Deep Learning

(CSL7590)

ASSIGNMENT 1

Topic: Build a Neural Network from Scratch

Submitted By:

Shivani Tiwari (M24CSA029)

Submitted To:

Dr. Deepak Mishra

Code Link:
https://colab.research.google.com/drive/1WMZjRlNuo_hB5EF0bpIP-KtbtGaj0avL?usp=sharing

In this assignment, I Build a Neural Network from Scratch in python. The Neural Network Structure is as followed:

1. The Network architecture contains 2 hidden layers, input layer and output layer.
    - Input layer : Flatten Images of 784 features (28x28).
    - Hidden layer 1 : 128 neurons
    - Hidden layer 2 : 64 neurons
    - Output layer : 10 neurons for 10 classes
2. The weights are randomly initialized using a seed value as 29 (M24CSA029) and biases are initialized with 1.
3. Weight Initialization techniques are also used like Xavier and He initialization **(Bonus)**
4. Train - Test split ratio is 70:30, 80:20, 90:10.
5. Batch Size is 24 (M24CSA029).
6. Cross entropy loss is used as a loss function, it is appropriate for multi-class classification.
7. Different Gradient Descent algorithms are used for optimization techniques : Batch Gradient Descent , Stochastic Gradient Descent , Mini-Batch Gradient Descent and plotted accuracy and loss per epoch for each algorithm.
8. 'Relu' is used as an activation function for hidden layers and 'softmax' for output layer.
9. Other activation functions like 'sigmoid' and 'tanh' is also used in hidden layers **(Bonus)**
10. Trained for 25 epochs.
11. Prepared a Confusion matrix for all the combinations of the network.
12. Used L2 regularization to prevent overfitting **(Bonus)**.

Methodology:

1. **Dataset**
    - Imported MNIST Dataset from Scikit learn library.
    - Normalized the data in the range [0,1].
    - Flattened the images into 784 (28x28) input value.
    - One-hot encoded the labels.
2. **Splitting Data**
    - Split data into training and testing set 70:30, 80:20, 90:10.
    - Data is shuffled before training to avoid overfitting.
3. **Neural Network architecture**
    - Input layer : 784 neurons
    - Hidden 1 layer : 128 neurons with Relu activation function
    - Hidden 2 layer : 64 neurons with Relu activation function.
    - Output layer : 10 neurons with softmax activation function.
4. **Gradient Descent algorithms**
    - Batch Gradient Descent : weights updates were done after computing gradients on the entire training set.
    - Stochastic Gradient Descent : weights updates were done after computing gradients on individual samples.
    - Mini-Batch Gradient Descent : weights updates were done using small batch size = 24
5. **Weight Initialization**
    - Initialized weights with random seed value =29, and biases are set to be as 1.
    - Also included Xavier and He initialization techniques depending on the different activation function. (Bonus)
    - Xavier initialization formula

$$W_i \sim \mathcal{N}(0, \sqrt{\frac{2}{\mathrm{n_{in}} + \mathrm{n_{out}}}})$$

    - He initialization

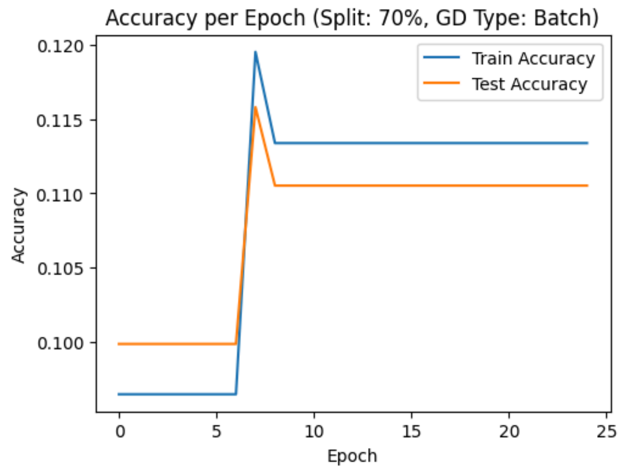
$$W_i \sim \mathcal{N}(0, \sqrt{\frac{2}{\mathrm{n_{in}}}})$$

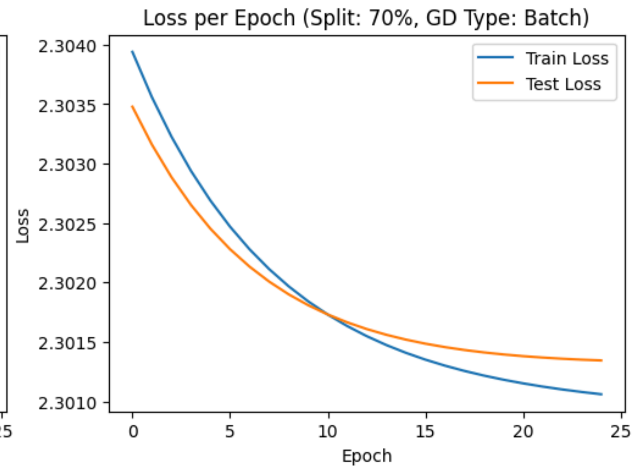
6. **Forward Propagation**
    - Different activation functions are used in the hidden layers like sigmoid, Relu, tanh. Output layers contain only the softmax activation function because it is a multi class classification.
    - Conducts forward propagation through a neural network.
    - Computes activations and intermediate outputs for each layer using weights and biases.

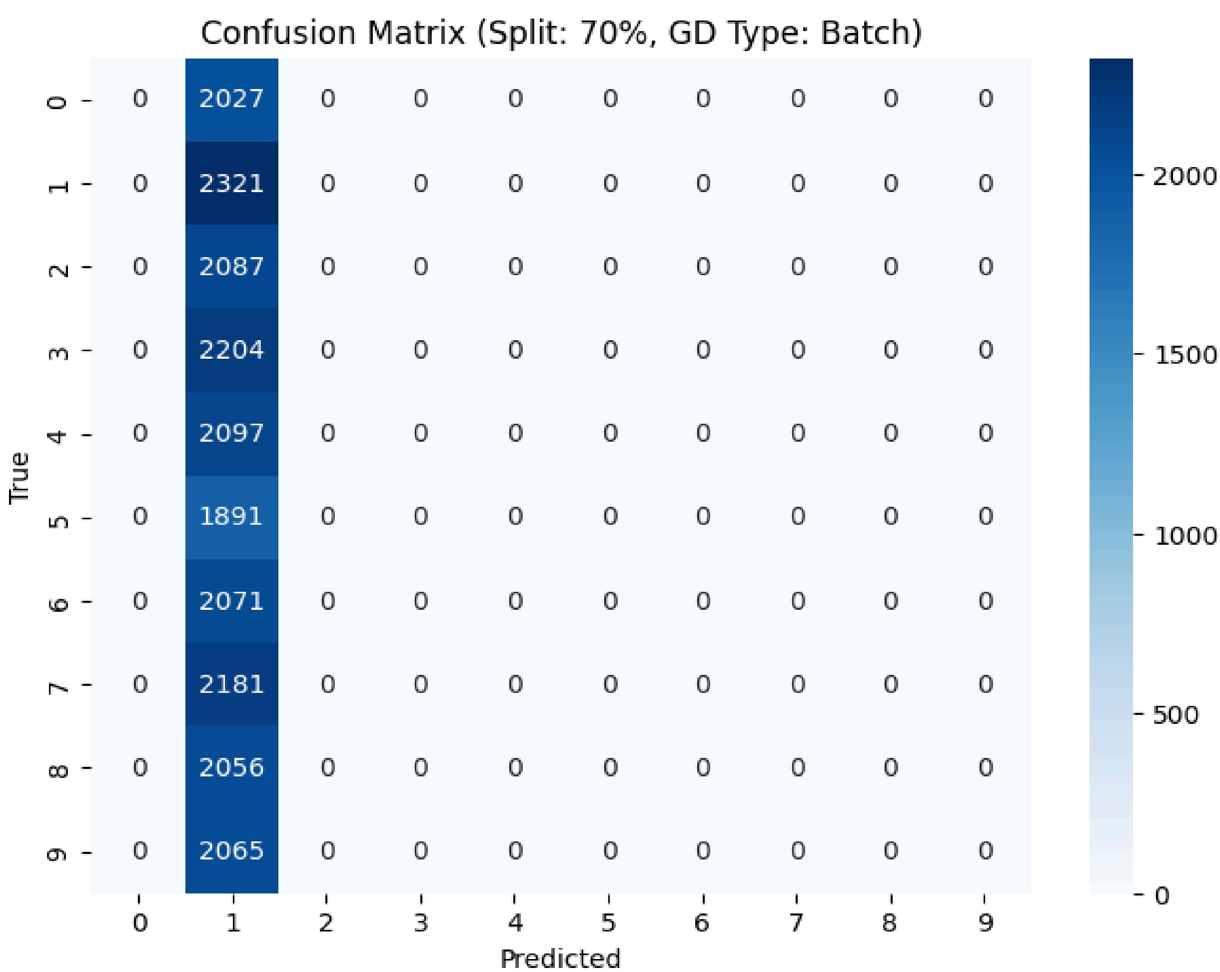
7. **Backward Propagation**
    - Implemented backpropagation to update weights and biases based on calculated gradients.
    - Computes errors, gradients and updates weights and biases using different gradient descent algorithms.
    - Incorporated L2 regularization to avoid overfitting (Bonus)
8. **Training the neural network**
    - Neural network trained using the provided training data and specified hyperparameters.
    - Conducted training for specified number of epochs and updated weights and biases.
    - Calculated and store training losses and accuracies per epoch.

