Robert Heeter
ELEC 576 Introduction to Deep Learning
25 October 2023

# Problem Set #2

## I. Visualizing a CNN with CIFAR10

***Designing CNN***

The CIFAR10 dataset was used for training a convolutional neural network built with PyTorch. The CIFAR10 dataset is composed of RGB 32 x 32 pixel natural images in 10 classes: *airplane, automobile, bird, cat, deer, dog, frog, horse, ship,* and *truck*. These images were converted to grayscale for this project.

The LeNet5 architecture for image classification was designed with the following structure in PyTorch using an Adam or SGD optimizer and CrossEntropyLoss loss criterion:

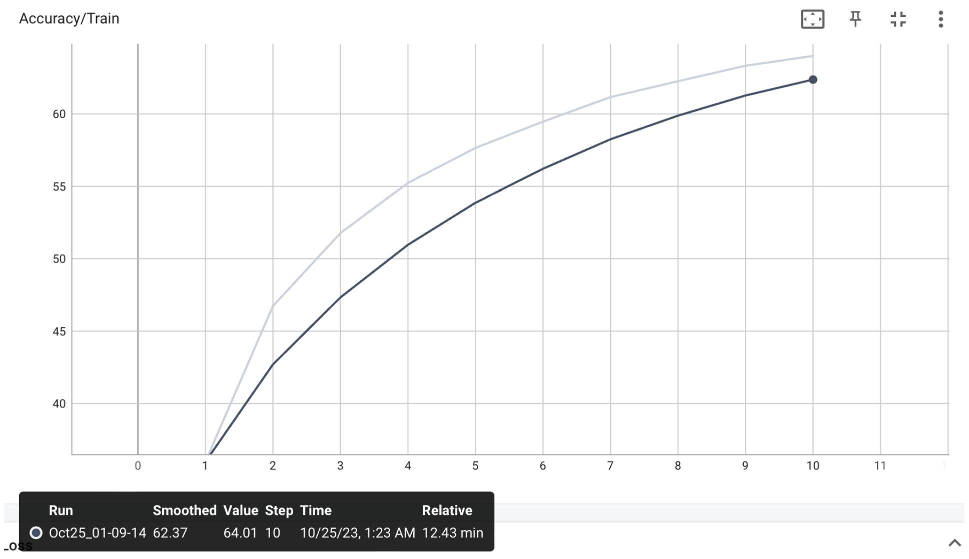
- Convolutional layer with kernel 5 x 5 and 32 filter maps followed by ReLU
- Max Pooling layer subsampling by 2
- Convolutional layer with kernel 5 x 5 and 64 filter maps followed by ReLU
- Max Pooling layer subsampling by 2
- Fully connected layer that has input 8*8*64 = 4096 and output 1024
- Fully connected layer that has input 1024 and output 10 (for the 10 classes)
- Softmax layer (softmax regression + softmax nonlinearity)

***Training & Testing CNN***

TensorBoard was used to log training and testing metrics for 10 epochs. A learning rate of 1e-4 was found to provide the best test accuracy, though 1e-3 produced similar results. Both Adam and SGD optimizers were tried; the Adam optimizer produced a higher test accuracy over 10 epochs. The training and test accuracies and training loss for both optimizers are shown below with a learning rate of 1e-4.

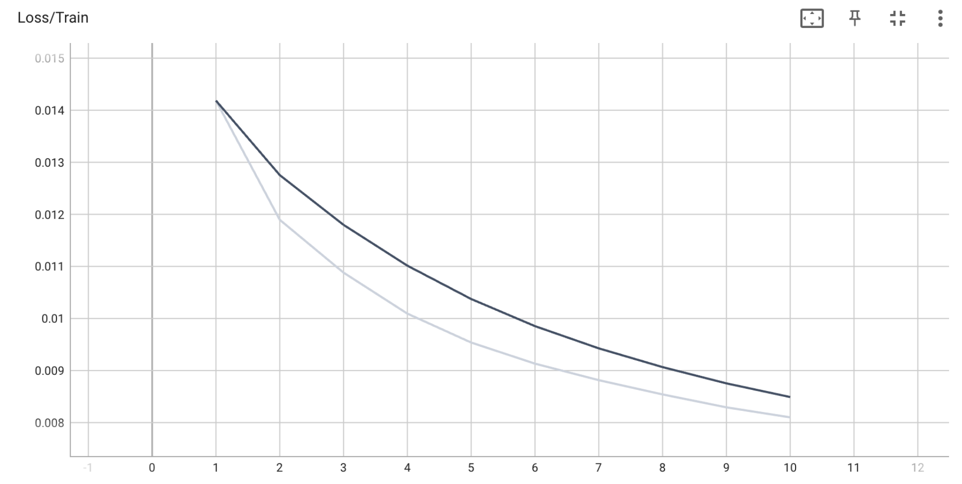
With an Adam optimizer, the test accuracy after 10 epochs was 61.3%.

