DMLPaperSummary

Keerthana Golla

October 2023

Question-1

Visualizing a CNN with CIFAR10

On running the code uploaded for question-1 , I got the fig1,2 as the plots for accuracy and loss for the respective model in tensorboard .

From fig 1-5 we can see accuracy and loss plot and also the accuracy before hyper parameter tuning is around 65% .We can also see the accuracy increased to 68% after hyperparameter tuning .Although it is not much difference but due to tuning hyperparameter with different set of values we can atleast be able to increase the accuracy to 3-4% in this case.

And from fig 6-9 we can clearly see the filters in layer 1 and also the statistics of the activations in the convolutional layers on test images.There are filters of 32 and are captured in fig 6,7 like mentioned in question we using gray-scale images. In fliters you can see what kind of edge detections are been done-though it may not be clearly put into words we can observe the differenc ein each filter few pixels are light and few are dark allowing to extract some feature using each of these filters.

Respective code can be seen under Q1 folder

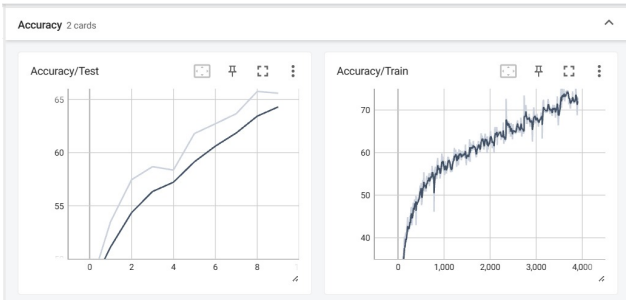


Figure 1: 1B- Accuracy plot

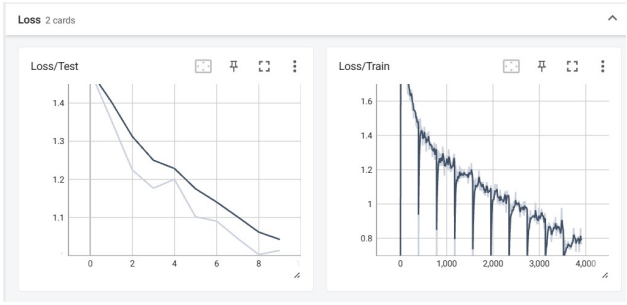
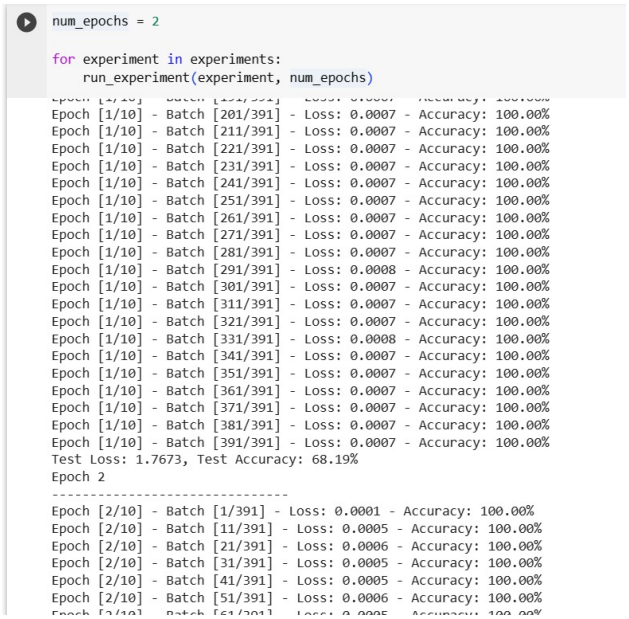


Figure 2: 1B- Loss plot

```
✓ 3m [7] Epoch [10/10] - Batch [61/391] - Loss: 0.7701 - Accuracy: 73.52%
         Epoch [10/10] - Batch [71/391] - Loss: 0.7531 - Accuracy: 72.42%
         Epoch [10/10] - Batch [81/391] - Loss: 0.7383 - Accuracy: 74.92%
         Epoch [10/10] - Batch [91/391] - Loss: 0.7689 - Accuracy: 73.75%
         Epoch [10/10] - Batch [101/391] - Loss: 0.7854 - Accuracy: 72.03%
         Epoch [10/10] - Batch [111/391] - Loss: 0.7672 - Accuracy: 73.83%
         Epoch [10/10] - Batch [121/391] - Loss: 0.7618 - Accuracy: 72.97%
         Epoch [10/10] - Batch [131/391] - Loss: 0.7343 - Accuracy: 74.53%
         Epoch [10/10] - Batch [141/391] - Loss: 0.7635 - Accuracy: 73.83%
         Epoch [10/10] - Batch [151/391] - Loss: 0.7572 - Accuracy: 74.53%
         Epoch [10/10] - Batch [161/391] - Loss: 0.7331 - Accuracy: 74.69%
         Epoch [10/10] - Batch [171/391] - Loss: 0.7694 - Accuracy: 73.91%
         Epoch [10/10] - Batch [181/391] - Loss: 0.7784 - Accuracy: 72.97%
         Epoch [10/10] - Batch [191/391] - Loss: 0.7919 - Accuracy: 71.33%
         Epoch [10/10] - Batch [201/391] - Loss: 0.7710 - Accuracy: 72.89%
         Epoch [10/10] - Batch [211/391] - Loss: 0.8182 - Accuracy: 71.80%
         Epoch [10/10] - Batch [221/391] - Loss: 0.8610 - Accuracy: 70.31%
         Epoch [10/10] - Batch [231/391] - Loss: 0.8226 - Accuracy: 71.56%
         Epoch [10/10] - Batch [241/391] - Loss: 0.7834 - Accuracy: 72.34%
         Epoch [10/10] - Batch [251/391] - Loss: 0.7907 - Accuracy: 72.81%
         Epoch [10/10] - Batch [261/391] - Loss: 0.8033 - Accuracy: 71.95%
         Epoch [10/10] - Batch [271/391] - Loss: 0.7208 - Accuracy: 74.38%
         Epoch [10/10] - Batch [281/391] - Loss: 0.7903 - Accuracy: 72.27%
         Epoch [10/10] - Batch [291/391] - Loss: 0.7930 - Accuracy: 71.56%
         Epoch [10/10] - Batch [301/391] - Loss: 0.7904 - Accuracy: 71.64%
         Epoch [10/10] - Batch [311/391] - Loss: 0.8042 - Accuracy: 71.64%
         Epoch [10/10] - Batch [321/391] - Loss: 0.8125 - Accuracy: 72.27%
         Epoch [10/10] - Batch [331/391] - Loss: 0.7958 - Accuracy: 72.89%
         Epoch [10/10] - Batch [341/391] - Loss: 0.7196 - Accuracy: 75.47%
         Epoch [10/10] - Batch [351/391] - Loss: 0.8025 - Accuracy: 72.66%
         Epoch [10/10] - Batch [361/391] - Loss: 0.7910 - Accuracy: 72.27%
         Epoch [10/10] - Batch [371/391] - Loss: 0.8600 - Accuracy: 68.75%
         Epoch [10/10] - Batch [381/391] - Loss: 0.7642 - Accuracy: 72.97%
         Epoch [10/10] - Batch [391/391] - Loss: 0.7783 - Accuracy: 72.24%
         Test Loss: 1.0133, Test Accuracy: 65.58%
```