Results:

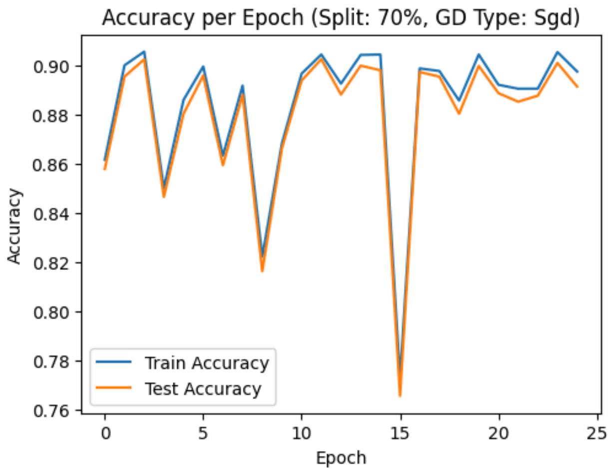
1. **Train-test split ratio 70:30**
    - Plotted confusion matrix, accuracy and loss per epoch for Batch Gradient descent.

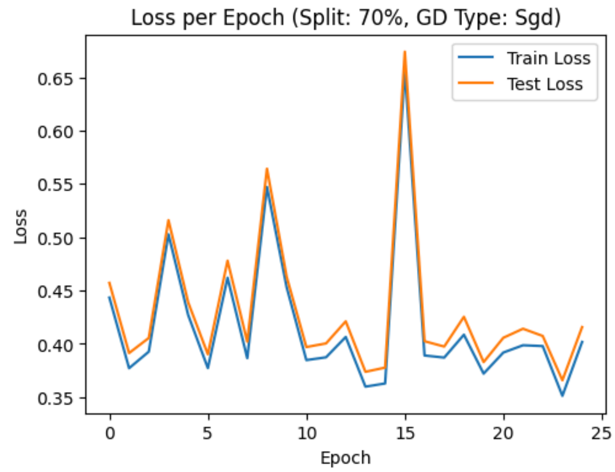


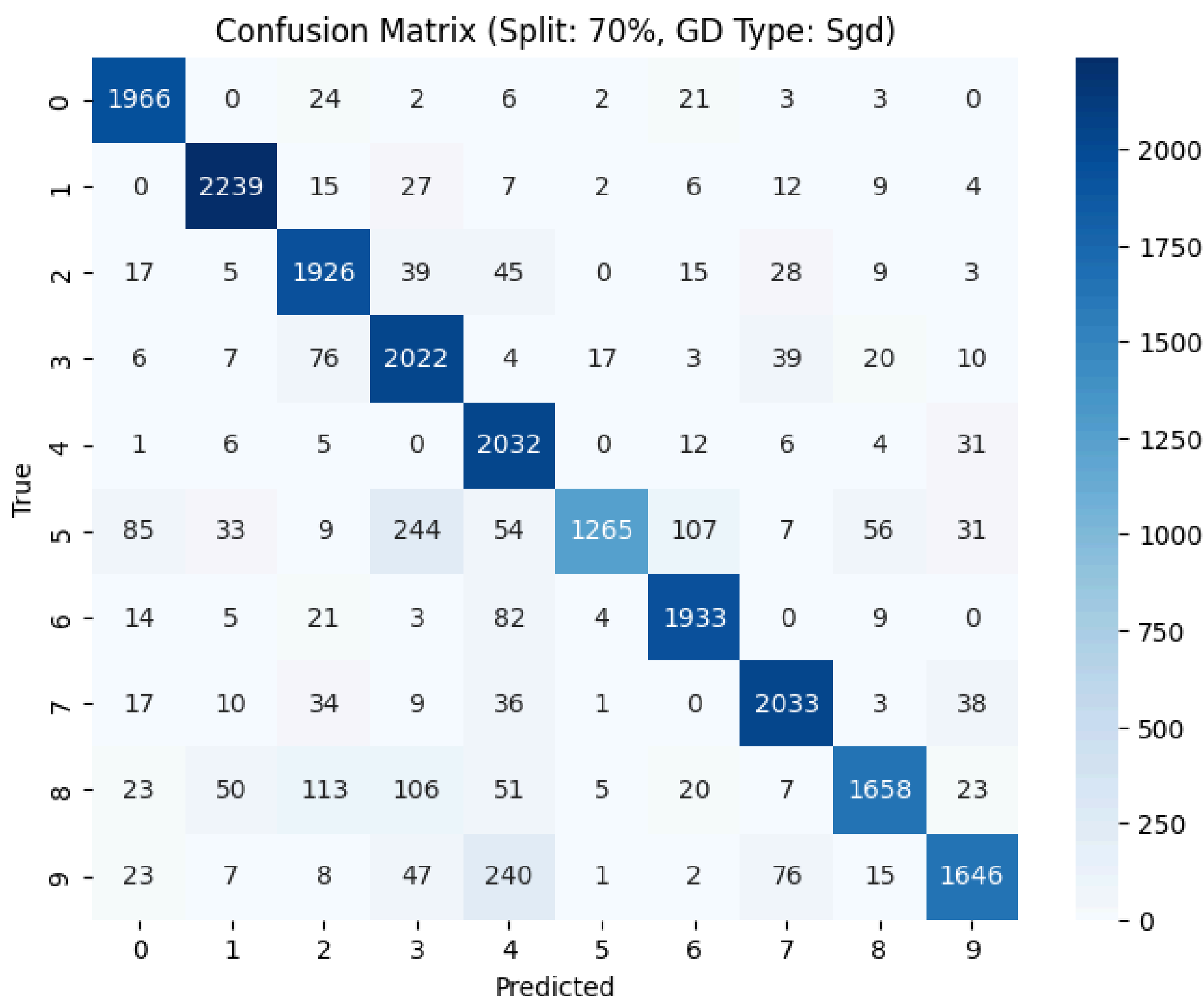




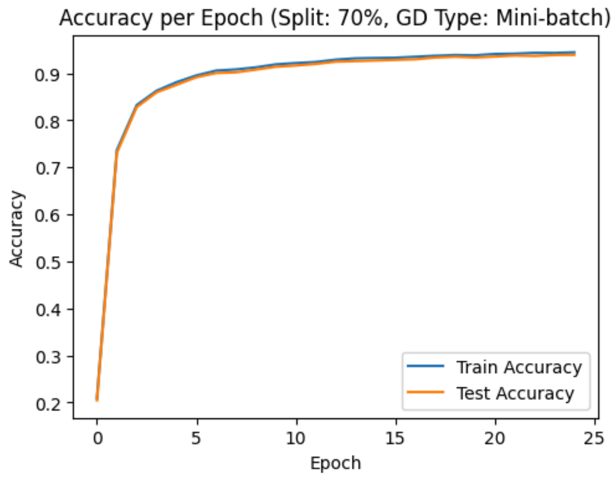
- Plotted confusion matrix, accuracy and loss per epoch for Stochastic Gradient descent.

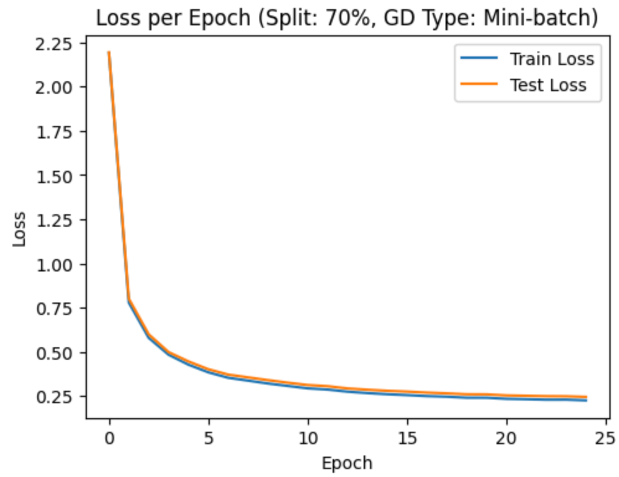


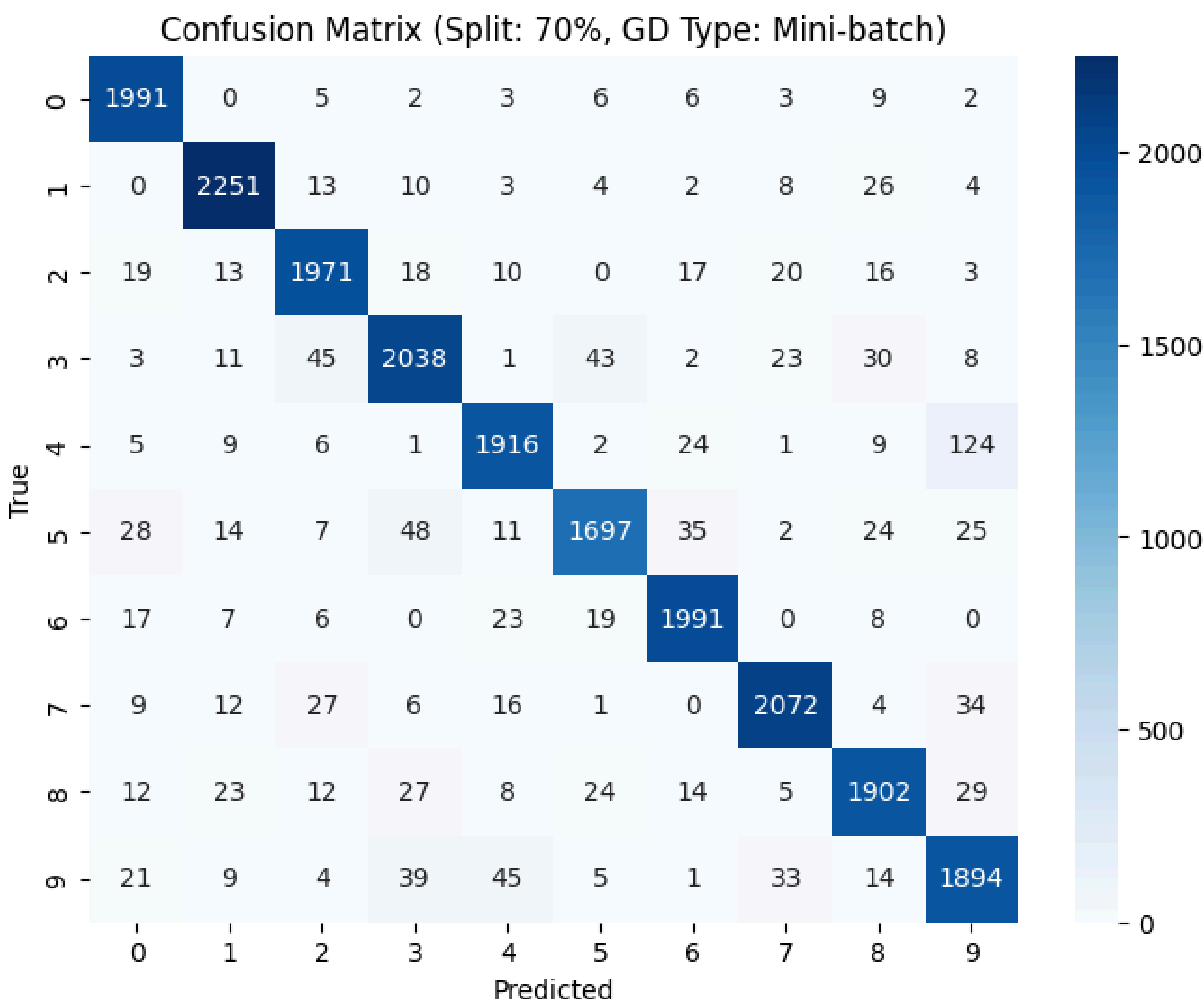




- Plotted confusion matrix, accuracy and loss per epoch for Mini-batch Gradient descent.