Training accuracy vs. epoch:

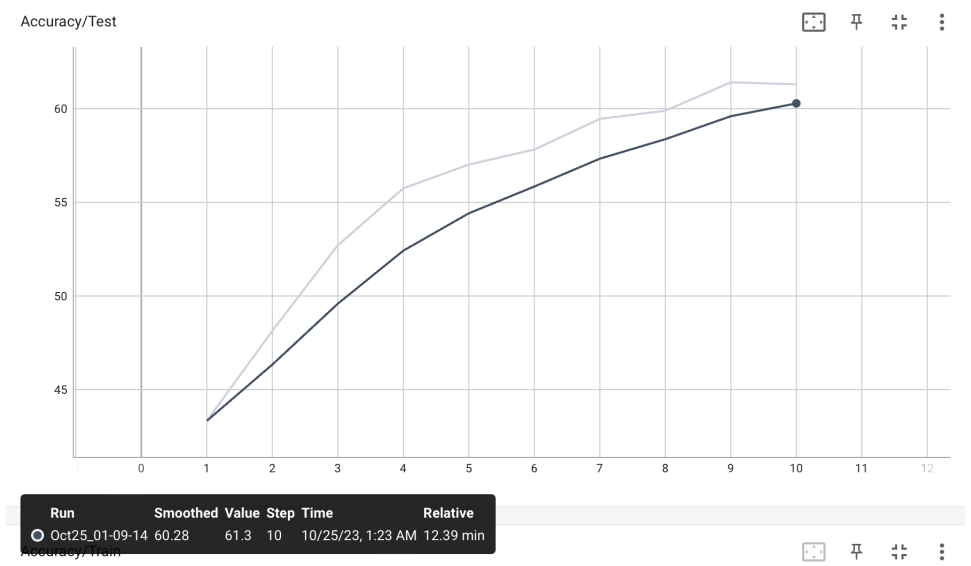


```
Run             Smoothed  Value  Step  Time                 Relative
Oct25_01-09-14  62.37     64.01  10    10/25/23, 1:23 AM    12.43 min
```

Training loss vs. epoch (normalized by batch size of 128):



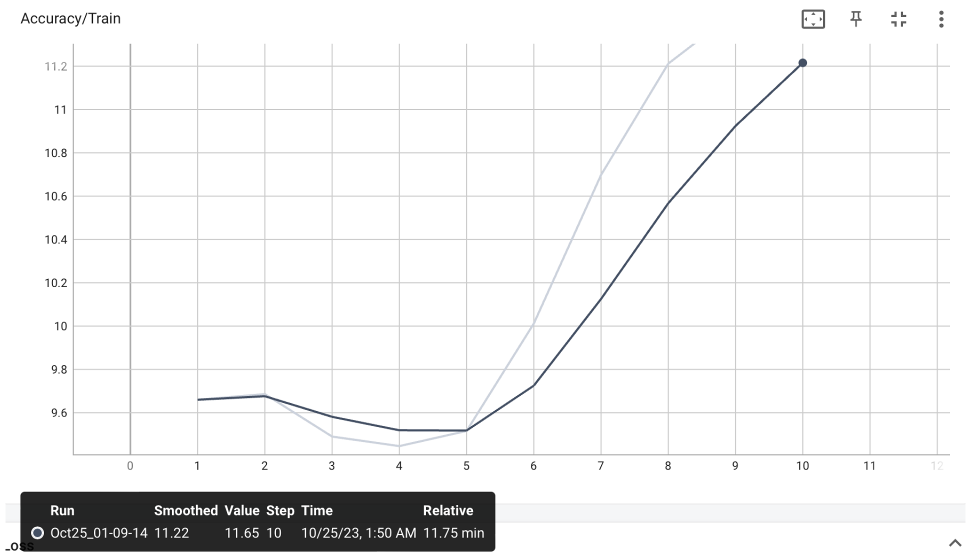
Test accuracy vs. epoch:



| Run | Smoothed | Value | Step | Time | Relative |
|---|---|---|---|---|---|
| Oct25_01-09-14 | 60.28 | 61.3 | 10 | 10/25/23, 1:23 AM | 12.39 min |