Figure 3: 1B- Accuracy before hyperparameter tunning

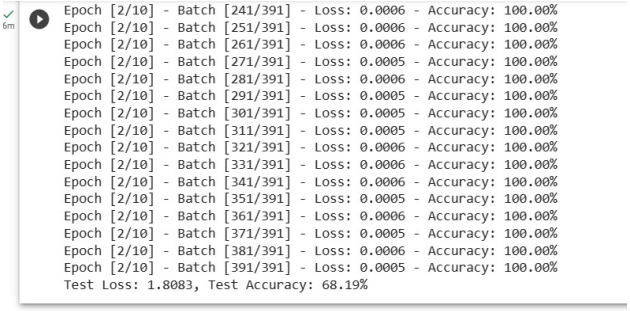


```
num_epochs = 2

for experiment in experiments:
    run_experiment(experiment, num_epochs)
```

```
Epoch [1/10] - Batch [201/391] - Loss: 0.0007 - Accuracy: 100.00%
Epoch [1/10] - Batch [211/391] - Loss: 0.0007 - Accuracy: 100.00%
Epoch [1/10] - Batch [221/391] - Loss: 0.0007 - Accuracy: 100.00%
Epoch [1/10] - Batch [231/391] - Loss: 0.0007 - Accuracy: 100.00%
Epoch [1/10] - Batch [241/391] - Loss: 0.0007 - Accuracy: 100.00%
Epoch [1/10] - Batch [251/391] - Loss: 0.0007 - Accuracy: 100.00%
Epoch [1/10] - Batch [261/391] - Loss: 0.0007 - Accuracy: 100.00%
Epoch [1/10] - Batch [271/391] - Loss: 0.0007 - Accuracy: 100.00%
Epoch [1/10] - Batch [281/391] - Loss: 0.0007 - Accuracy: 100.00%
Epoch [1/10] - Batch [291/391] - Loss: 0.0008 - Accuracy: 100.00%
Epoch [1/10] - Batch [301/391] - Loss: 0.0007 - Accuracy: 100.00%
Epoch [1/10] - Batch [311/391] - Loss: 0.0007 - Accuracy: 100.00%
Epoch [1/10] - Batch [321/391] - Loss: 0.0007 - Accuracy: 100.00%
Epoch [1/10] - Batch [331/391] - Loss: 0.0008 - Accuracy: 100.00%
Epoch [1/10] - Batch [341/391] - Loss: 0.0007 - Accuracy: 100.00%
Epoch [1/10] - Batch [351/391] - Loss: 0.0007 - Accuracy: 100.00%
Epoch [1/10] - Batch [361/391] - Loss: 0.0007 - Accuracy: 100.00%
Epoch [1/10] - Batch [371/391] - Loss: 0.0007 - Accuracy: 100.00%
Epoch [1/10] - Batch [381/391] - Loss: 0.0007 - Accuracy: 100.00%
Epoch [1/10] - Batch [391/391] - Loss: 0.0007 - Accuracy: 100.00%
Test Loss: 1.7673, Test Accuracy: 68.19%
Epoch 2
------------------------------
Epoch [2/10] - Batch [1/391] - Loss: 0.0001 - Accuracy: 100.00%
Epoch [2/10] - Batch [11/391] - Loss: 0.0005 - Accuracy: 100.00%
Epoch [2/10] - Batch [21/391] - Loss: 0.0006 - Accuracy: 100.00%
Epoch [2/10] - Batch [31/391] - Loss: 0.0005 - Accuracy: 100.00%
Epoch [2/10] - Batch [41/391] - Loss: 0.0005 - Accuracy: 100.00%
Epoch [2/10] - Batch [51/391] - Loss: 0.0006 - Accuracy: 100.00%
```

Figure 4: 1B- Accuracy after Hyperparameter tuning epoch 1



```
Epoch [2/10] - Batch [241/391] - Loss: 0.0006 - Accuracy: 100.00%
Epoch [2/10] - Batch [251/391] - Loss: 0.0006 - Accuracy: 100.00%
Epoch [2/10] - Batch [261/391] - Loss: 0.0006 - Accuracy: 100.00%
Epoch [2/10] - Batch [271/391] - Loss: 0.0005 - Accuracy: 100.00%
Epoch [2/10] - Batch [281/391] - Loss: 0.0006 - Accuracy: 100.00%
Epoch [2/10] - Batch [291/391] - Loss: 0.0005 - Accuracy: 100.00%
Epoch [2/10] - Batch [301/391] - Loss: 0.0005 - Accuracy: 100.00%
Epoch [2/10] - Batch [311/391] - Loss: 0.0005 - Accuracy: 100.00%
Epoch [2/10] - Batch [321/391] - Loss: 0.0006 - Accuracy: 100.00%
Epoch [2/10] - Batch [331/391] - Loss: 0.0006 - Accuracy: 100.00%
Epoch [2/10] - Batch [341/391] - Loss: 0.0006 - Accuracy: 100.00%
Epoch [2/10] - Batch [351/391] - Loss: 0.0005 - Accuracy: 100.00%
Epoch [2/10] - Batch [361/391] - Loss: 0.0006 - Accuracy: 100.00%
Epoch [2/10] - Batch [371/391] - Loss: 0.0005 - Accuracy: 100.00%
Epoch [2/10] - Batch [381/391] - Loss: 0.0006 - Accuracy: 100.00%
Epoch [2/10] - Batch [391/391] - Loss: 0.0005 - Accuracy: 100.00%
Test Loss: 1.8083, Test Accuracy: 68.19%
```

