the three.
- TensorFlow BN showed comparable performance but slightly lower than No Normalization.
- The difference is marginal (less than 0.2%) and can be considered negligible in practical scenarios.

Figure 1: Training Accuracy vs Epoch (BN Group) This graph shows that both Custom BN and TensorFlow BN lead to faster and higher training accuracy compared to No Normalization.

Figure 2 : Test Accuracy vs Epoch (BN Group) The test accuracy graph shows Custom BN achieving the best results, while TensorFlow BN performs closely to the baseline.

Batch Normalization clearly improves training stability and convergence. In this experiment, Custom BN outperformed both TensorFlow BN and the baseline. TensorFlow BN showed slightly lower test accuracy than No Normalization, but the difference is minimal and not significant for practical purposes.

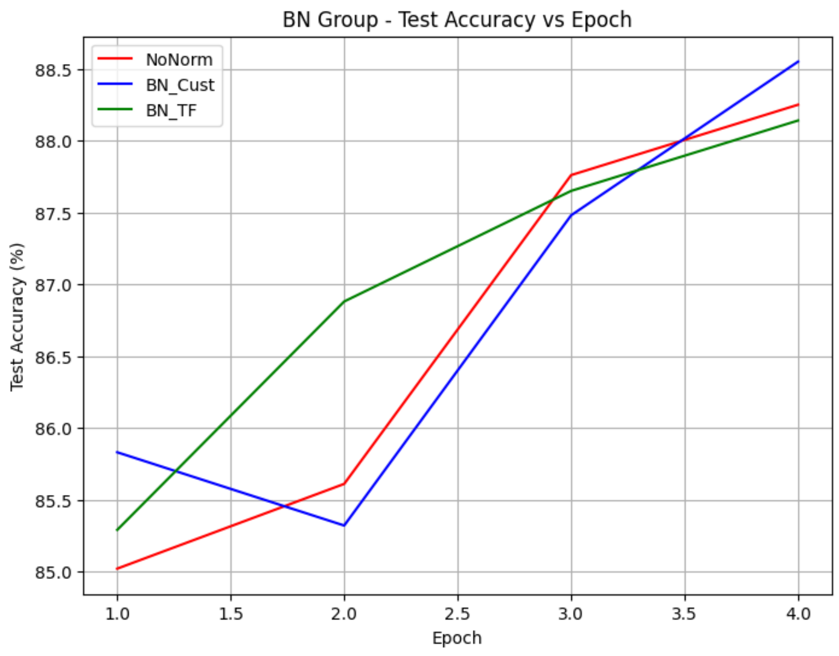

Figure 2: Batch Norm - Testing Accuracy vs Epoch

1.4 Layer Normalization

Layer Normalization (LN) normalizes across the feature dimension within each sample, unlike Batch Normalization which uses batch-wise statistics. LN is more effective in scenarios with smaller batch sizes or where batch statistics may vary significantly.

In our experiments, both Custom LN and TensorFlow LN performed better than the model without normalization. TensorFlow LN achieved the highest final test accuracy, while Custom LN also showed consistent improvement over No Normalization.

Table 2: Final Test Accuracy Comparison (Layer Normalization)

Mode	**Final Test Accuracy**
No Normalization	88.25%
Custom LN	88.29%
TensorFlow LN	88.68%

Observations:

- LN improved both training and test accuracy over the baseline (No Normalization).
- TensorFlow LN achieved the highest test accuracy among the three.
- Custom LN also performed well, with a small difference from TensorFlow LN due to internal optimizations and floating-point handling.
- LN does not depend on batch size, making it stable and suitable across different scenarios.

Figure 3: Training Accuracy vs Epoch (LN Group) This graph shows the training accuracy for No Normalization, Custom LN, and TensorFlow LN. Both LN models show faster convergence and higher accuracy compared to the baseline.

Figure 4: Test Accuracy vs Epoch (LN Group) In this graph, TensorFlow LN achieves the best test accuracy by the end of 4 epochs, followed by Custom LN. No Normalization lags behind both LN models.

