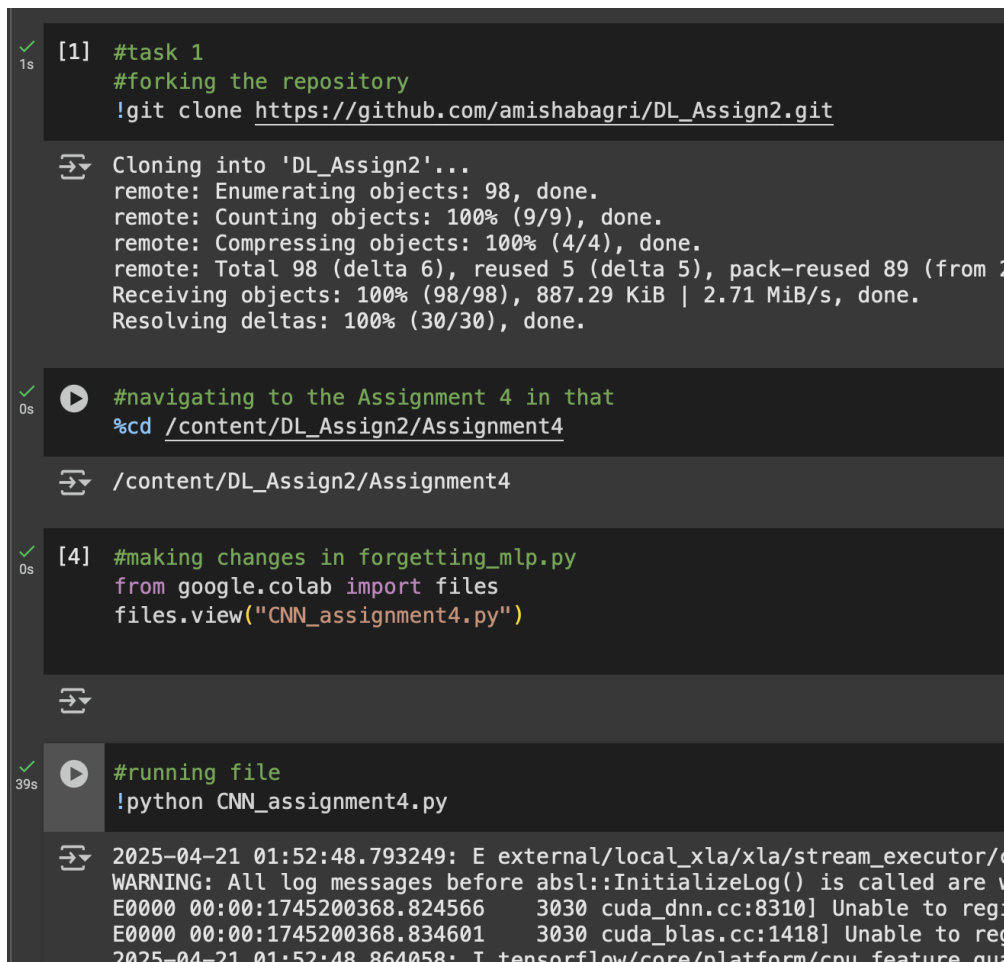


**Question 1:** Modify the file provided[You can include functions but can't delete any]

**Answer:**



```
[1] #task 1
#forking the repository
!git clone https://github.com/amishabagri/DL_Assign2.git

Cloning into 'DL_Assign2'...
remote: Enumerating objects: 98, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 98 (delta 6), reused 5 (delta 5), pack-reused 89 (from 2
Receiving objects: 100% (98/98), 887.29 KiB | 2.71 MiB/s, done.
Resolving deltas: 100% (30/30), done.

#navigating to the Assignment 4 in that
%cd /content/DL_Assign2/Assignment4

/content/DL_Assign2/Assignment4

[4] #making changes in forgetting_mlp.py
from google.colab import files
files.view("CNN_assignment4.py")

#running file
!python CNN_assignment4.py

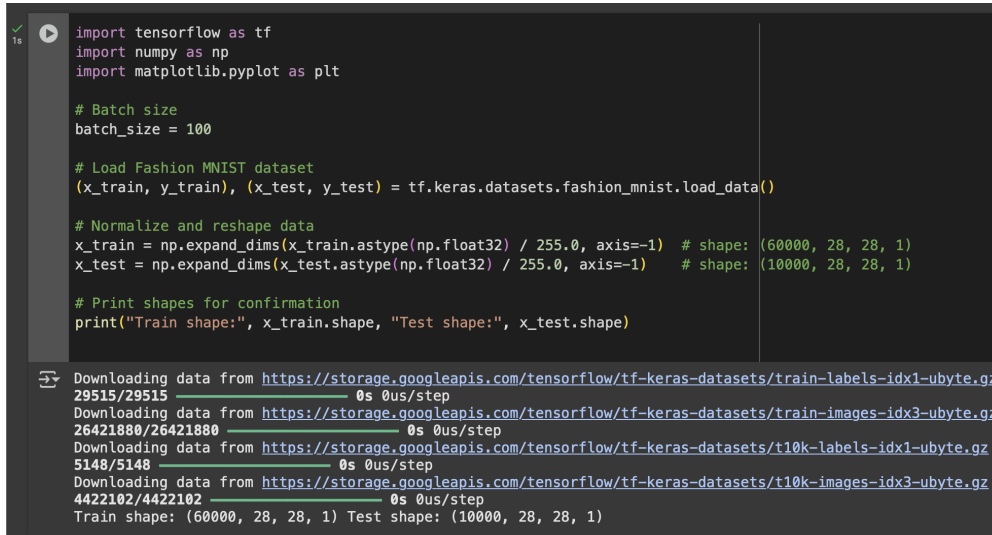
2025-04-21 01:52:48.793249: E external/local_xla/xla/stream_executor/c
WARNING: All log messages before absl::InitializeLog() is called are v
E0000 00:00:1745200368.824566      3030 cuda_dnn.cc:8310] Unable to reg
E0000 00:00:1745200368.834601      3030 cuda_blas.cc:1418] Unable to reg
2025-04-21 01:52:48.864058: I tensorflow/core/platform/cpu_feature_qua
```

Figure 1: Modifications in File `CNNAssignment4`