Figure 5: 1B- Accuracy after Hyperparameter tuning epoch 2

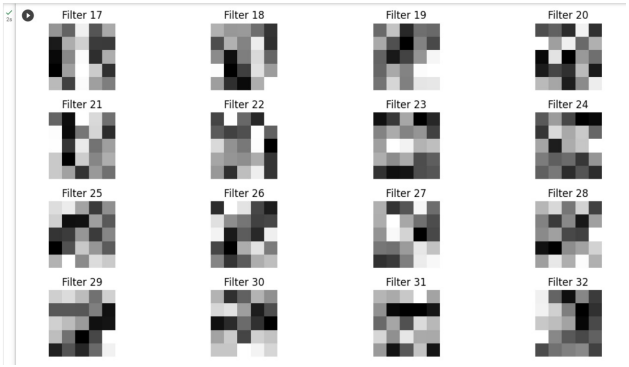


Figure 6: 1C- Filters from 17 to 32

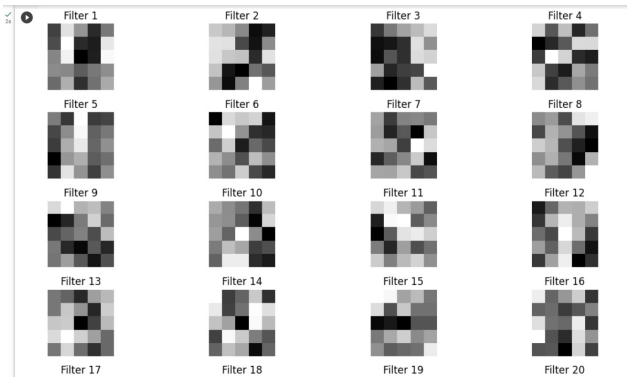


Figure 7: 1C- Filters from 1 to 16

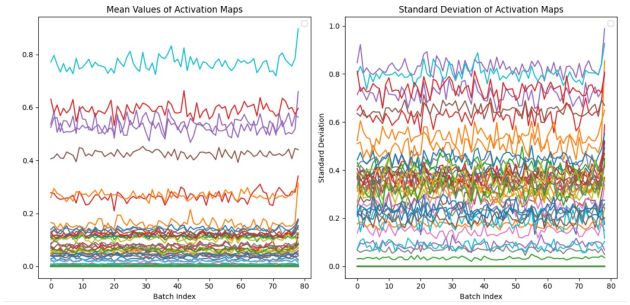


Figure 8: 1C - statistics - mean and standard deviation

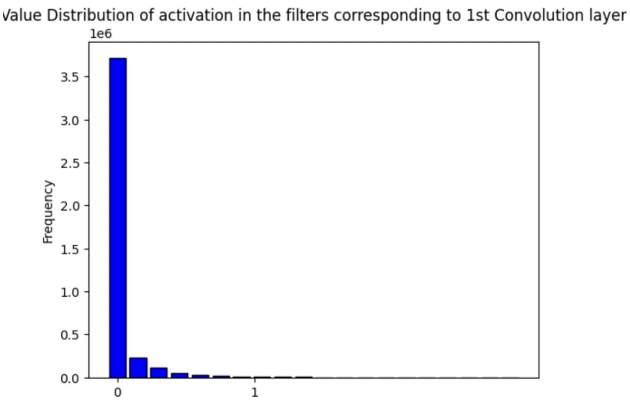


Figure 9: 1C-histogram

Question -2

Paper Summary: Visualizing and Understanding Convolutional Neural Networks

This paper introduces novel visualization techniques for understanding Convolutional Neural Networks (CNNs) commonly used in computer vision. Using a deconvolutional network, the authors project feature activations back onto input images, revealing what specific features the CNN recognizes at each layer, from edges and colors in early layers to entire objects in later layers. These visualizations offer valuable insights into the CNN's functioning and the nature of features it detects.The paper's visualization technique involves finding the optimal stimulus for each unit in the network by performing gradient descent in image space to maximize the unit's activation. The resulting visualizations show the features that each unit is sensitive to, and how these features change as we move up the layers of the network.

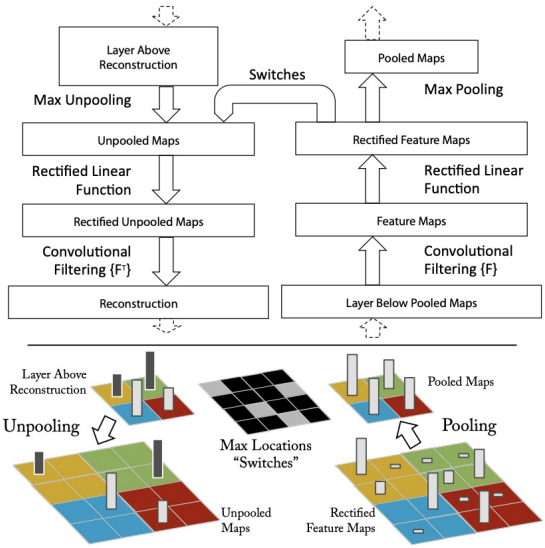
The paper also introduces several technical innovations that are important for understanding the operation of CNNs. One of these is the unpooling operation, which is used to obtain an approximate inverse of the max pooling operation. The max pooling operation is non-invertible, but the paper shows that an approximate inverse can be obtained by recording the locations of the maxima within each pooling region in a set of switch variables. In the deconvnet, the unpooling operation uses these switches to place the reconstructions from the layer above into appropriate locations, preserving the structure of the stimulus.

The authors apply these visualizations to a CNN trained on a vast dataset of 1.3 million images. Notably, by systematically occluding parts of images, they demonstrate that the model localizes and recognizes objects based on key features, rather than relying solely on scene context. The CNN even shows the ability to identify corresponding features across different instances of objects, such as eyes or noses in various dog images.

Through a series of experiments, the paper highlights the excellent transferability of features learned by the CNN on a large dataset to smaller datasets. Merely retraining the final layer yields state-of-the-art performance on benchmark datasets like Caltech-256.

An analysis of the CNN's architecture reveals that overall depth is crucial, as removing multiple layers significantly impairs performance. Deeper layers learn more discriminative features for the given task. Even on challenging datasets with complex scenes and multiple objects, the proposed approach achieves competitive accuracy compared to leading methods without the need for extensive tuning.

In summary, this paper presents innovative visualization techniques that shed light on CNN operations and learned features at various layers. These insights are invaluable for diagnosing and enhancing model architectures. Moreover, the ability to transfer learned features from large datasets results in superior performance on related, smaller datasets with minimal retraining. The work significantly advances the comprehension and design of CNNs, which are fundamental in the field of computer vision.



Additional work : Bit more detail on Deconvolution explained in class

The above picture gives a high level understanding of deconv. (picutre is also mentioned in paper and also explained by professor in class) like mentioned in summary 2nd paragraph: Pooling operations, such as max pooling and average pooling, are essential for reducing computational complexity and enforcing translation invariance in Convolutional Neural Networks (CNNs). However, these operations are non-invertible, which means that you cannot perfectly reverse them to reconstruct the original input. During pooling, information is lost as the operation selects the maximum or averages values within a pooling window, reducing spatial dimensions. To address this non-invertibility, techniques like max unpooling are used, which rely on switch variables to place reconstructions from the layer above into the appropriate positions. This helps preserve the structure of the stimulus. Additionally, filter flipping is essential when convolving feature maps with learned filters. When reconstructing an ideal image for a given filter, the filter is effectively "flipped" to ensure that the reconstruction accurately reflects the learned features of the filter. This flipping aligns the filter's weights with the spatial structure of the feature map, allowing it to detect the presence of the learned feature across different regions of the input.

In the deconvolutional network, the unpooling operation is achieved by using switch variables that record the locations of maxima in the pooling regions during the convolutional network's forward pass. This process helps place reconstructed features from higher layers into the appropriate positions, preserving the stimulus's structure. Rectification involves applying ReLU non-linearities to ensure that feature maps and reconstructions remain positive. The filtering step uses transposed versions of learned filters, applied to rectified feature maps, but with a vertical and horizontal flip. By projecting down from higher layers using switch settings generated during the max pooling process, the deconvolutional network reconstructs input image sections, highlighting discriminative regions. These projections do not involve a generative process, offering insights into discriminative features within the input data.