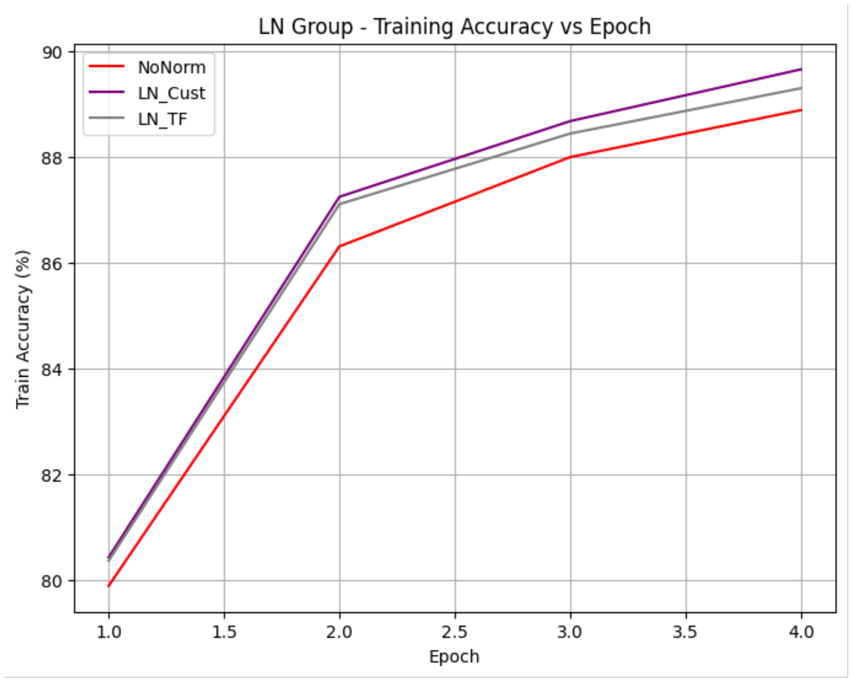

Figure 3: Layer Norm - Training Accuracy vs Epoch

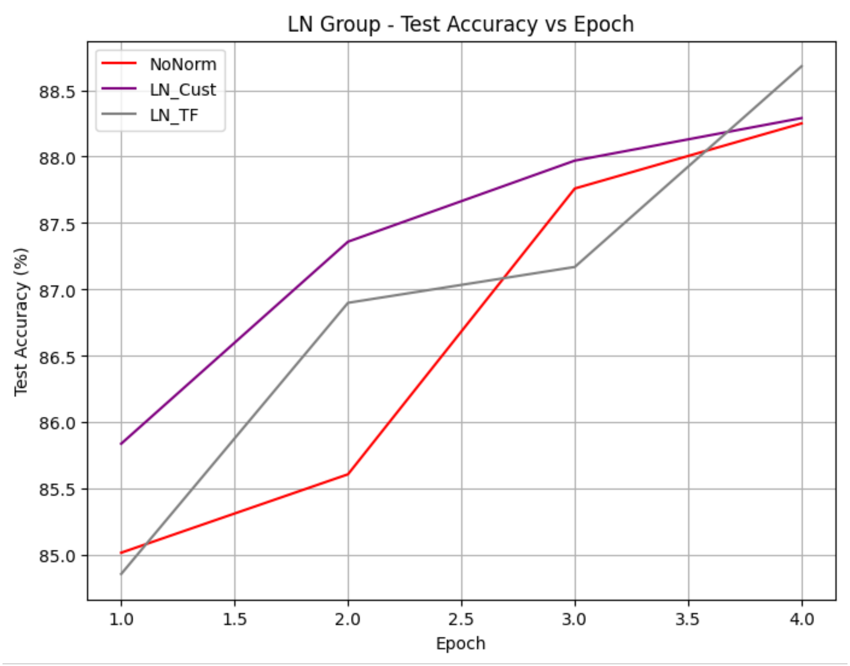

Figure 4: Layer Norm - Testing Accuracy vs Epoch

Layer Normalization provides clear benefits over No Normalization by improving stability and performance. TensorFlow LN performed the best in terms of test accuracy, while Custom LN also showed effective improvement. LN is particularly useful when batch sizes are small or variable, as it normalizes across features independently of the batch.