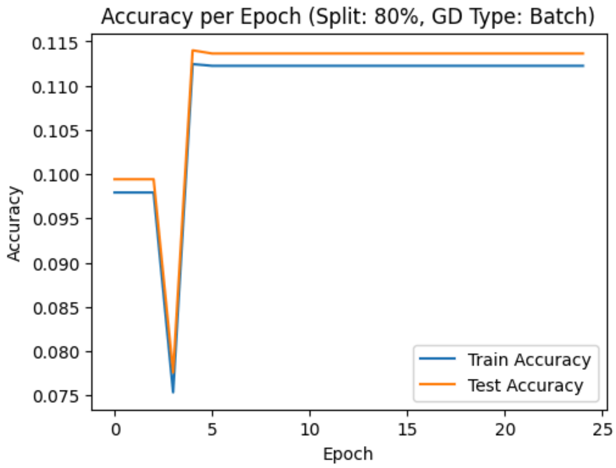


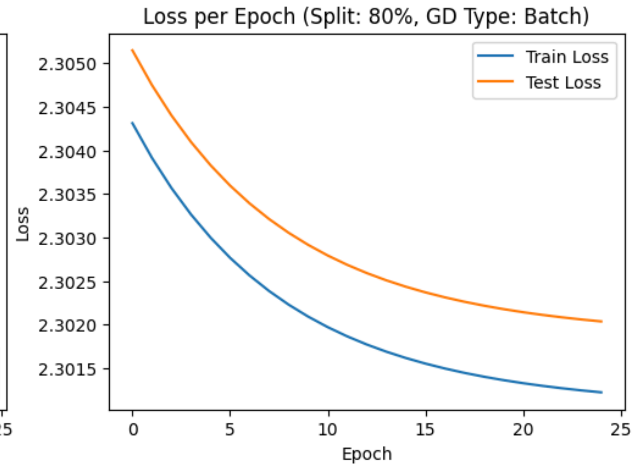


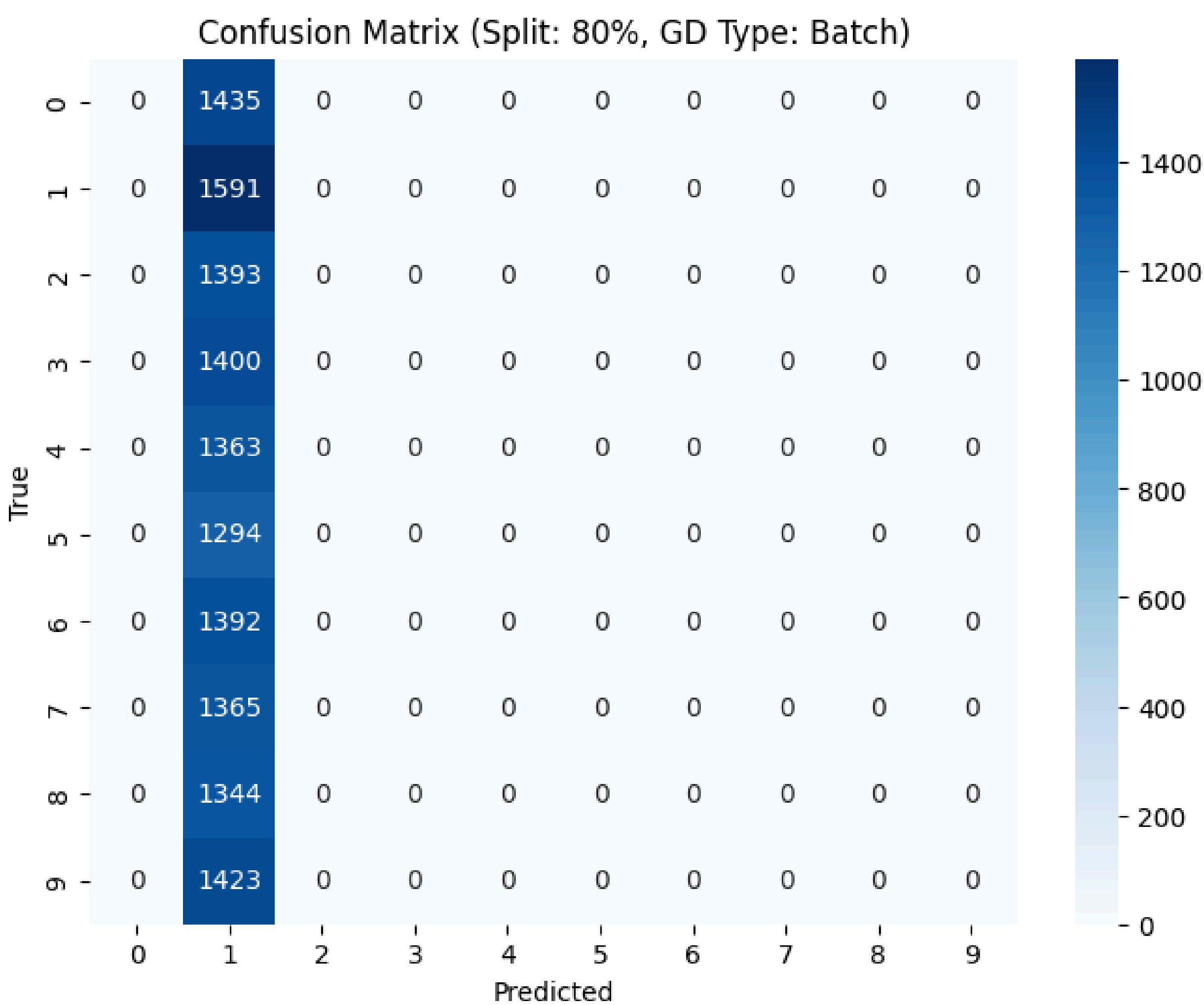
1. Split Ratio: 70:30

- Batch Gradient Descent:
    - Training Accuracy: Showed slight improvement compared to without L2 (~11% from ~10%), though overall learning remained poor.
    - Test Accuracy: Improved slightly (~11%), but still inadequate.
    - Loss: A minor reduction was observed due to L2's weight penalization, but learning was still ineffective.
- Stochastic Gradient Descent :
    - Training Accuracy: Slightly lower compared to non-regularized SGD, as L2 penalizes weights. Final accuracy stabilized at ~87%-89%.
    - Test Accuracy: Improved generalization was observed with test accuracy aligning closely with training accuracy (~89%).
    - Loss: Reduced loss values over epochs, confirming L2's stabilizing effect.
- Mini-batch Gradient Descent:
    - Training Accuracy: Improved gradually to ~94%, slightly lower than without regularization due to penalization.
    - Test Accuracy: Demonstrated stronger generalization, achieving ~89%-94%.
    - Loss: Smooth and consistent convergence, with lower fluctuations compared to non-regularized training.

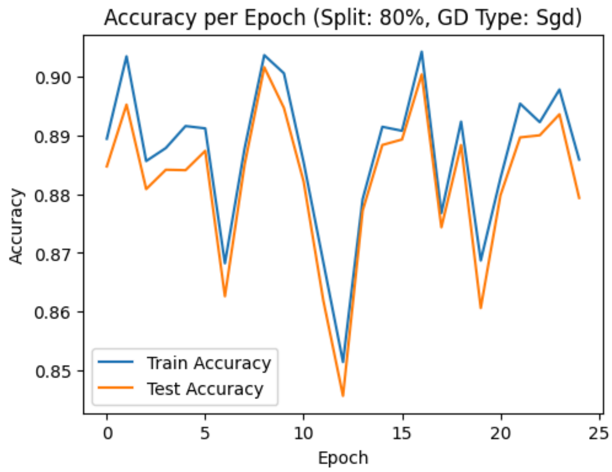
2. **Train-test split ratio as 80:20.**
   - Plotted confusion matrix, accuracy and loss per epoch for Batch Gradient descent.