Extra Credit - Activation Maximization

: The code uploaded for Q1, iteratively adjusts the input image to maximize the activation of a specific filter in the chosen layer of the model. This is the essence of activation maximization, where we can modify the input image to make it produce the maximum response for the selected filter in the network. Picture is seen below in Fig 10

```
# Convert the optimized image to a NumPy array
optimized_image = input_image.squeeze().detach().cpu().numpy()
optimized_image = np.transpose(optimized_image, (1, 2, 0))
optimized_image = np.clip(optimized_image, 0, 1)

# Display the optimized image
plt.imshow(optimized_image)
plt.axis('off')
plt.title(f'Activation Maximization for Filter {filter_index}')
plt.show()
```

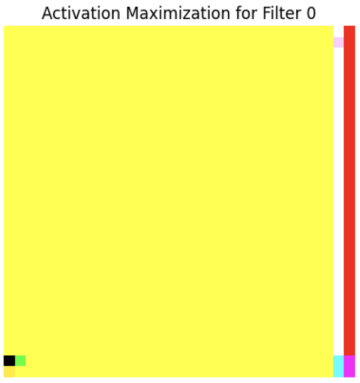


Figure 10: 2-Activation Maximization

Question -3

Respective code can be seen under Q3/RNN folder RNN with no.of hidden layer nodes 128 , in fig 12,13 we can see the accuracy and loss graphs for this model and the accuracy to be 38.69% in fig 14
The fig 11 is the image - one test image

Figure 11: 3A- Sample image

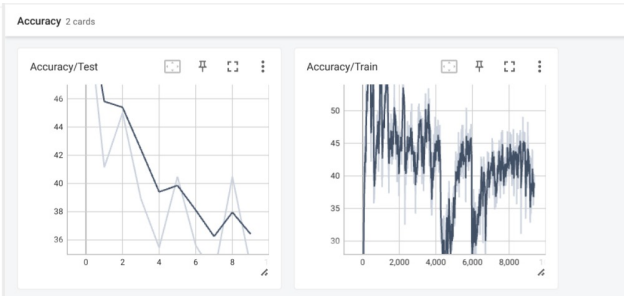


Figure 12: 3A- RNN-Accuracy graphs when No.of nodes in hidden layer=128

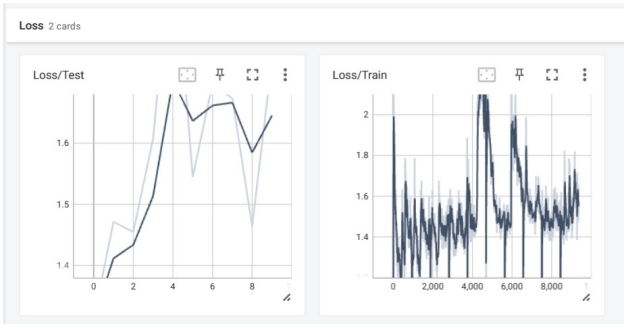


Figure 13: 3A- RNN-Loss graphs when No.of nodes in hidden layer=128

```
[19] Epoch [10/10] - Batch [561/938] - Loss: 1.6158 - Accuracy: 37.34%
     Epoch [10/10] - Batch [571/938] - Loss: 1.6355 - Accuracy: 39.84%
     Epoch [10/10] - Batch [581/938] - Loss: 1.5258 - Accuracy: 38.91%
     Epoch [10/10] - Batch [591/938] - Loss: 1.5906 - Accuracy: 38.75%
     Epoch [10/10] - Batch [601/938] - Loss: 1.6199 - Accuracy: 36.09%
     Epoch [10/10] - Batch [611/938] - Loss: 1.5991 - Accuracy: 40.78%
     Epoch [10/10] - Batch [621/938] - Loss: 1.6550 - Accuracy: 36.41%
     Epoch [10/10] - Batch [631/938] - Loss: 1.5344 - Accuracy: 37.97%
     Epoch [10/10] - Batch [641/938] - Loss: 1.7698 - Accuracy: 34.69%
     Epoch [10/10] - Batch [651/938] - Loss: 1.9157 - Accuracy: 32.03%
     Epoch [10/10] - Batch [661/938] - Loss: 1.8010 - Accuracy: 34.53%
     Epoch [10/10] - Batch [671/938] - Loss: 1.6964 - Accuracy: 36.56%
     Epoch [10/10] - Batch [681/938] - Loss: 1.5965 - Accuracy: 39.38%
     Epoch [10/10] - Batch [691/938] - Loss: 1.6723 - Accuracy: 35.16%
     Epoch [10/10] - Batch [701/938] - Loss: 1.6256 - Accuracy: 39.69%
     Epoch [10/10] - Batch [711/938] - Loss: 1.4951 - Accuracy: 43.28%
     Epoch [10/10] - Batch [721/938] - Loss: 1.6698 - Accuracy: 37.81%
     Epoch [10/10] - Batch [731/938] - Loss: 1.5805 - Accuracy: 38.59%
     Epoch [10/10] - Batch [741/938] - Loss: 1.5228 - Accuracy: 36.25%
     Epoch [10/10] - Batch [751/938] - Loss: 1.5767 - Accuracy: 36.56%
     Epoch [10/10] - Batch [761/938] - Loss: 1.5429 - Accuracy: 38.44%
     Epoch [10/10] - Batch [771/938] - Loss: 1.5515 - Accuracy: 39.06%
     Epoch [10/10] - Batch [781/938] - Loss: 1.5322 - Accuracy: 37.81%
     Epoch [10/10] - Batch [791/938] - Loss: 1.6161 - Accuracy: 40.00%
     Epoch [10/10] - Batch [801/938] - Loss: 1.6471 - Accuracy: 36.72%
     Epoch [10/10] - Batch [811/938] - Loss: 1.4598 - Accuracy: 42.19%
     Epoch [10/10] - Batch [821/938] - Loss: 1.5345 - Accuracy: 39.38%
     Epoch [10/10] - Batch [831/938] - Loss: 1.5757 - Accuracy: 39.22%
     Epoch [10/10] - Batch [841/938] - Loss: 1.9006 - Accuracy: 33.44%
     Epoch [10/10] - Batch [851/938] - Loss: 1.8035 - Accuracy: 33.12%
     Epoch [10/10] - Batch [861/938] - Loss: 1.8054 - Accuracy: 33.28%
     Epoch [10/10] - Batch [871/938] - Loss: 1.6249 - Accuracy: 34.53%
     Epoch [10/10] - Batch [881/938] - Loss: 1.6344 - Accuracy: 36.88%
     Epoch [10/10] - Batch [891/938] - Loss: 1.4987 - Accuracy: 39.38%
     Epoch [10/10] - Batch [901/938] - Loss: 1.4416 - Accuracy: 45.00%
     Epoch [10/10] - Batch [911/938] - Loss: 1.6659 - Accuracy: 37.03%
     Epoch [10/10] - Batch [921/938] - Loss: 1.6094 - Accuracy: 35.78%
     Epoch [10/10] - Batch [931/938] - Loss: 1.5243 - Accuracy: 41.56%
     Test Loss: 1.5545, Test Accuracy: 38.69%
```