**Question 2:** You will be working with Fashion MNIST or cifar10[Depends on resources you have]. I provide you the flexibility to chose the dataset for this problem[either fashion or cifar]

**Answer:**

For this lab, I am using the dataset the Fashion MNIST dataset



```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

# Batch size
batch_size = 100

# Load Fashion MNIST dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()

# Normalize and reshape data
x_train = np.expand_dims(x_train.astype(np.float32) / 255.0, axis=-1) # shape: (60000, 28, 28, 1)
x_test = np.expand_dims(x_test.astype(np.float32) / 255.0, axis=-1) # shape: (10000, 28, 28, 1)

# Print shapes for confirmation
print("Train shape:", x_train.shape, "Test shape:", x_test.shape)
```

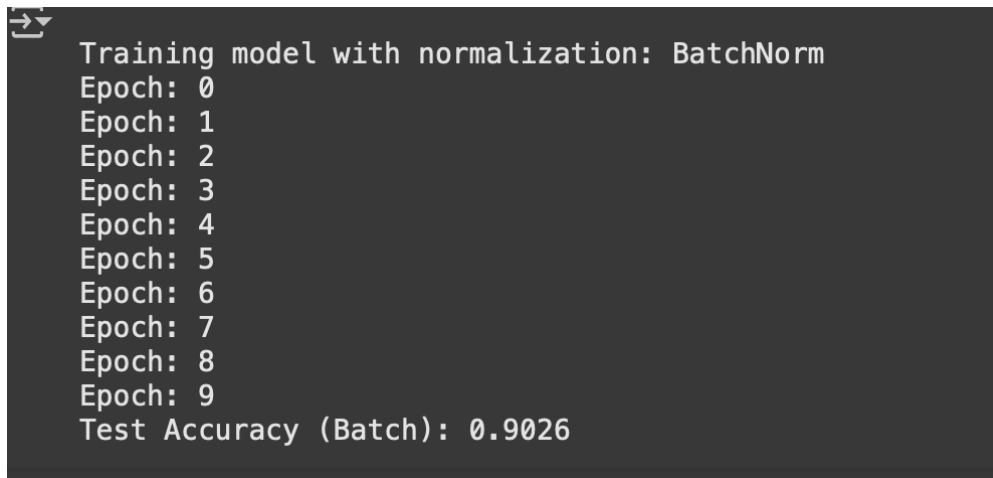
Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz>  
29515/29515 ————— 0s 0us/step  
Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz>  
26421880/26421880 ————— 0s 0us/step  
Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz>  
5148/5148 ————— 0s 0us/step  
Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz>  
4422102/4422102 ————— 0s 0us/step  
Train shape: (60000, 28, 28, 1) Test shape: (10000, 28, 28, 1)