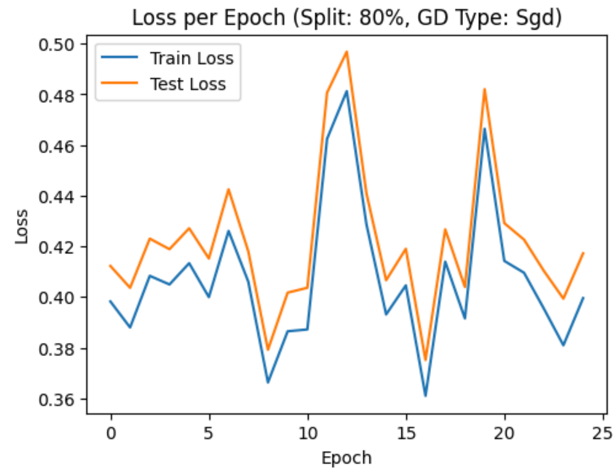


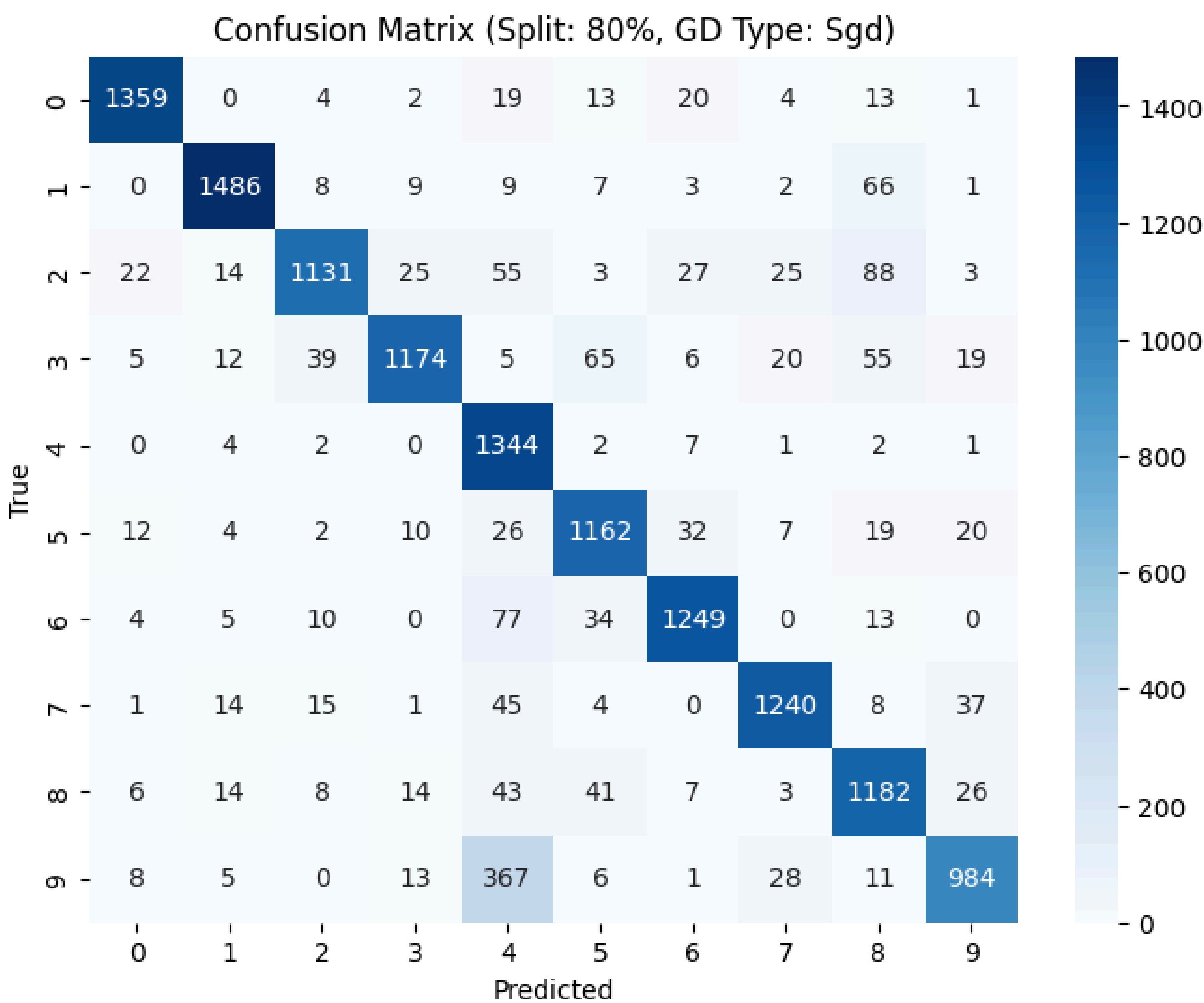




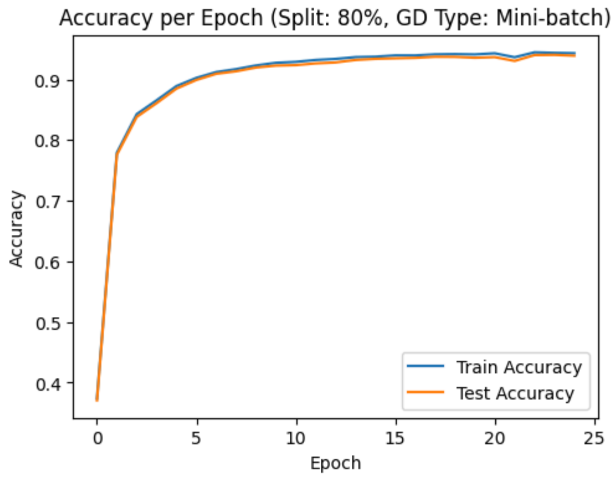
- Plotted confusion matrix, accuracy and loss per epoch for Stochastic Gradient descent.

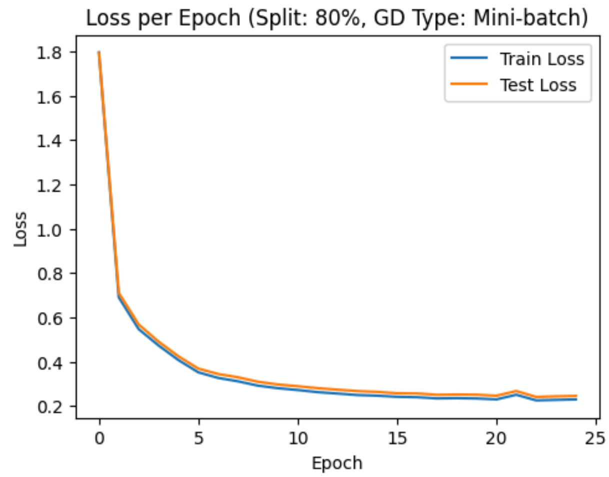


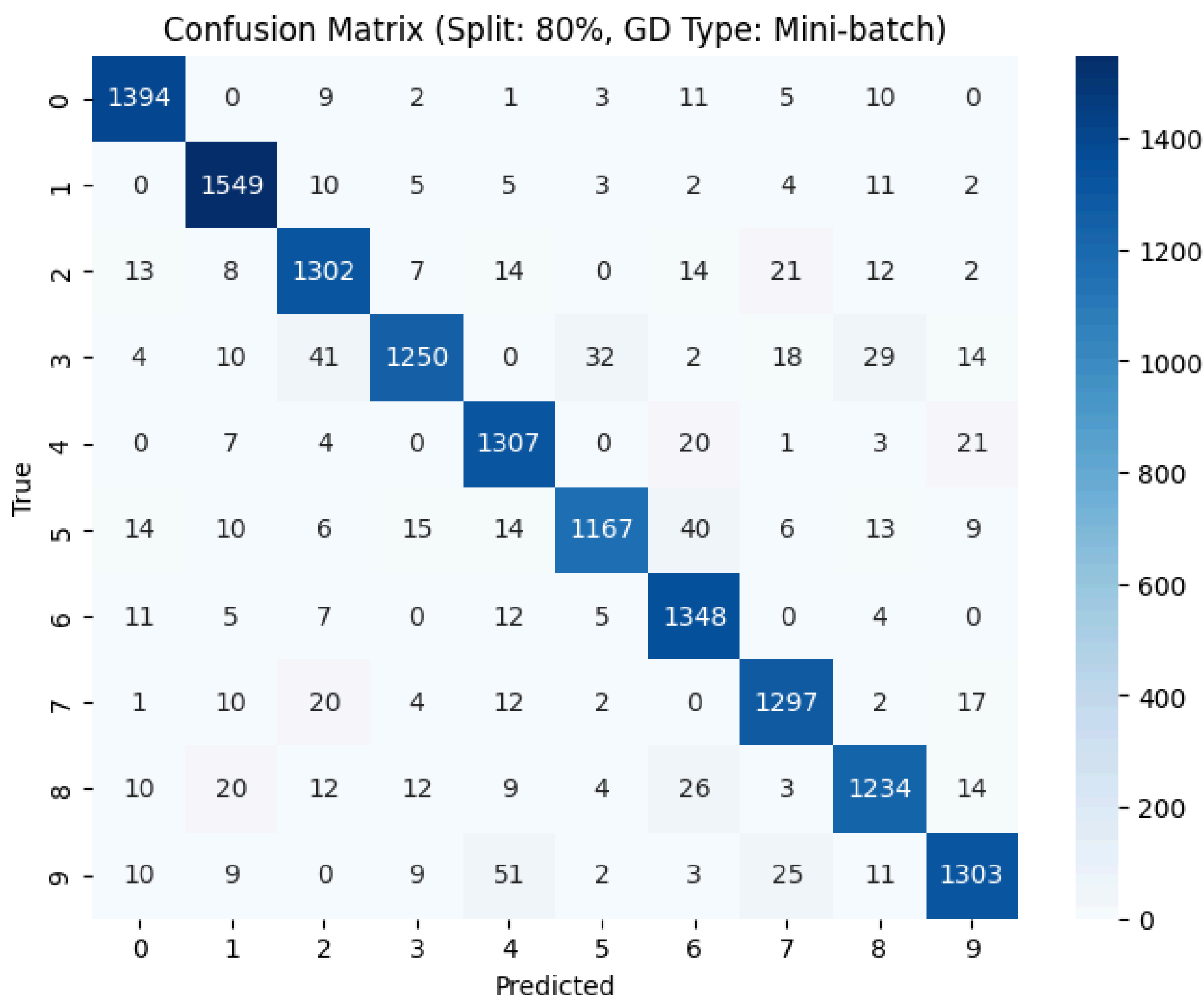




- Plotted confusion matrix, accuracy and loss per epoch for Mini-batch Gradient descent.