Figure 14: 3A- RNN-Accuracy Percentage when No.of nodes in hidden layer=128

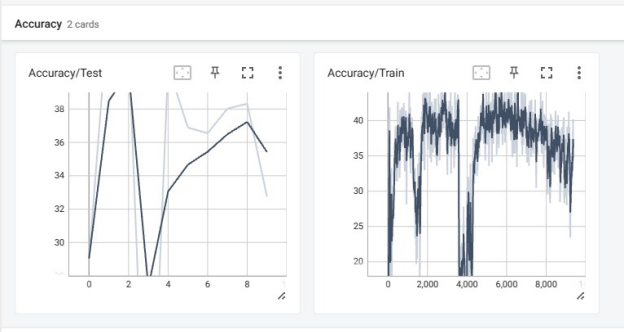


Figure 15: 3A- RNN-Accuracy graphs when No.of nodes in hidden layer=256

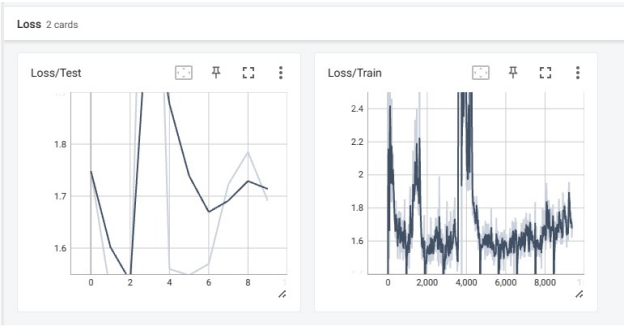


Figure 16: 3A- RNN-Loss graphs when No.of nodes in hidden layer=256

EXPERIMENTING with the mentioned parameters : I have changed the number of hidden layer nodes to 256 and maintain all the remaining parameters same as above and tried to check the accuracy ,we can see the accuracy and loss graphs for this model in fig 15,16 and the accuracy to be 32.75% in fig 17

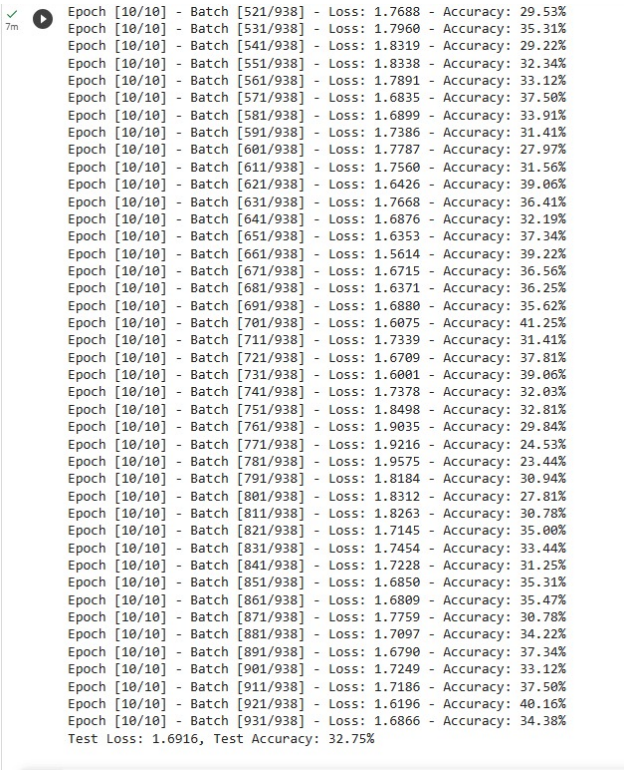


Figure 17: 3A- RNN-Accuracy Percentage when No.of nodes in hidden layer=256

EXPERIMENTING with the mentioned parameters : This time I have changed the learning rate to 0.001 , I have also changed the optimizer to SGD and number of iterations to 12. And finally changed the number of hidden layer nodes to 256 .we can see the accuracy and loss graphs for this model in fig 18,19 and the accuracy to be 82.40% in fig 20