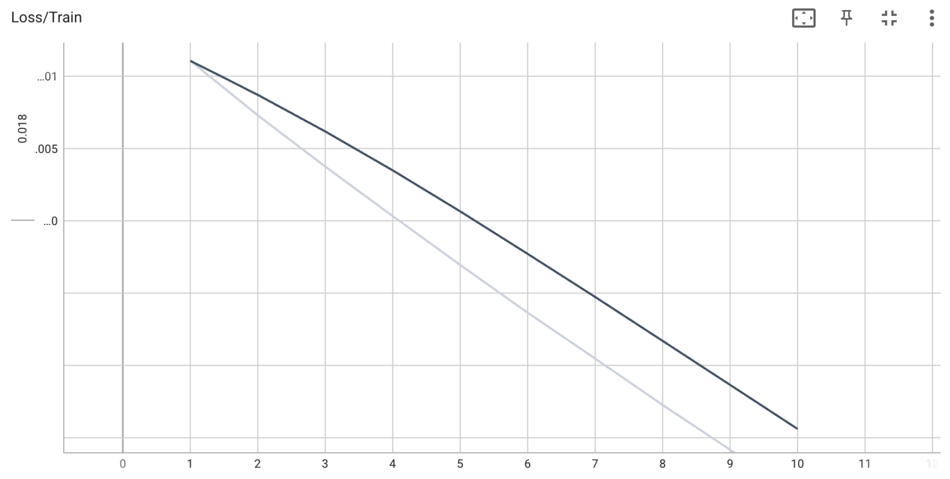
With an SGD optimizer, the test accuracy after 10 epochs was 11.33%.

Training accuracy vs. epoch:

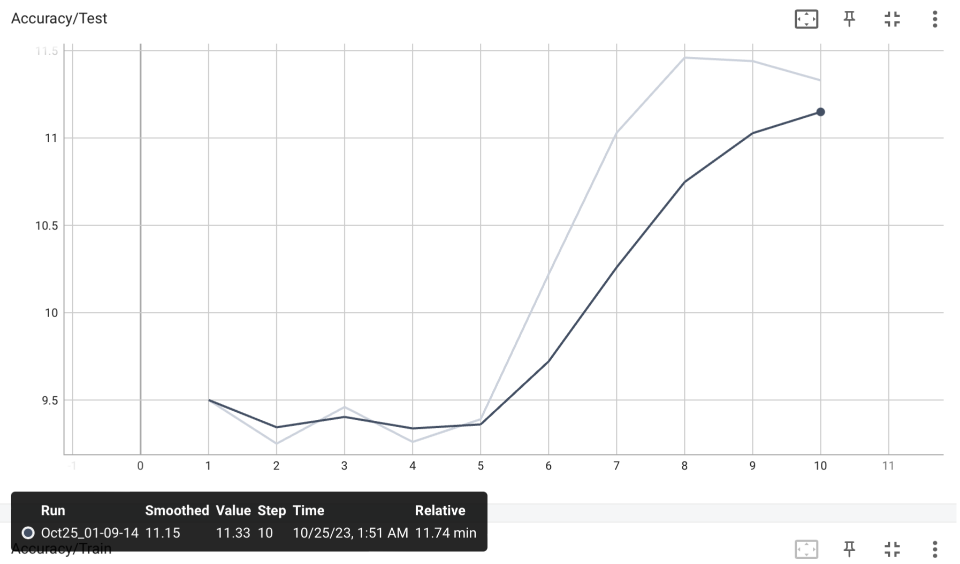


| Run | Smoothed | Value | Step | Time | Relative |
|---|---|---|---|---|---|
| Oct25_01-09-14 | 11.22 | 11.65 | 10 | 10/25/23, 1:50 AM | 11.75 min |

Training loss vs. epoch (normalized by batch size of 128):



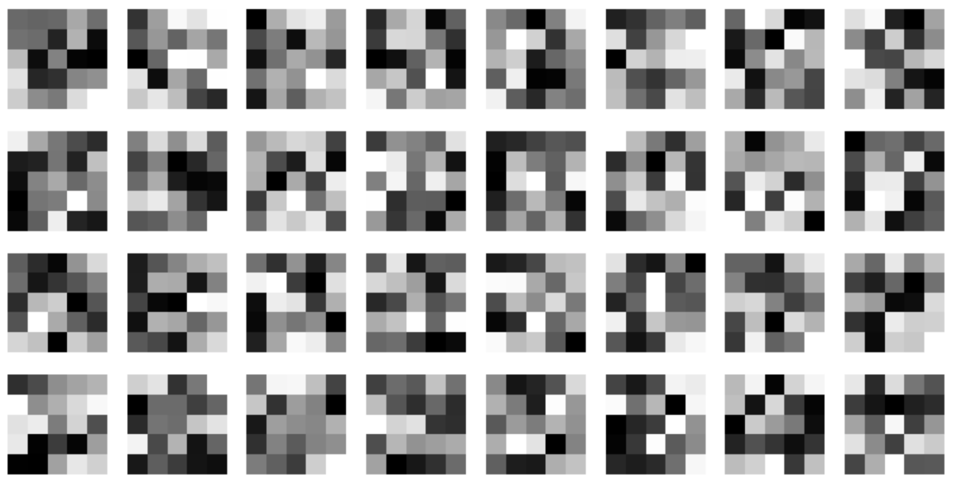
Test accuracy vs. epoch:



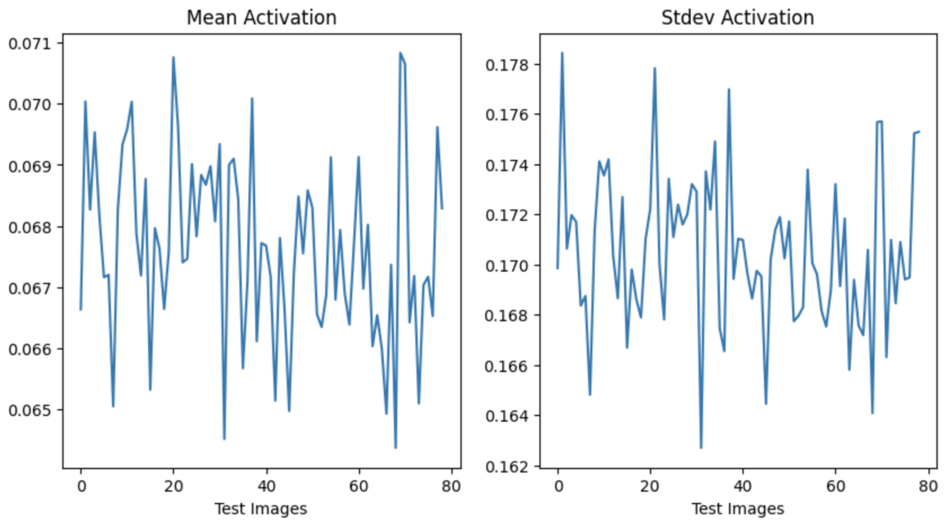
| Run | Smoothed | Value | Step | Time | Relative |
|---|---|---|---|---|---|
| Oct25_01-09-14 | 11.15 | 11.33 | 10 | 10/25/23, 1:51 AM | 11.74 min |

## ***Visualizing Trained Network***

Using an Adam optimizer and 1e-4 learning rate, the first convolutional layer’s weights were visualized (shown below), and roughly resemble Gabor filters (edge detectors/edges on the dataset images).



The mean and standard deviation of the activations in the convolutional layers on the test images are shown below.



## II. Paper: Visualizing and Understanding Convolutional Networks

Zeiler, M. D., & Fergus, R. (2013). *Visualizing and understanding convolutional networks.* arXiv. https://doi.org/10.48550/arXiv.1311.2901

***Summary of Key Ideas***

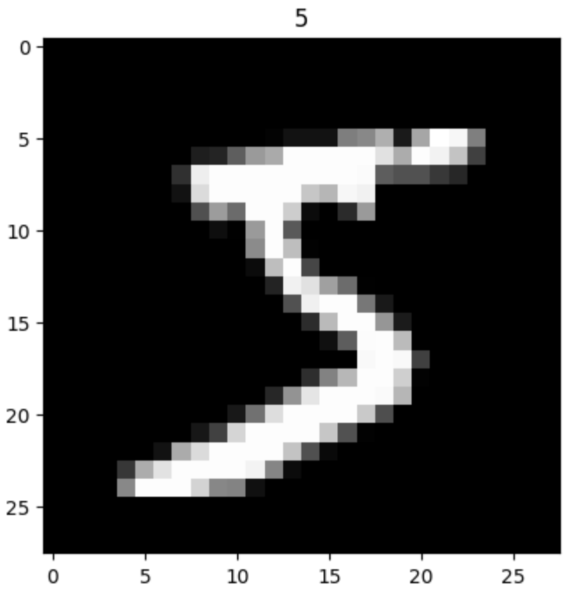
In this publication, Zeiler and Fergus propose a method for better visualizing the hidden (intermediate) layers of a deep convolutional neural network, with the hope of improving the explainability and interpretability of the internal mechanisms of these complex networks. Their method, applied to image object recognition, involves modifying a previously described ‘deconvolutional network’ that reconstructs an input-space representation of node activations in hidden layers. This representation uses both the weights of the trained convolutional neural network and max-pool mappings (‘mapping features to pixels’). Unpooling, rectification, and filtering steps are iteratively used to return to the input pixel space from the final layer.

Visualizations of these deconvolution layers are very interesting; more superficial layers demonstrate that the convolutional network learns simpler patterns in the image data, such as edges, color detectors, contours, and texture, while deeper layers highlight more complex patterns, such as object components, ‘compositionality’, ‘increasing invariance’, and ‘class discrimination.’ They demonstrate these concepts by training their model on the ImageNet 2012 training set, visualizing the sequential layer filters, and ranking the ‘performance contribution’ of specific layers. More generally, this approach allows for finer tuning of model hyperparameters/ model design with the greater resolution afforded by visualizing hidden layers.

## III. Building and Training an RNN on MNIST

### ***Designing RNN, GRU, & LSTM***

Similar to Problem Set #1, the MNIST dataset was used for training a recurrent neural network (RNN), gated recurrent unit network (GRU), and long short-term memory network (LSTM) built with PyTorch. The handwritten digit images of the numbers 0 to 9 are 28x28 pixels. An example digit is shown below.