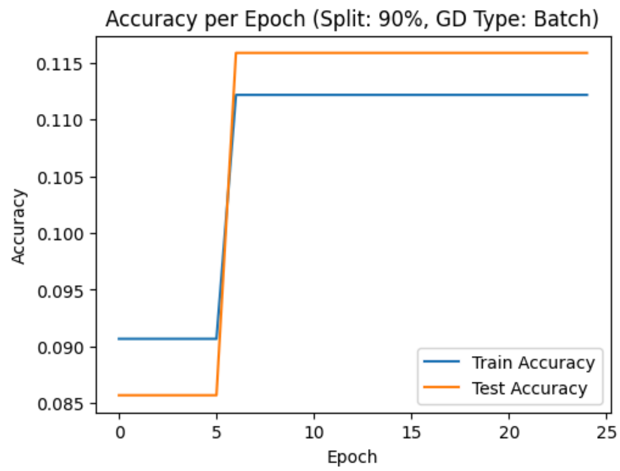


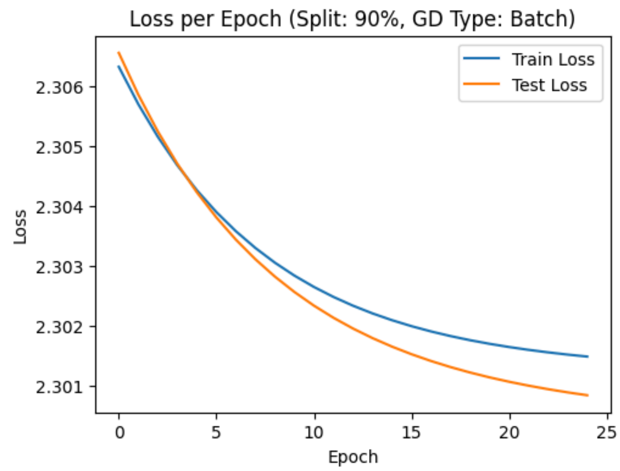
2. Split Ratio: 80:20

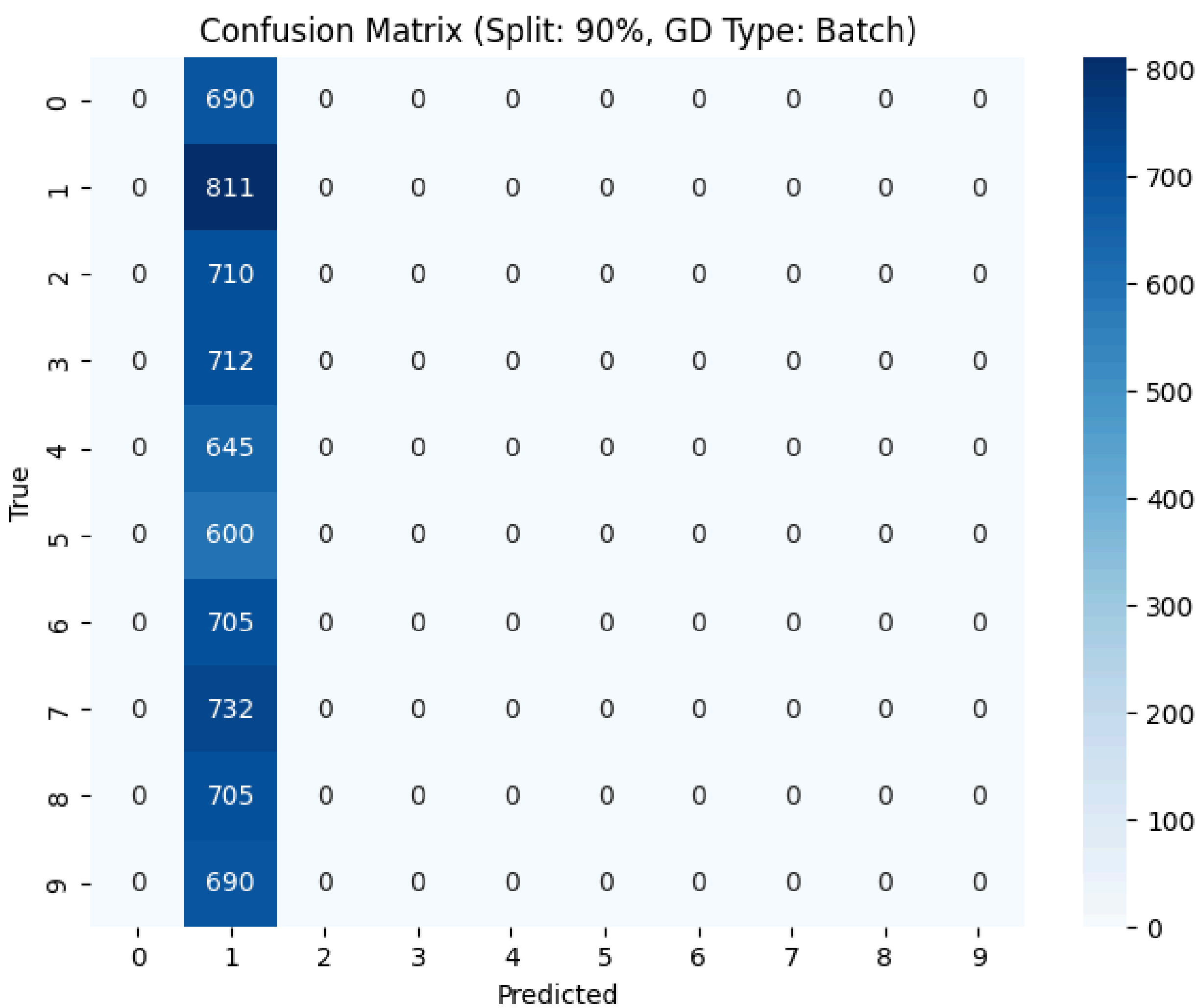
- Batch Gradient Descent :
    - Training Accuracy: Improved marginally (~11%), but still no significant learning observed.
    - Test Accuracy: Slight increase (~9%-11%), though performance remained poor.
    - Loss: L2 regularization contributed to slightly reduced loss, but the model failed to converge meaningfully.
- Stochastic Gradient Descent:
    - Training Accuracy: Stabilized around ~87%-88%, slightly lower than without L2 due to the regularization effect.
    - Test Accuracy: Improved generalization, achieving ~86%-87% accuracy.
    - Loss: Consistent reduction in loss values, with smoother convergence compared to non-regularized SGD.
- Mini-batch Gradient Descent:
    - Training Accuracy: Gradually reached ~94%, slightly lower than without L2.
    - Test Accuracy: Showed strong generalization, achieving ~93%-94%.
    - Loss: Regularized weights resulted in smoother and more stable convergence.

3. **Train-test split ratio as 90:10**

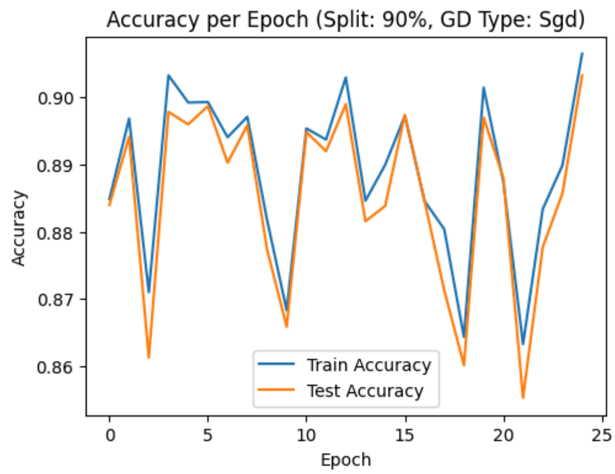
- Plotted confusion matrix, accuracy and loss per epoch for Batch Gradient descent.