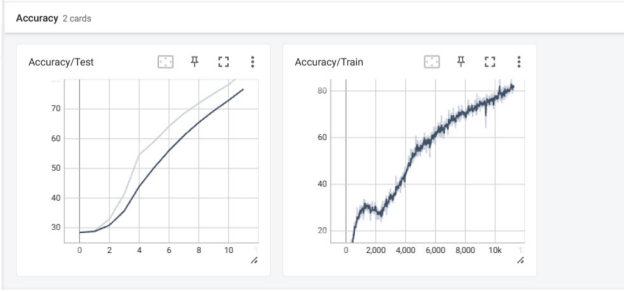


Figure 18: 3A- RNN with changed parameters -Accuracy Graphs

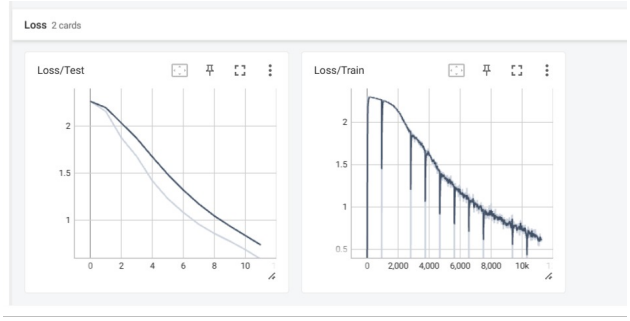


Figure 19: 3A- RNN with changed parameters - Loss Graphs

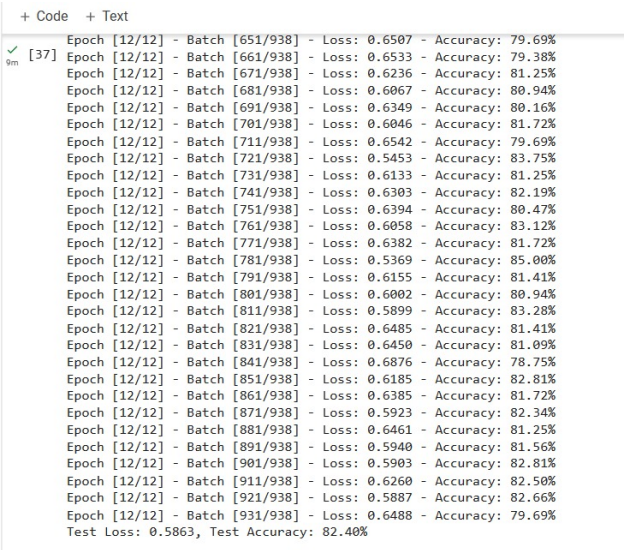


Figure 20: 3A- RNN with changed parameters -Accuracy Percentage

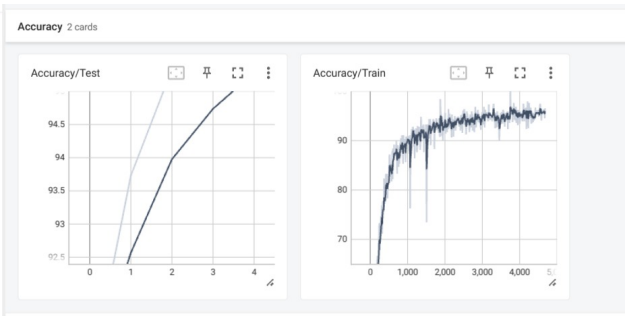


Figure 21: 3A- RNN with changed parameters -Accuracy Graphs

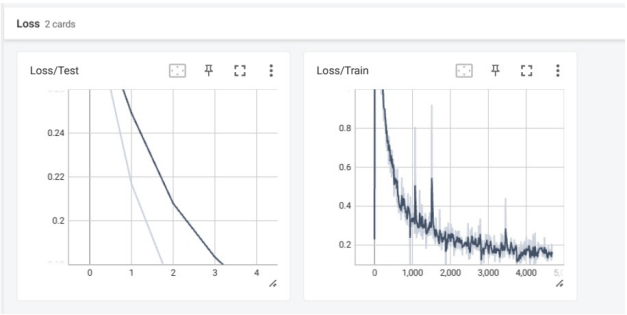


Figure 22: 3A- RNN with changed parameters -Loss Graphs

EXPERIMENTING with the mentioned parameters : This time I have changed the learning rate to 0.001 , I have also changed the optimizer to RMSProp and number of iterations to 5. And finally changed the number of hidden layer nodes to 128 .we can see the accuracy and loss graphs for this model in fig 21,22 and the accuracy to be 95.99% in fig 23

Clearly the highest accuracy is due to less no.of iterations and also optimizer RMSProp may have a significant effect on accuracy percentage - we can see a significant change in accuarcy percentage in the previous experiment too where the optimizer is SGD. Overall, change in these 4 parameters have an effect on accuracy percentage and loss(error) of the model.Doing hyperparameter tunning may increase the accuracy but may not be a significant increase like we saw in question -1 of this assignment.

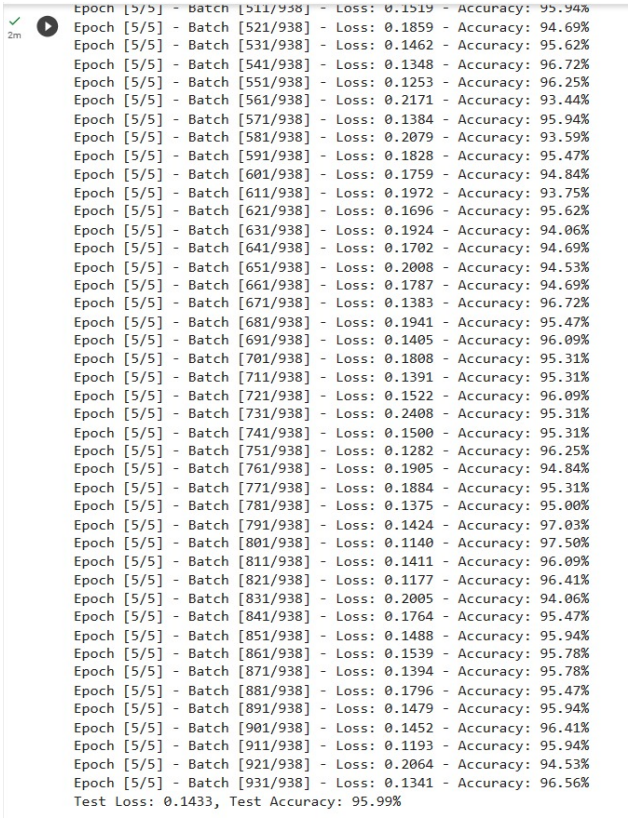


Figure 23: 3A- RNN with changed parameters -Accuracy Percentage

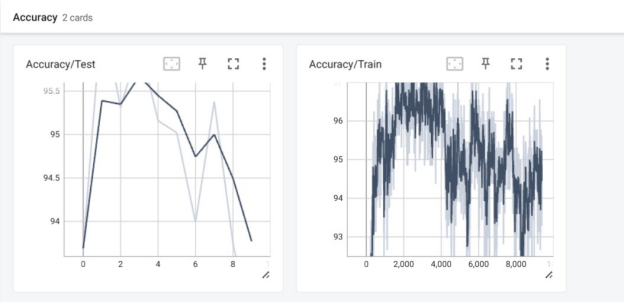


Figure 24: 3B-LSTM-Accuracy graphs when No.of nodes in hidden layer=128

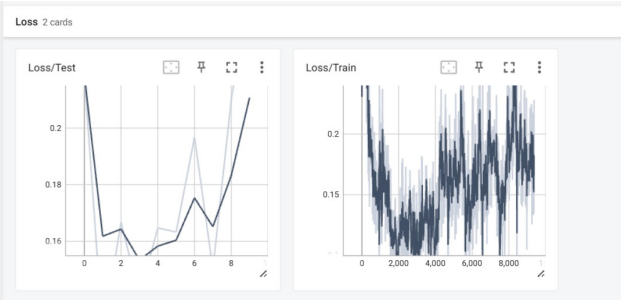


Figure 25: 3B-LSTM-Loss graphs when No.of nodes in hidden layer=128

LSTM : Respective code can be seen under Q3/LSTM folder LSTM with no.of hidden layer nodes 128 , in fig 24,25 we can see the accuracy and loss graphs for this model and the accuracy to be 92.69% in fig 26 We can compare this with the accuracy percentage of RNN. clearly almost 55% change in accuracy is seen with same hyperparameters (same optimizer , same no of epochs and learning rate)