2-layer RNN, GRU, and LSTM models were designed with the following structure in PyTorch using an Adam or SGD optimizer and CrossEntropyLoss loss criterion:

- Input size: 28 (pixels)
- Hidden size: varies, see below (number of hidden units)
- Layers: 2

This was followed by a size 10 fully connected output layer for the 10 digit classes. TensorBoard was used to log training and testing metrics for 10 epochs. The following hyperparameters were tested:

- Learning rate: 1e-1, 1e-3, 1e-4, 1e-5
- Optimizer: Adam, SGD
- Iterations (epochs): up to 10
- Number of nodes (channels) in hidden layer: 5, 10, 20, 30

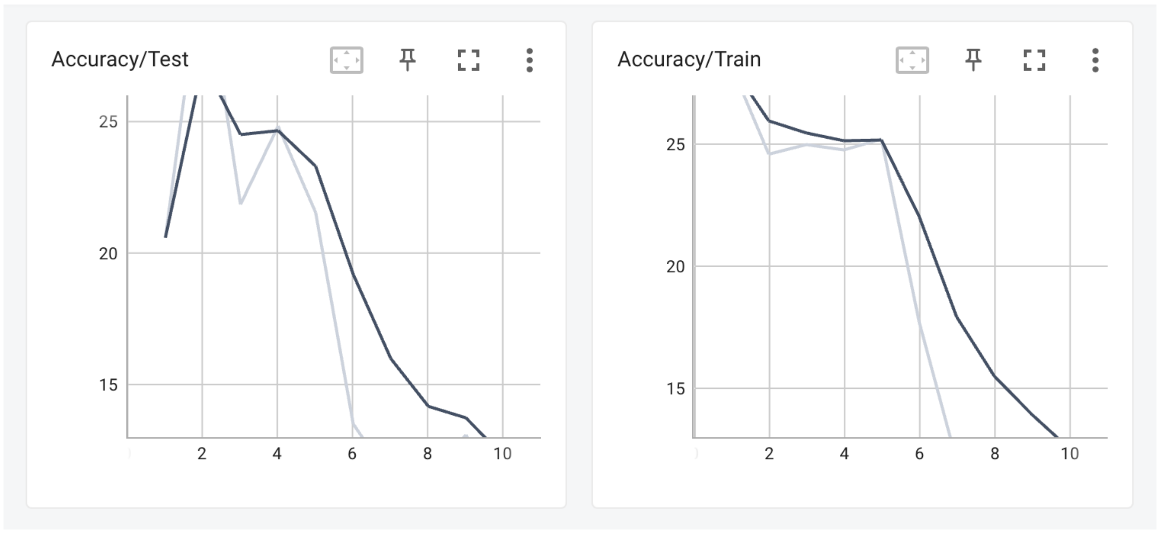
A 1e-3 learning rate, Adam optimizer, and 30 nodes in the hidden layer were found to produce the best test accuracy of 94.17% (see below).

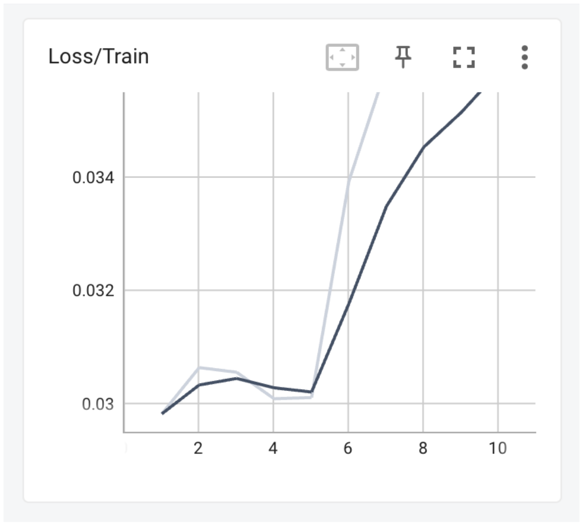
***Training & Testing RNN***

All configurations were run for 10 epochs.