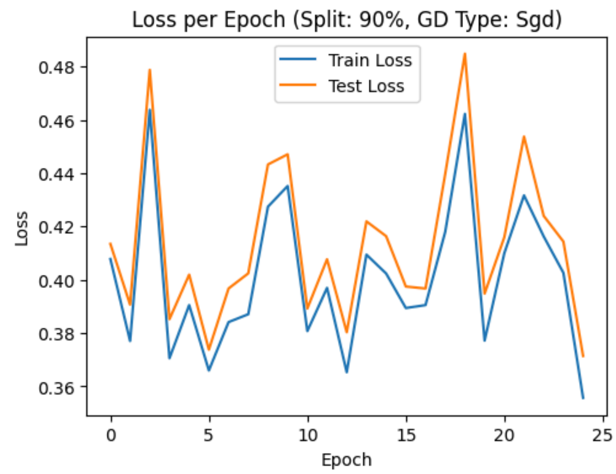


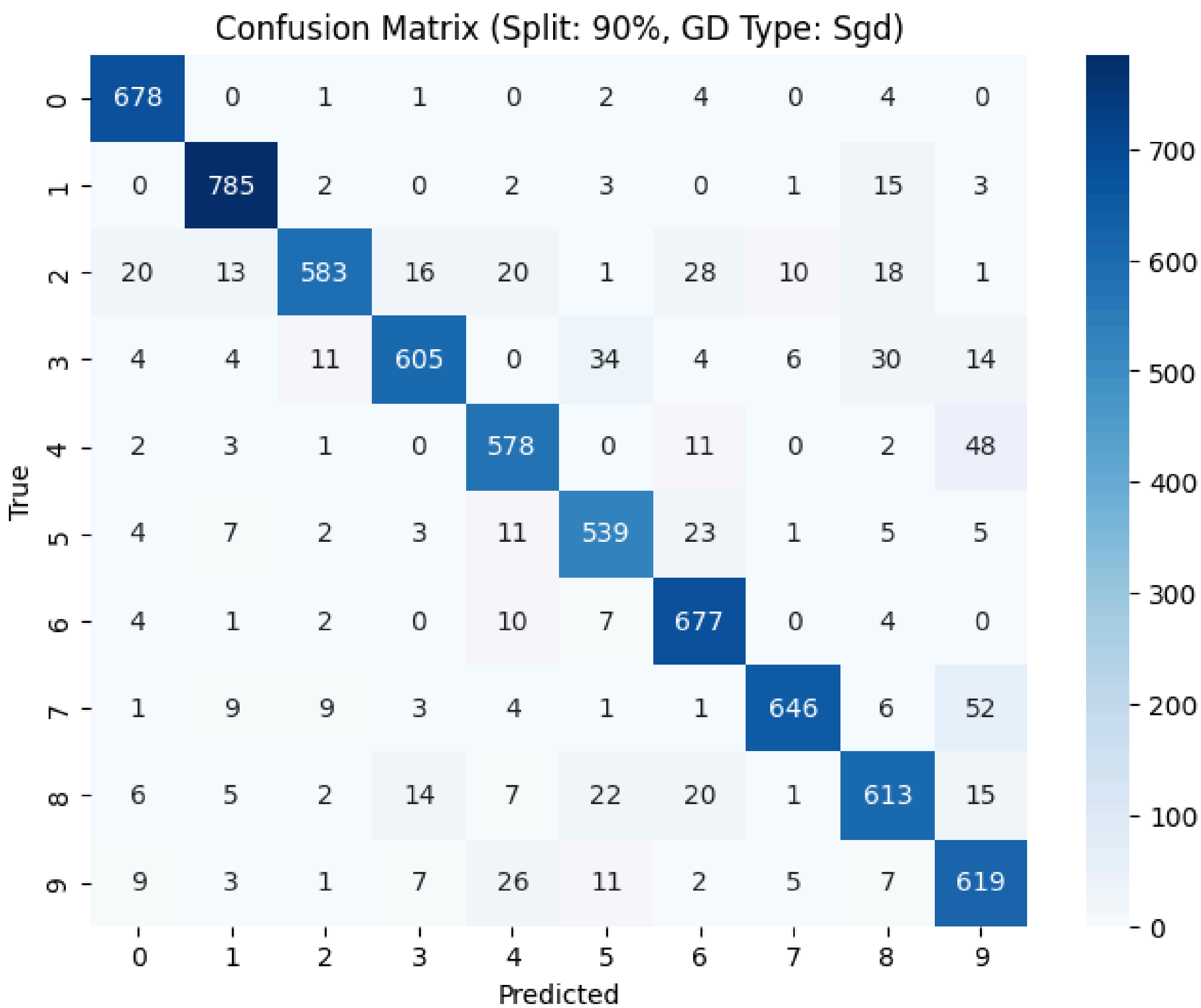




- Plotted confusion matrix, accuracy and loss per epoch for Stochastic Gradient descent.

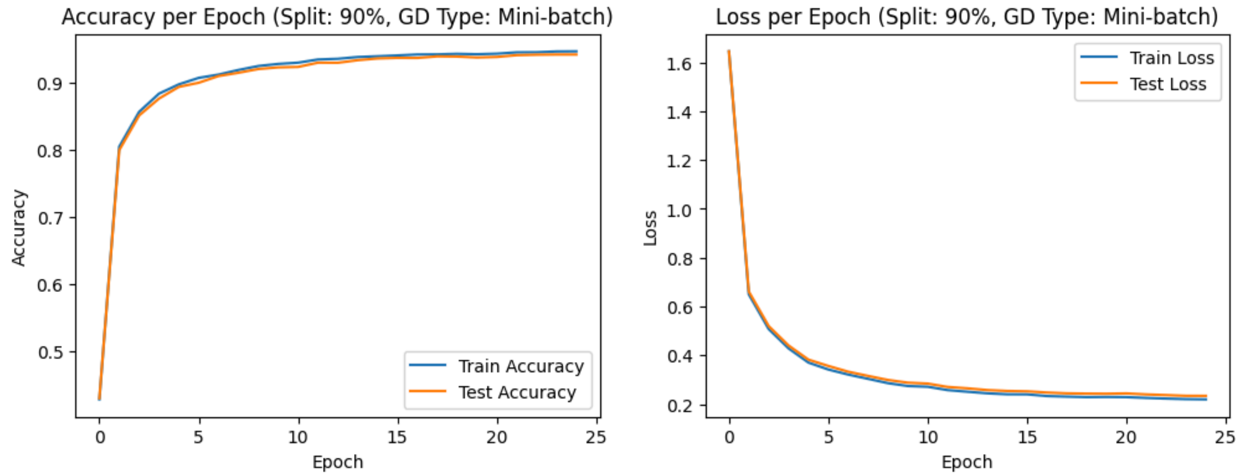


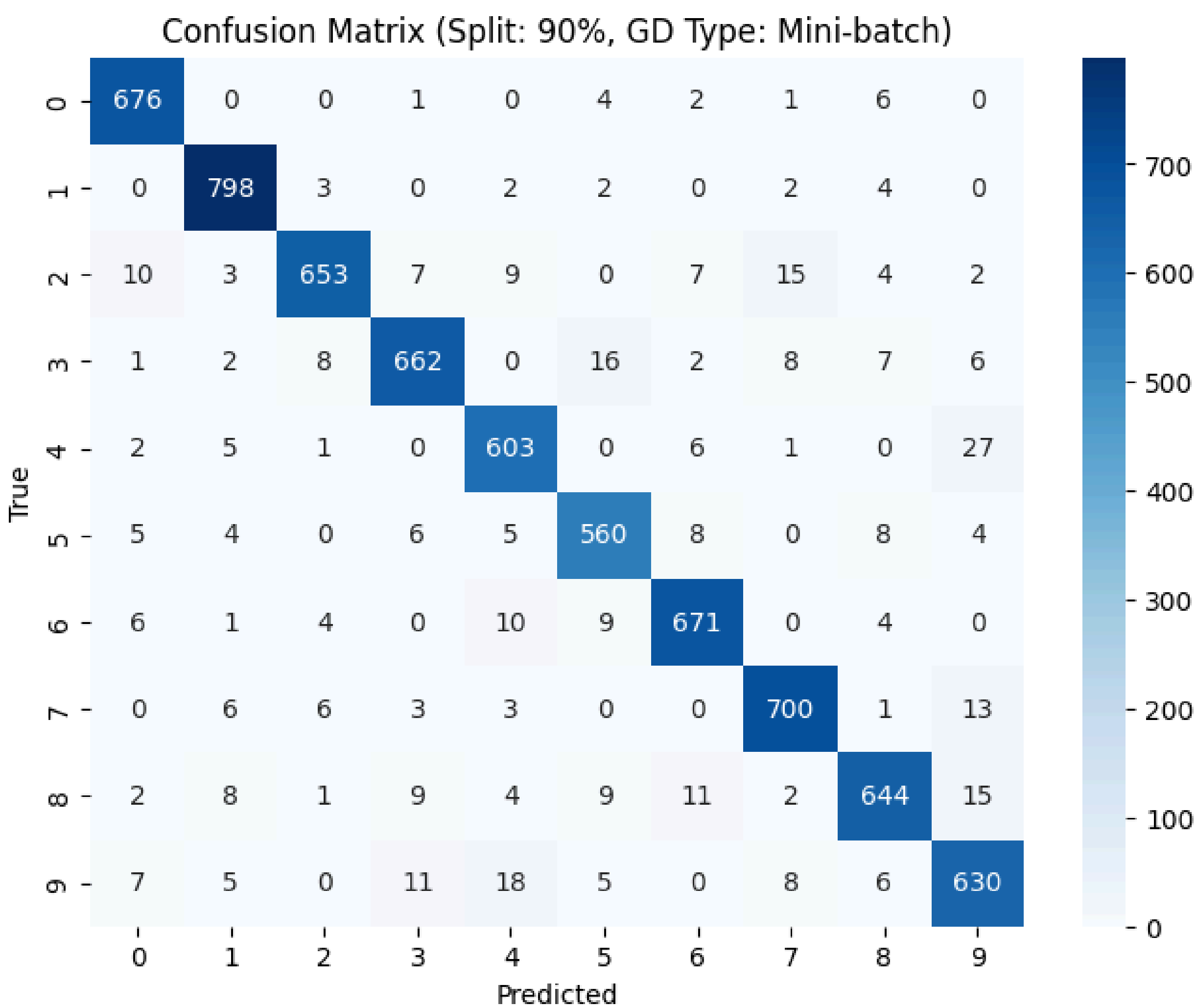




True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	678	0	1	1	0	2	4	0	4	0
1	0	785	2	0	2	3	0	1	15	3
2	20	13	583	16	20	1	28	10	18	1
3	4	4	11	605	0	34	4	6	30	14
4	2	3	1	0	578	0	11	0	2	48
5	4	7	2	3	11	539	23	1	5	5
6	4	1	2	0	10	7	677	0	4	0
7	1	9	9	3	4	1	1	646	6	52
8	6	5	2	14	7	22	20	1	613	15
9	9	3	1	7	26	11	2	5	7	619

- Plotted confusion matrix, accuracy and loss per epoch for Mini-batch Gradient descent.





3. Split Ratio: 90:10

- Batch Gradient Descent:
  - Training Accuracy: Accuracy = ~11%, but the method remained ineffective for learning meaningful patterns.
  - Test Accuracy: Slight improvement (~12%), but performance remained poor overall.
  - Loss: Minimal reduction, similar to other split ratios, with limited convergence.
- Stochastic Gradient Descent:
  - Training Accuracy: Reached ~90%-91%, slightly lower than non-regularized training due to penalization.
  - Test Accuracy: Achieved ~90%, with improved generalization due to L2.
  - Loss: Reduced loss values with smoother convergence.
- Mini-batch Gradient Descent:
  - Training Accuracy: Gradually improved to ~94%-95%.
  - Test Accuracy: Achieved ~94%-95%, showing the best generalization among all methods.
  - Loss: The addition of L2 regularization resulted in smoother and more stable loss reduction, confirming the improved training dynamics.

Number of Trainable and Non-trainable parameters:

- **Trainable parameters:** Weights and biases in the neural networks layers, it is equal to 109386.
- **Non-Trainable parameters**: Direct connection from input to output connections.