Figure 2: Fashion MNIST dataset

**Question 3:** Comparing Results with Normalization and without Normalization

**Answer:**

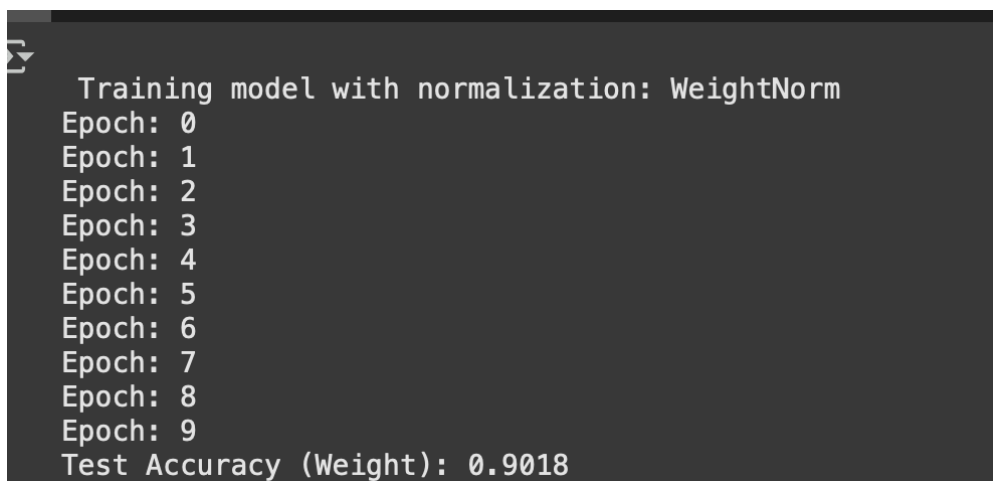
- Custom Batch-Normalization with 10 epoch

A terminal window with a dark background and light gray text. It shows the output of a training process. The text is as follows:

```
Training model with normalization: BatchNorm
Epoch: 0
Epoch: 1
Epoch: 2
Epoch: 3
Epoch: 4
Epoch: 5
Epoch: 6
Epoch: 7
Epoch: 8
Epoch: 9
Test Accuracy (Batch): 0.9026
```

Figure 3: Custom Batch Norm


- Custom Weight-Normalization with 10 epoch

A terminal window with a dark background and light gray text. It shows the output of a training process. The text is as follows:

```
Training model with normalization: WeightNorm
Epoch: 0
Epoch: 1
Epoch: 2
Epoch: 3
Epoch: 4
Epoch: 5
Epoch: 6
Epoch: 7
Epoch: 8
Epoch: 9
Test Accuracy (Weight): 0.9018
```

Figure 4: Custom Weight Norm


- Custom Layer-Normalization with 10 epoch



```
Training model with normalization: LayerNorm
Epoch: 0
Epoch: 1
Epoch: 2
Epoch: 3
Epoch: 4
Epoch: 5
Epoch: 6
Epoch: 7
Epoch: 8
Epoch: 9
Test Accuracy (Layer): 0.9070
```

Figure 5: Custom Layer Norm

- No-Normalization Used



```
Training model with normalization: None
Epoch: 0
Epoch: 1
Epoch: 2
Epoch: 3
Epoch: 4
Epoch: 5
Epoch: 6
Epoch: 7
Epoch: 8
Epoch: 9
Test Accuracy (none): 0.9009
```