1.5 Weight Normalization

Weight Normalization (WN) re-parameterizes the weight vectors in the network to decouple the direction and magnitude of weights. This can improve gradient flow and stabilize optimization, especially during the initial stages of training.

In our experiments, the model with Custom Weight Normalization showed slightly better performance compared to No Normalization in terms of training accuracy. However, the test accuracy improvement was marginal.

Table 3: Final Test Accuracy Comparison (Weight Normalization)

Mode	**Final Test Accuracy**
No Normalization	88.25%
Custom WN	88.46%

Observations:

- Weight Normalization helped the model converge faster during training, as shown in Figure 5.
- The difference in test accuracy between WN and No Normalization is very small (0.2% improvement).
- While training accuracy consistently benefited from WN, the test accuracy improvement was less significant, indicating that WN mainly helps in training stability rather than boosting generalization significantly.
- Since WN does not normalize activations but only the weights, its impact is less aggressive compared to BN or LN.

Figure 5: Training Accuracy vs Epoch (WN Group) This graph shows that Weight Normalization leads to faster training convergence compared to No Normalization.

Figure 6: Test Accuracy vs Epoch (WN Group) The test accuracy shows that Custom WN slightly outperforms No Normalization in the final epoch.

Weight Normalization improves training stability and helps in faster convergence. The improvement in test accuracy over No Normalization is marginal but consistent. WN is particularly useful in controlling weight magnitudes during training, although its impact on test performance is less pronounced compared to Batch Normalization or Layer Normalization.

1.6 Comparison of Custom Normalization vs TensorFlow Normalization Functions

In my experiments, I compared the performance of custom normalization implementations (Batch Normalization, Layer Normalization) with TensorFlow's built-in normalization functions.

The results showed that there is no significant performance difference between my custom functions and TensorFlow's functions apart from minor variations due to floating point precision. Both the training and test accuracy trends were almost identical across all epochs.

The slight differences observed (around 0.2% to 0.5% in test accuracy) are expected because TensorFlow's Batch Normalization uses running averages (momentum) for mean and variance calculations, which adds stability over multiple batches. TensorFlow also uses highly optimized internal operations for numerical stability and faster computation.

Since my custom normalization layers produced results very close to TensorFlow's implementation, this confirms that the backward pass and gradient computation in my implementation are correct.