Table:

Split Ratio	**Gradient Descent Type**	**Trainable parameters**	**Non-trainable Parameters**	**Train accuracy**	**Test accuracy**
70:30	Batch GD	109386	7840	11.34%	11.05%
70:30	Stochastic GD	109386	7840	89.76%	89.14%
70:30	Mini-Batch GD	109386	7840	94.45%	93.92%
80:20	Batch GD	109386	7840	11.23%	11.36%
80:20	Stochastic GD	109386	7840	88.59%	87.94%
80:20	Mini-Batch GD	109386	7840	94.35%	93.94%
90:10	Batch GD	109386	7840	11.22%	11.59%
90:10	Stochastic GD	109386	7840	90.65%	90.33%
90:10	Mini-Batch GD	109386	7840	94.72%	94.24%

Observations:

1. Batch Gradient Descent:
- It performs poorly across all split ratios, and fails to learn meaningful patterns. Both training and test accuracy remained low (~11%-12%), with very low loss reduction.
2. Stochastic Gradient Descent (SGD):
- It performs well across all split ratios, with training and test accuracy stabilizing around ~90%.

- Fluctuations in accuracy and loss were observed due to the noisy nature of SGD updates.
- A higher training split (90%) improved performance slightly, achieving ~91% accuracy.

3. <u>Mini-batch Gradient Descent:</u>

- Consistently outperformed the other methods across all split ratios.
- Achieved the highest accuracy (~94%) and lowest loss, with strong generalization to the test set.
- Performance improved with a higher training split, as observed in the 90:10 ratio.

4. <u>Impact of Split Ratios:</u>

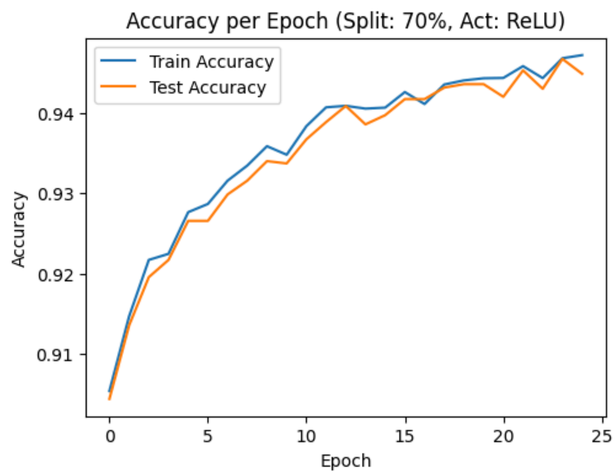
- A larger training split (90:10) consistently yielded better performance for all gradient descent methods.
- Mini-batch Gradient Descent was most effective in utilizing the additional training data, achieving the best results in the 90:10 split.

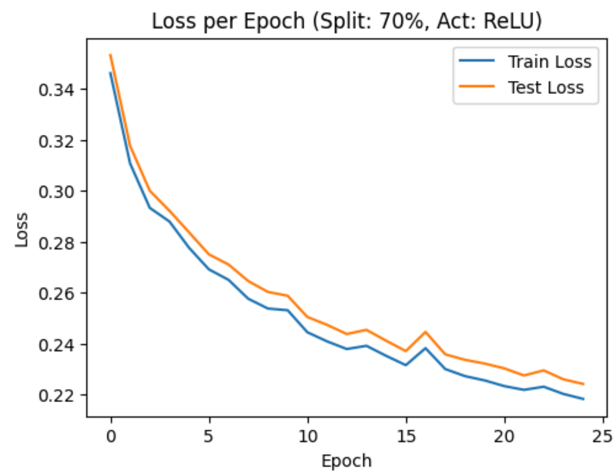
5. <u>Effect of L2 Regularization:</u>

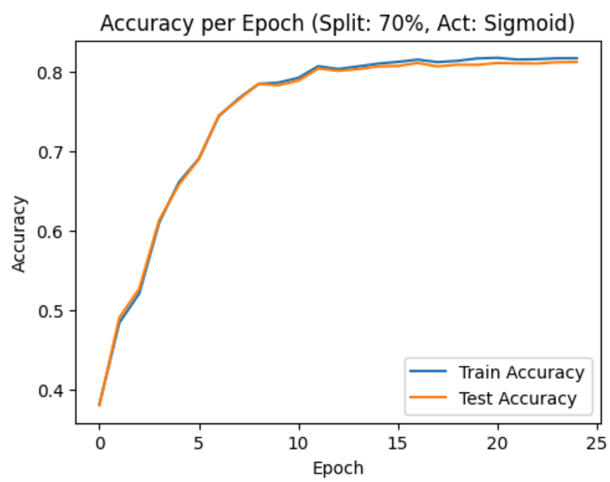
- Introduced weight penalization, preventing weights from growing too large and overfitting the data.
- Improved generalization across all gradient descent methods, particularly for SGD and Mini-batch.
- Minor reduction in training accuracy due to penalization but improved test accuracy, highlighting better generalization.

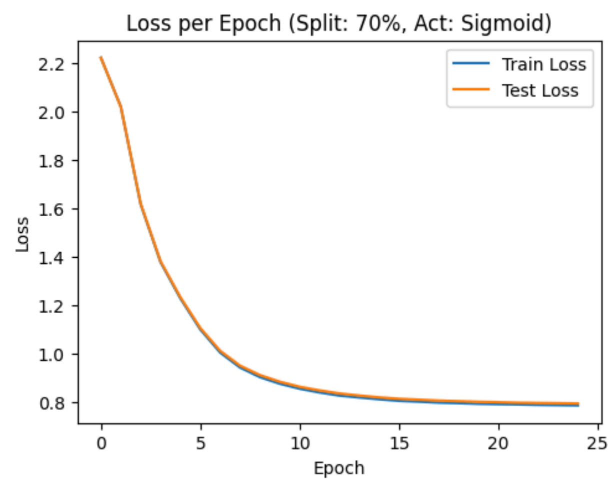
Different activation function with latest weight initialization techniques (Bonus part)

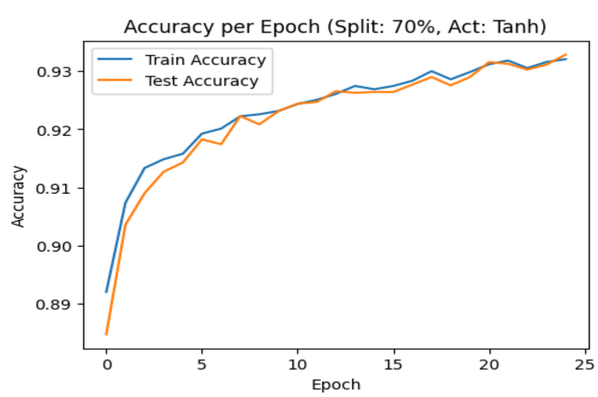
1. **Train-Test split ratio as 70:30 :**
    - For Relu, He initialization techniques is used
    - For Sigmoid and Tanh, Xavier initialization is used