LSTM with no.of hidden layer nodes 256 , in fig 27,28 we can see the accuracy and loss graphs for this model and the accuracy to be 92.04% in fig 29

```
Epoch [10/10] - Batch [511/938] - Loss: 0.1672 - Accuracy: 95.78%
Epoch [10/10] - Batch [521/938] - Loss: 0.2298 - Accuracy: 92.19%
Epoch [10/10] - Batch [531/938] - Loss: 0.1564 - Accuracy: 96.25%
Epoch [10/10] - Batch [541/938] - Loss: 0.1047 - Accuracy: 96.41%
Epoch [10/10] - Batch [551/938] - Loss: 0.1903 - Accuracy: 94.38%
Epoch [10/10] - Batch [561/938] - Loss: 0.1537 - Accuracy: 95.47%
Epoch [10/10] - Batch [571/938] - Loss: 0.1630 - Accuracy: 94.53%
Epoch [10/10] - Batch [581/938] - Loss: 0.1834 - Accuracy: 94.53%
Epoch [10/10] - Batch [591/938] - Loss: 0.1674 - Accuracy: 95.62%
Epoch [10/10] - Batch [601/938] - Loss: 0.1489 - Accuracy: 94.69%
Epoch [10/10] - Batch [611/938] - Loss: 0.1712 - Accuracy: 94.38%
Epoch [10/10] - Batch [621/938] - Loss: 0.1723 - Accuracy: 95.78%
Epoch [10/10] - Batch [631/938] - Loss: 0.1935 - Accuracy: 94.22%
Epoch [10/10] - Batch [641/938] - Loss: 0.1775 - Accuracy: 94.53%
Epoch [10/10] - Batch [651/938] - Loss: 0.1499 - Accuracy: 95.00%
Epoch [10/10] - Batch [661/938] - Loss: 0.1703 - Accuracy: 95.16%
Epoch [10/10] - Batch [671/938] - Loss: 0.1651 - Accuracy: 95.16%
Epoch [10/10] - Batch [681/938] - Loss: 0.1523 - Accuracy: 95.78%
Epoch [10/10] - Batch [691/938] - Loss: 0.1848 - Accuracy: 94.22%
Epoch [10/10] - Batch [701/938] - Loss: 0.1756 - Accuracy: 95.31%
Epoch [10/10] - Batch [711/938] - Loss: 0.2216 - Accuracy: 92.81%
Epoch [10/10] - Batch [721/938] - Loss: 0.1712 - Accuracy: 95.00%
Epoch [10/10] - Batch [731/938] - Loss: 0.1601 - Accuracy: 94.69%
Epoch [10/10] - Batch [741/938] - Loss: 0.1686 - Accuracy: 95.16%
Epoch [10/10] - Batch [751/938] - Loss: 0.1692 - Accuracy: 95.00%
Epoch [10/10] - Batch [761/938] - Loss: 0.1895 - Accuracy: 94.38%
Epoch [10/10] - Batch [771/938] - Loss: 0.2118 - Accuracy: 93.12%
Epoch [10/10] - Batch [781/938] - Loss: 0.1800 - Accuracy: 93.91%
Epoch [10/10] - Batch [791/938] - Loss: 0.2247 - Accuracy: 93.59%
Epoch [10/10] - Batch [801/938] - Loss: 0.1938 - Accuracy: 94.22%
Epoch [10/10] - Batch [811/938] - Loss: 0.1766 - Accuracy: 95.16%
Epoch [10/10] - Batch [821/938] - Loss: 0.1255 - Accuracy: 96.56%
Epoch [10/10] - Batch [831/938] - Loss: 0.2042 - Accuracy: 94.69%
Epoch [10/10] - Batch [841/938] - Loss: 0.1754 - Accuracy: 94.53%
Epoch [10/10] - Batch [851/938] - Loss: 0.2179 - Accuracy: 92.66%
Epoch [10/10] - Batch [861/938] - Loss: 0.1837 - Accuracy: 94.22%
Epoch [10/10] - Batch [871/938] - Loss: 0.2144 - Accuracy: 93.12%
Epoch [10/10] - Batch [881/938] - Loss: 0.1812 - Accuracy: 94.84%
Epoch [10/10] - Batch [891/938] - Loss: 0.1335 - Accuracy: 95.78%
Epoch [10/10] - Batch [901/938] - Loss: 0.1336 - Accuracy: 95.16%
Epoch [10/10] - Batch [911/938] - Loss: 0.1761 - Accuracy: 95.00%
Epoch [10/10] - Batch [921/938] - Loss: 0.1350 - Accuracy: 95.62%
Epoch [10/10] - Batch [931/938] - Loss: 0.2282 - Accuracy: 93.28%
Test Loss: 0.2517, Test Accuracy: 92.69%
```

Figure 26: 3B-LSTM-Accuracy percentage when No.of nodes in hidden layer=128

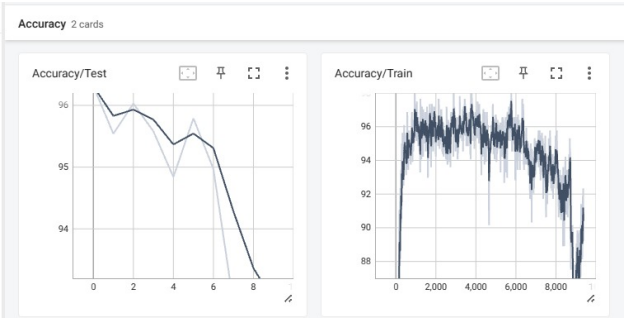


Figure 27: 3B-LSTM-Accuracy graphs when No.of nodes in hidden layer=256

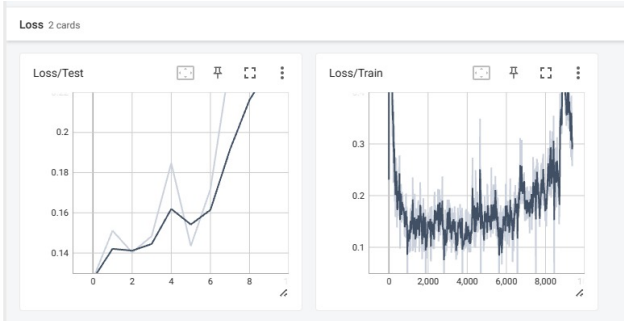


Figure 28: 3B-LSTM-Loss graphs when No.of nodes in hidden layer=256

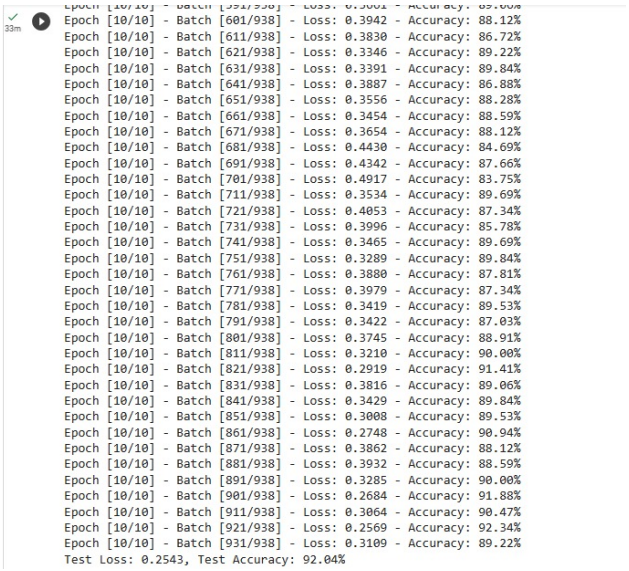


Figure 29: 3B-LSTM-Accuracy percentage when No.of nodes in hidden layer=256

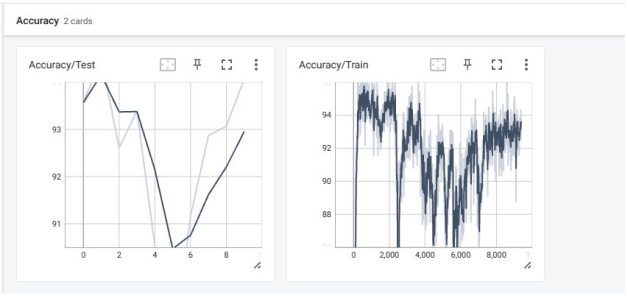


Figure 30: 3B-GRU-Accuracy graphs when No.of nodes in hidden layer=128

GRU: Respective code can be seen under Q3/GRU folder GRU with no.of hidden layer nodes 128 , in fig 30,31 we can see the accuracy and loss graphs for this model and the accuracy to be 94.07% in fig 32 We can compare this with the accuracy percentage of RNN. clearly almost 55% change in accuracy is seen with same hyperparameters (same optimizer , same no of epochs and learning rate)

GRU with no.of hidden layer nodes 128 , in fig 33,34 we can see the accuracy and loss graphs for this model and the accuracy to be 92.25% in fig 35

I have tried LSTM , GRU with the number of nodes in hidden layer 128 and observed the below changes Accuracy of LSTM with number of hidden layer 256 is 92.69%, Accuracy of GRU with number of hidden layer 128 is 94.07%. Where as RNN cant be compared to LSTM and GRU , its accuracy is really low like seen previously. clearly we can see with the same optimizer and learning rates GRU model outperformed LSTM model by 2% I have tried increasing the number of nodes in hidden layer to 256 and observed the below changes Accuracy of LSTM with number of hidden layer 256 is 92.04%, Accuracy of GRU with number of hidden layer 256 is 92.25%, with ths change we can't see much difference but still GRU gave a better accuracy than LSTM- although it is really small change . In conclusion ,although LSTM has more complex control over memory and is better at handling longer-term dependencies, whereas GRU is computationally less intensive and may perform better with limited data.