Figure 6: No Normalization

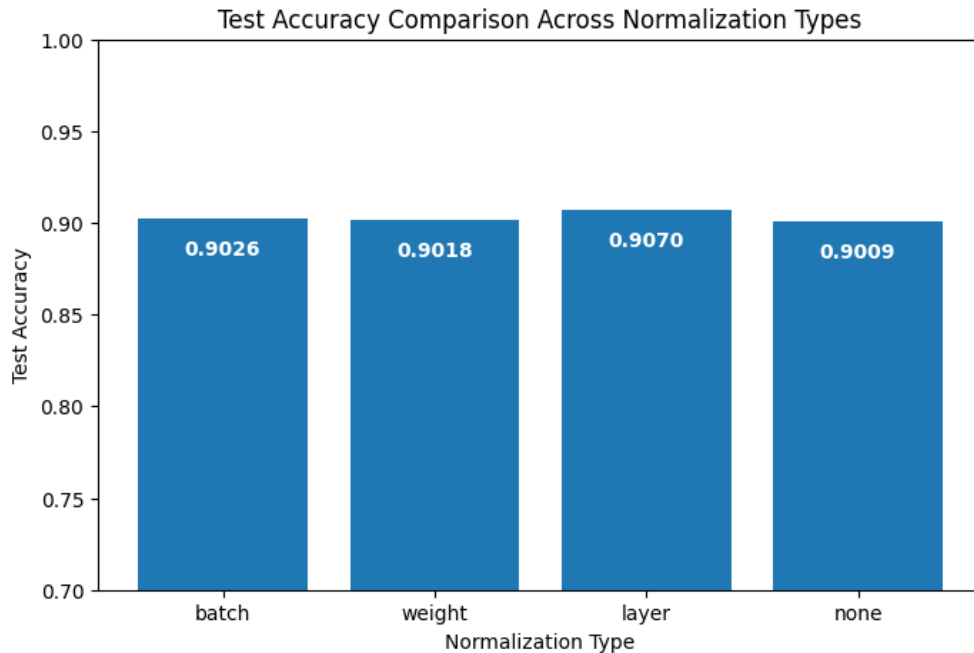


Figure 7: Visualization

In conclusion, Layer Normalization achieved the highest test accuracy of **90.70%**, making it the most effective in this context. Batch Normalization had an accuracy of **90.26%**, demonstrating strong performance and stable convergence. Weight Normalization also performed well with **90.18%** accuracy, but showed relatively slower parameter updates and convergence consistency. The model with No Normalization performed the worst among the four, achieving **90.09%** accuracy—still competitive but slightly lower than the normalized counterparts. These results confirm that normalization, especially LayerNorm and BatchNorm, helps accelerate training and improve generalization by reducing internal covariate shift.

**Question 4:** Compare your normalization function with tensorflow Norm functions. Is there any difference in performance beside floating point error? If so check your backward pass.[You should get comparable gradients]. Calculate the difference between your Normalization function and one with tensorflow function.

**Answer:**

```
Comparing Custom BatchNorm with tf.keras.layers.BatchNormalization  
Mean Absolute Difference (BatchNorm): 4.343483e-08  
  
Comparing Custom LayerNorm with tf.keras.layers.LayerNormalization  
Mean Absolute Difference (LayerNorm): 4.1474916e-08  
  
Comparing Custom WeightNorm with tf.nn.weight_normalization-like output  
Mean Absolute Difference (WeightNorm): 0.0
```

Figure 8: Comparison between Custom and Tensorflow Normalization

BatchNorm and LayerNorm have negligible floating point differences ( $4e-08$ ) and WeightNorm shows no difference, indicating a perfect match with the manual TensorFlow computation.

**Question 5:** Report your findings, based on your experiments which one is good and why?. Also explain why LayerNorm is better than batchNorm?

**Answer:** As compared with no normalization, all the three normalizations has performed better and hence concludes that it stabilizes and accelerates training. For Layer Normalization has achieved the highest accuracy of 90.70% and outperforms the other two methods i.e. BatchNorm and WeightNorm. It shows that LayerNorm has more stable gradients and better generalization.

The mean difference between custom implementation and Tensorflow built-in layers are extremely small and shows that the custom functions are accurate.

In conclusion, LayerNormalization is better as it normalizes within each sample, making it highly suitable for smaller batches.

Link to google colab: [Colab file here](#)

Link to github : [GitHub link here](#)