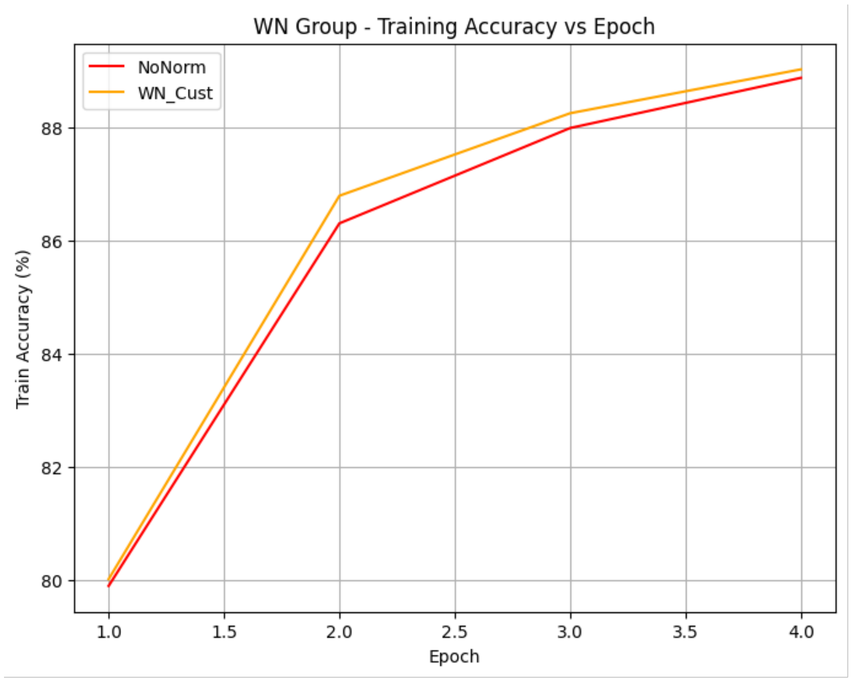

Figure 5: Weight Norm - Training Accuracy vs Epoch

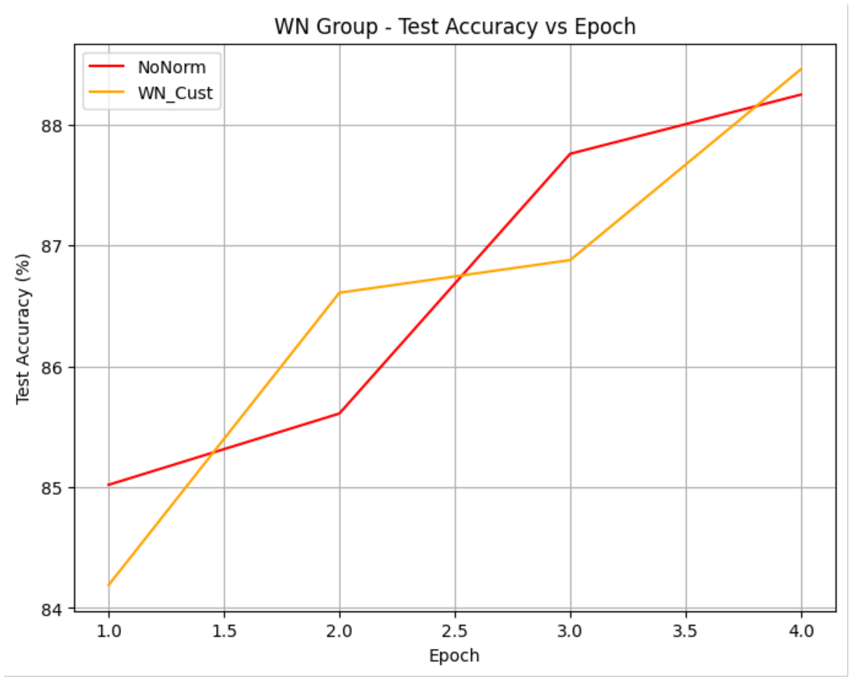

Figure 6: Weight Norm - Testing Accuracy vs Epoch

Findings and Which Normalization is Better Based on the experiments, I observed that Batch Normalization significantly improved training speed and accuracy compared to no normalization. Weight Normalization also helped in faster convergence, but its impact was more noticeable on training stability rather than test accuracy improvement.

Among all methods, Layer Normalization performed the best in terms of final test accuracy. It showed stable performance throughout training and generalized better on unseen data.

1.7 Why Layer Normalization is Better than Batch Normalization

Layer Normalization is better than Batch Normalization in certain scenarios because it does not depend on batch statistics. Batch Normalization computes mean and variance across the mini-batch, so its effectiveness reduces when batch sizes are small or when batch data is not representative. In contrast, Layer Normalization normalizes each individual sample across its features, making it stable even for small batch sizes or variable-length inputs.

This property of LayerNorm makes it particularly suitable for tasks like NLP, RNNs, transformers, and scenarios where batch sizes cannot be large due to memory limitations.

In my experiments, even with a batch size of 100, Layer Normalization outperformed Batch Normalization in terms of final test accuracy. It provided stable and consistent training behavior and achieved the best generalization performance across all normalization methods tested.