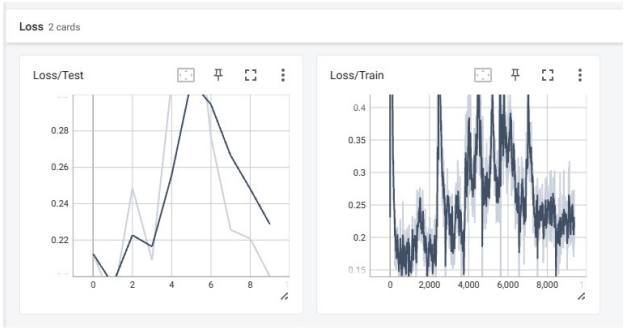


Figure 31: 3B-GRU-Loss graphs when No.of nodes in hidden layer=128

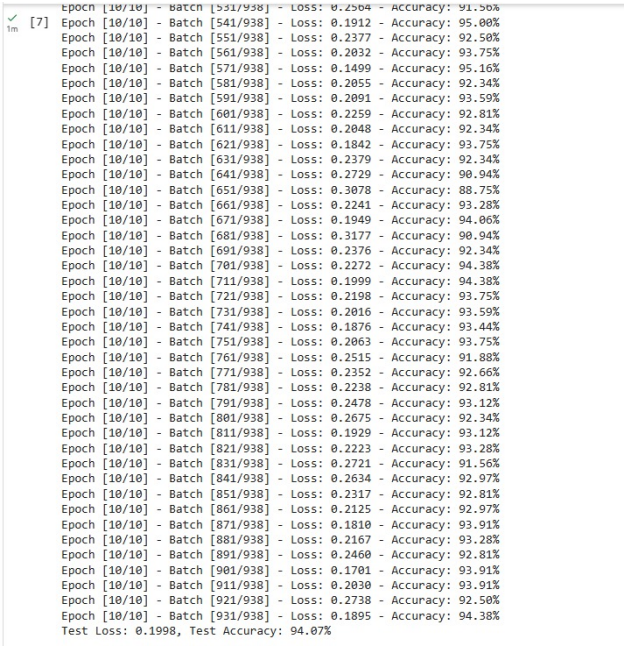


Figure 32: 3B-GRU-Accuracy percentage when No.of nodes in hidden layer=128

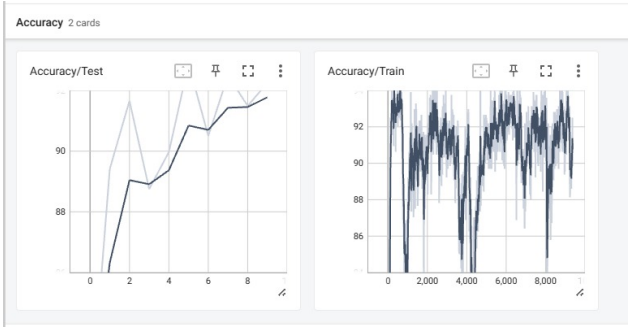


Figure 33: 3B-GRU-Accuracy graphs when No.of nodes in hidden layer=256

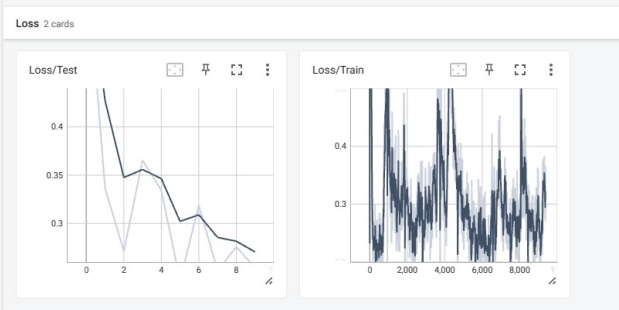


Figure 34: 3B-GRU-Loss graphs when No.of nodes in hidden layer=256

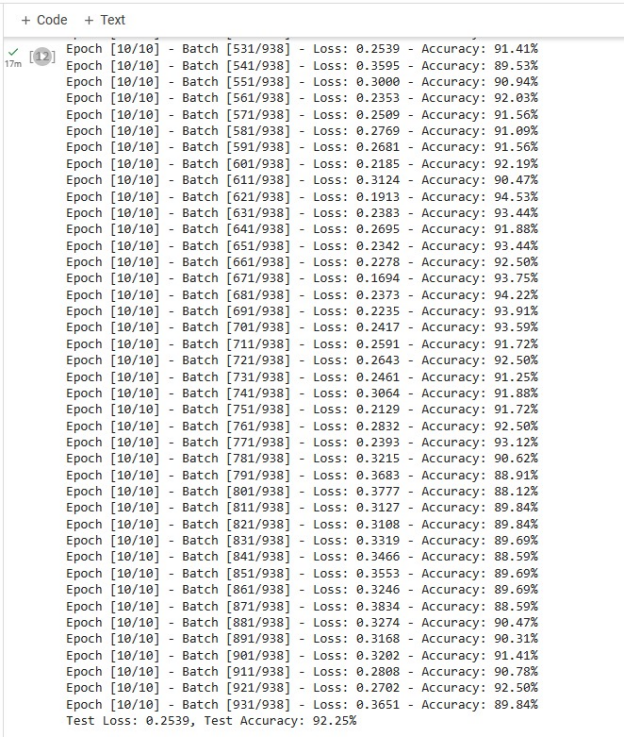


Figure 35: 3B-GRU-Accuracy percentage when No.of nodes in hidden layer=128

I got higher accuracy in assignment 1 when compare to rnn in this assignment , even trying with different activation function and doing hyperparameter tunning the accuracy is small compared to CNN in assignment -1 based on this we can say,
Similarities to Assignment 1: The CNN in Assignment 1 likely performed well on MNIST due to its ability to learn local features such as edges and textures, which are important in image recognition. Both models might have used a softmax layer for multiclass classification.
Differences from Assignment 1: The RNN's architecture differs significantly from a CNN. While CNNs have convolutional and pooling layers for feature extraction, RNNs have recurrent layers to process sequences. RNNs are more suited for sequential data like time series or natural language, and using them for image data is unconventional. This might explain the lower accuracy compared to a CNN on MNIST.
In summary, CNNs and RNNs have different architectures and are suitable for different types of data. The choice of architecture should align with the nature of your data and the specific task you're trying to solve. For image classification tasks like MNIST, CNNs are typically the preferred choice, while RNNs are better suited for sequential data analysis.