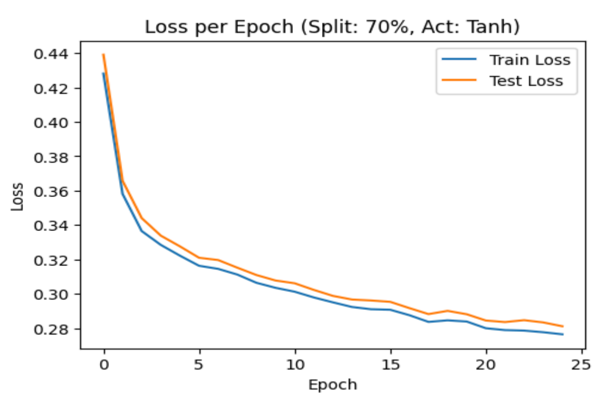




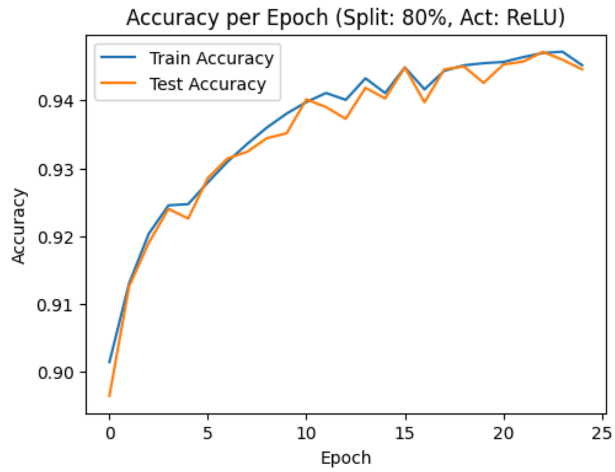


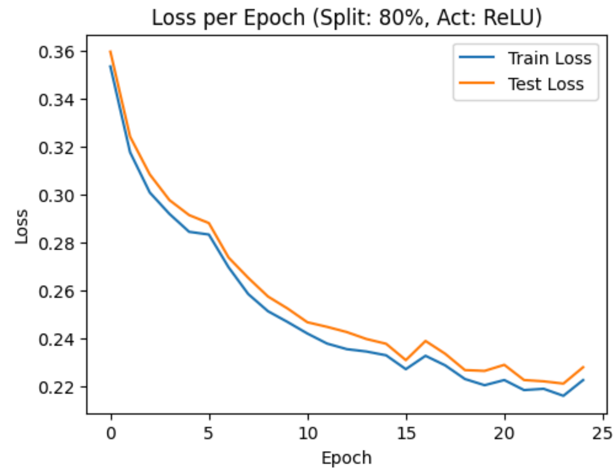


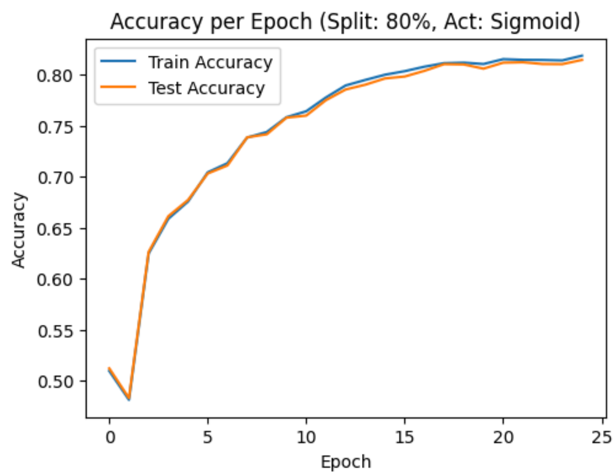


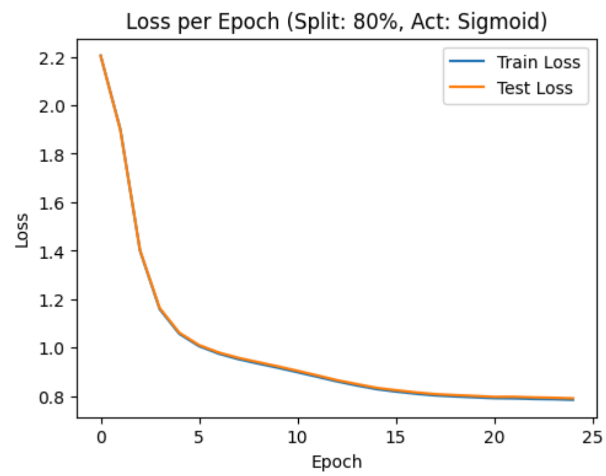


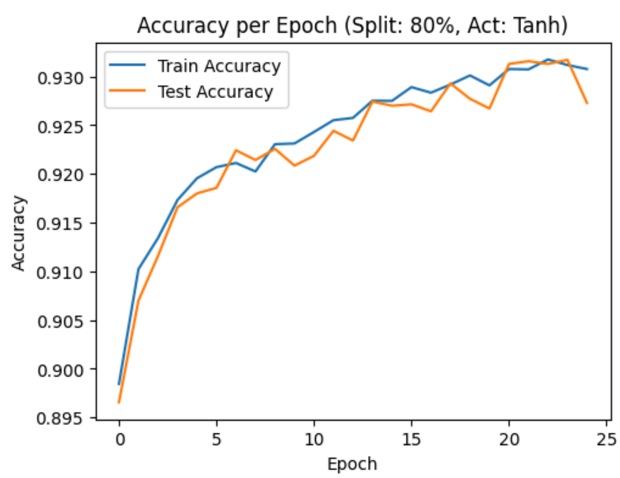
2. Train-Test split ratio as 80:20

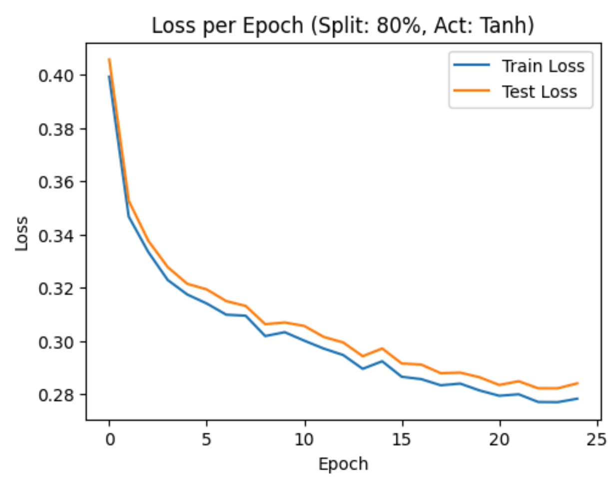




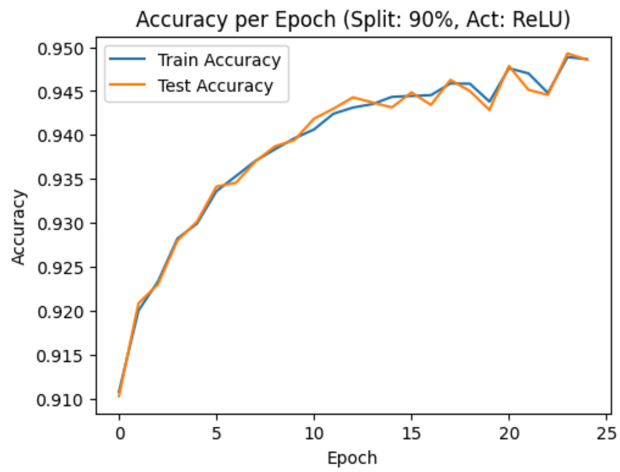


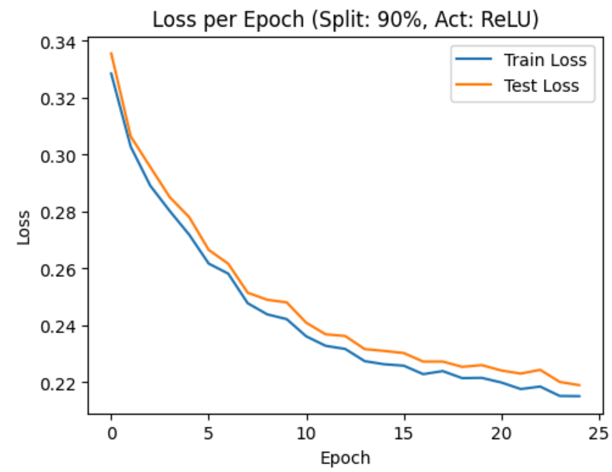


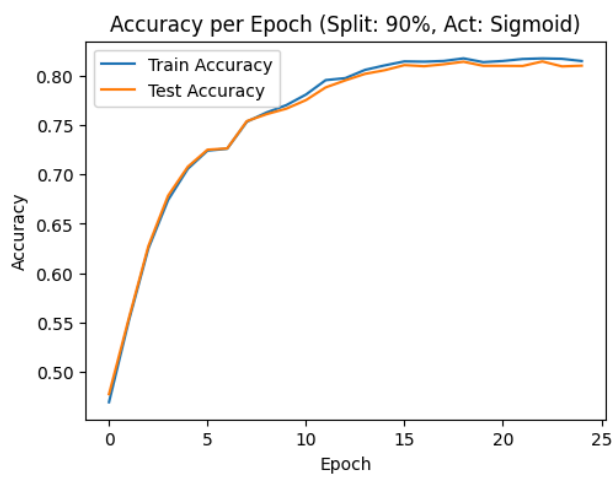


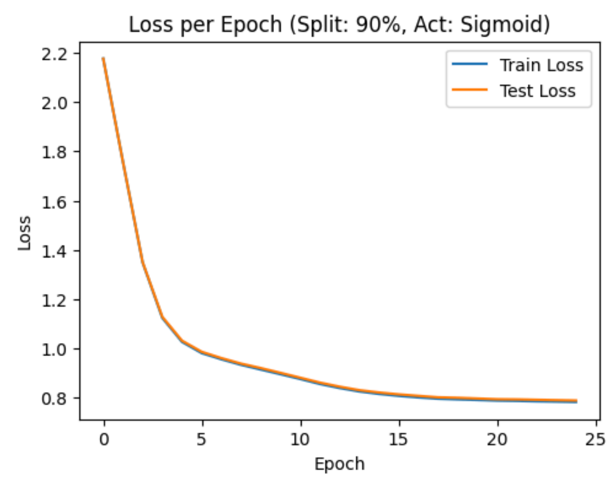


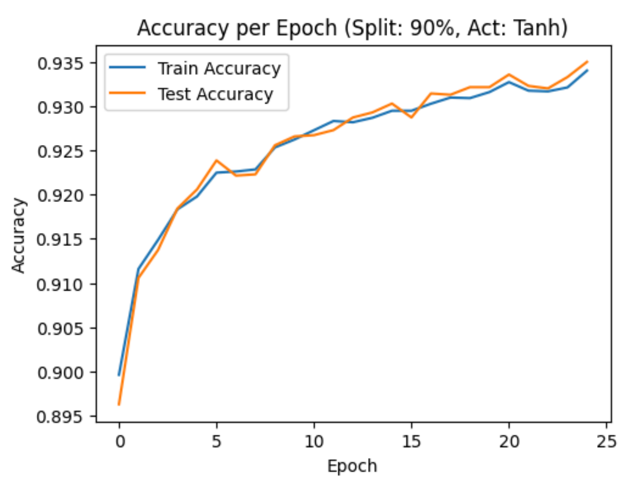
3. Train-Test split ratio as 90:10

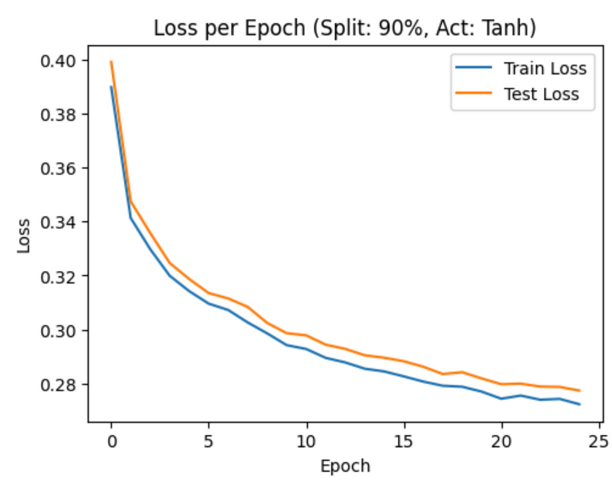












Observations

1. ReLU + He Initialization: Best performance with faster convergence and higher accuracy compared to Sigmoid and Tanh.
    - Train/Test Accuracy: ~94% at the 70% split and ~94% at the 80% split.
2. Sigmoid & Tanh: Poorer performance due to vanishing gradients, especially in deeper networks.
    - Train/Test Accuracy (Sigmoid): Peaks at around ~81% at the 70% split and ~81% at the 80% split.
    - Train/Test Accuracy (Tanh): Reaches ~93% at best (70% split), slightly lower than ReLU's performance.
3. Train-Test Splits: Performance improves slightly with an 90% train split, but activation and initialization dominate results.

References

1. https://mahendra-choudhary.medium.com/mnist-handwritten-digits-recognition-using-scikit-learn-36161768d10e
2. https://www.kaggle.com/datasets/oddrationale/mnist-in-csv
3. https://www.geeksforgeeks.org/backpropagation-in-neural-network/
4. https://www.javatpoint.com/pytorch-backpropagation-process-in-deep-neural-network
5. https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd
6. https://www.deeplearning.ai/ai-notes/initialization/index.html
7. https://machinelearningmastery.com/weight-initialization-for-deep-learning-neural-networks/