**1.**
Learning rate: **1e-1**
Optimizer: Adam
Number of nodes in hidden layer: 5
Test accuracy: 10.09%

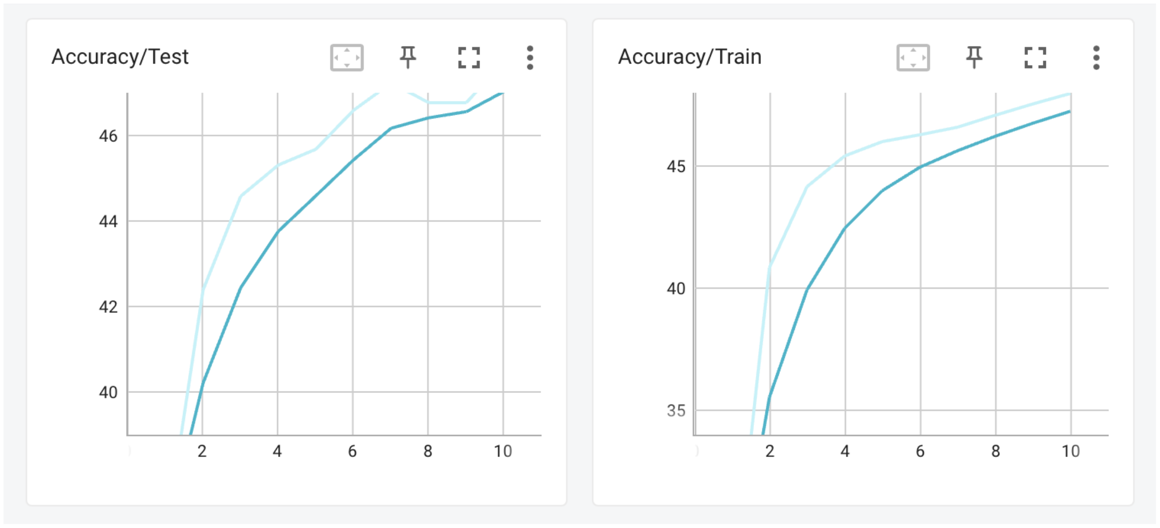
Training accuracy, training loss, test accuracy vs. epoch:

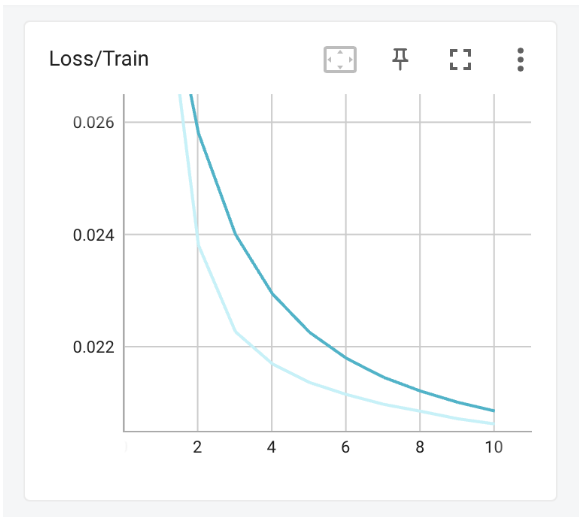




**2.**
Learning rate: **1e-3**
Optimizer: Adam
Number of nodes in hidden layer: 5
Test accuracy: 47.73%

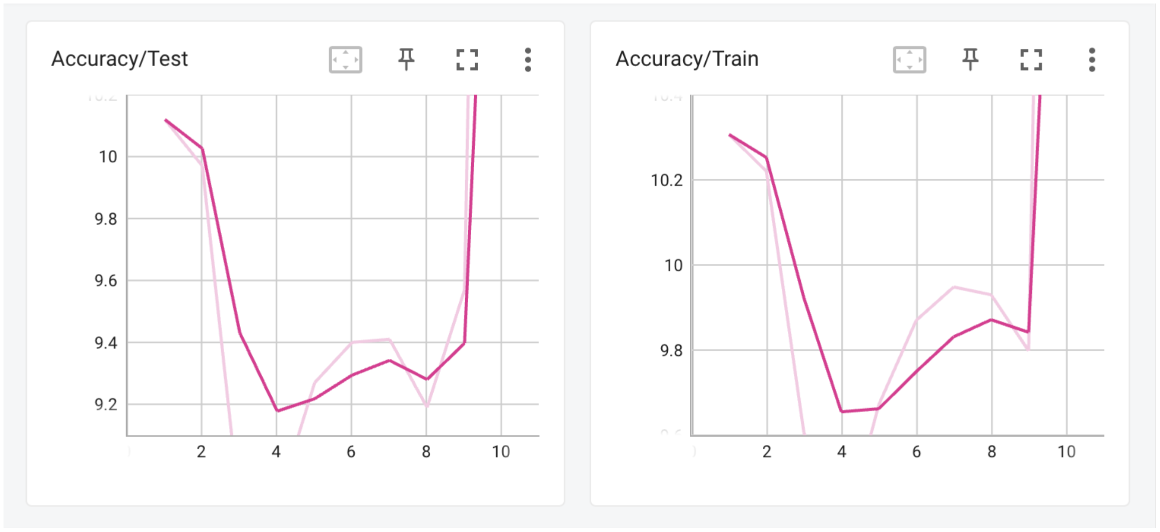
Training accuracy, training loss, test accuracy vs. epoch:

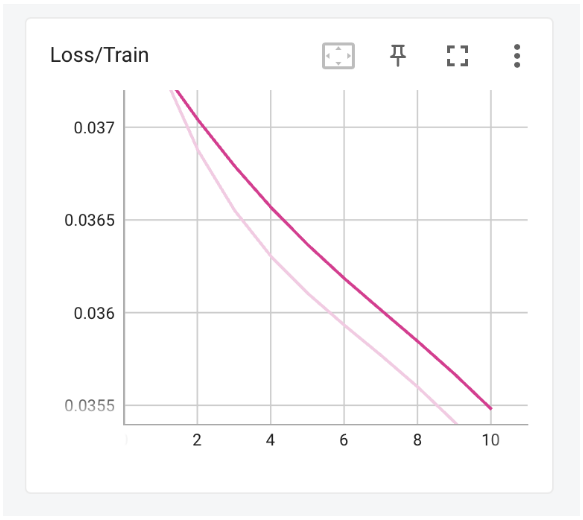




**3.**
Learning rate: **1e-5**
Optimizer: Adam
Number of nodes in hidden layer: 5
Test accuracy: 16.10%

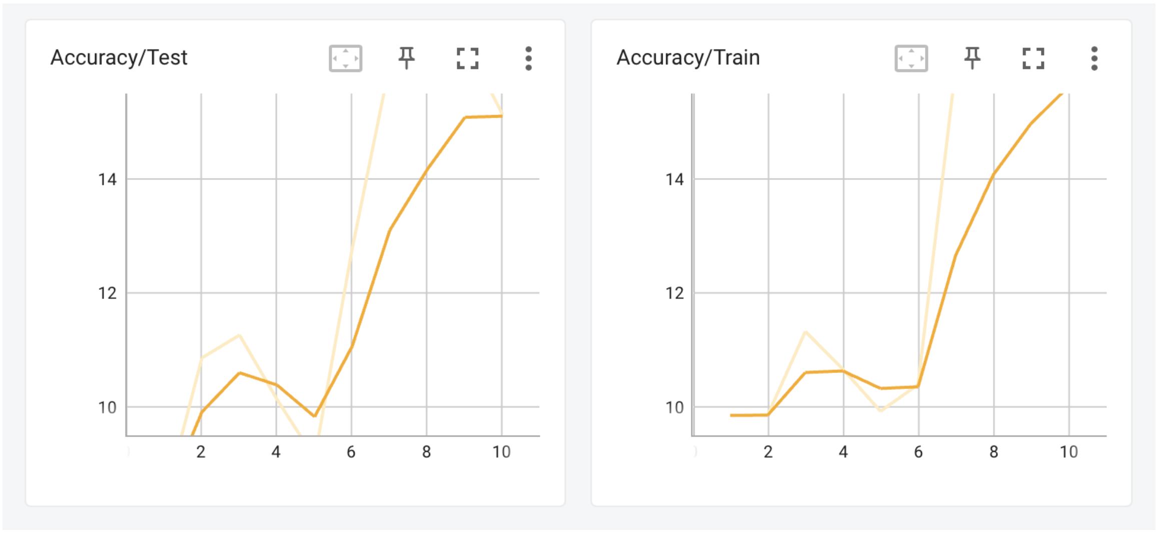
Training accuracy, training loss, test accuracy vs. epoch:

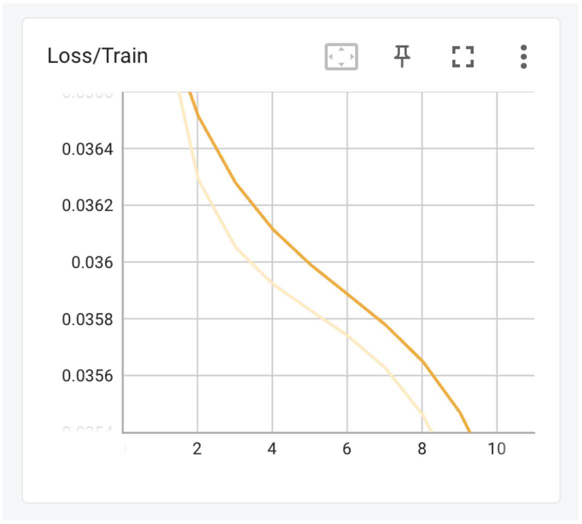




**4.**
Learning rate: 1e-3
Optimizer: **SGD**
Number of nodes in hidden layer: 5
Test accuracy: 15.13%

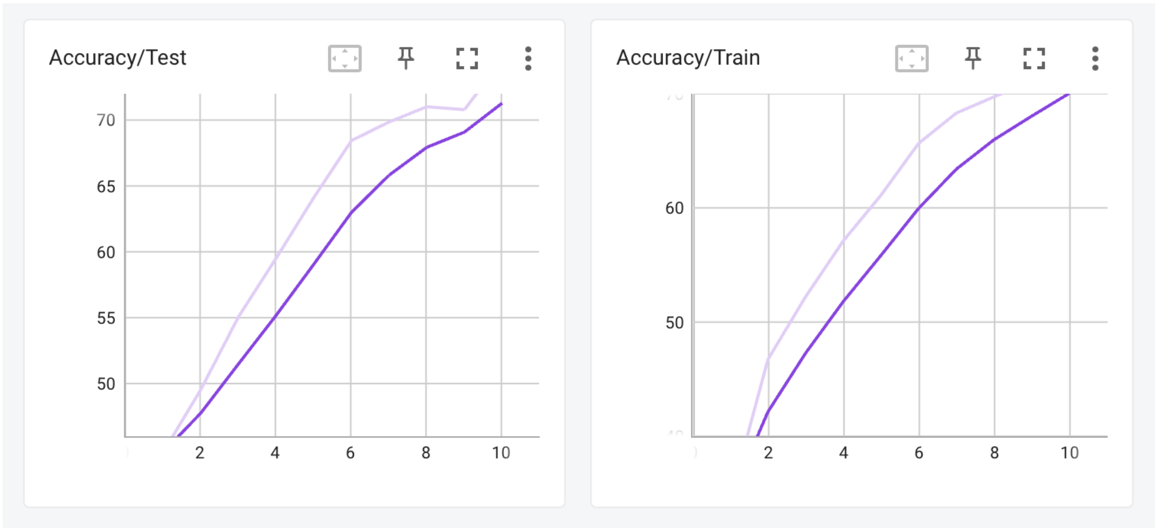
Training accuracy, training loss, test accuracy vs. epoch:

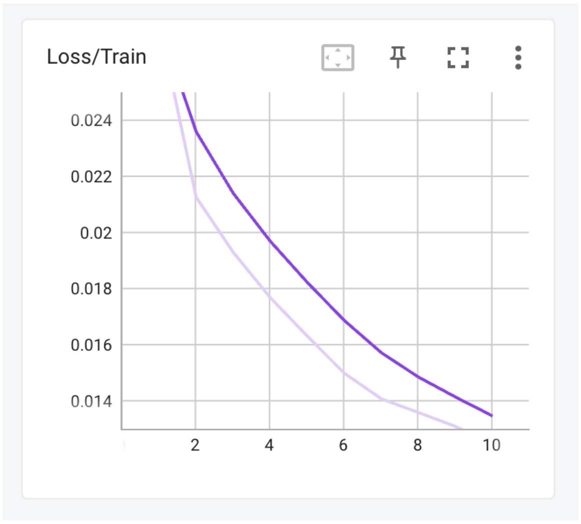




**5.**
Learning rate: 1e-3
Optimizer: Adam
Number of nodes in hidden layer: **10**
Test accuracy: 74.55%

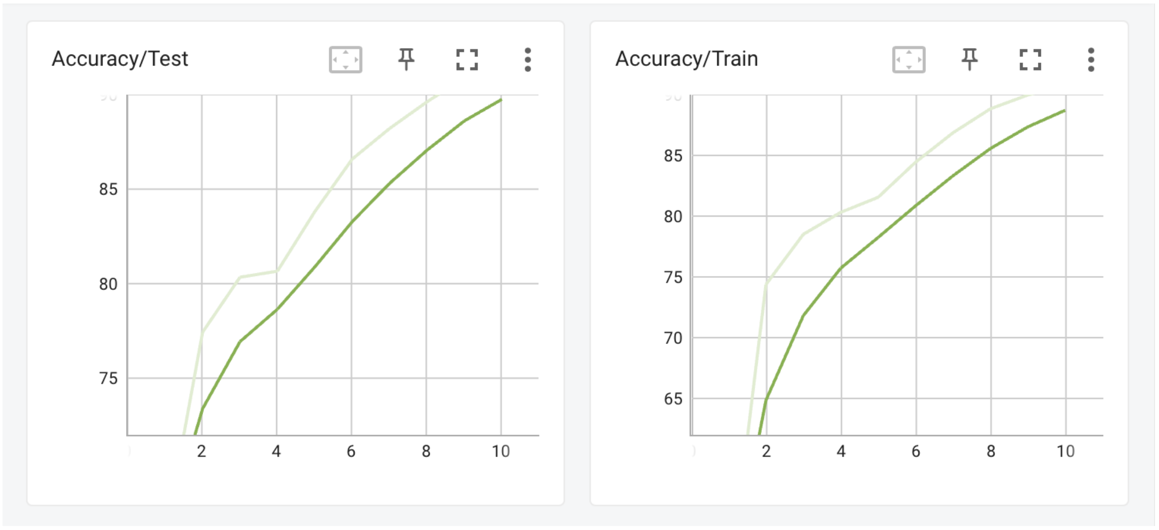
Training accuracy, training loss, test accuracy vs. epoch:

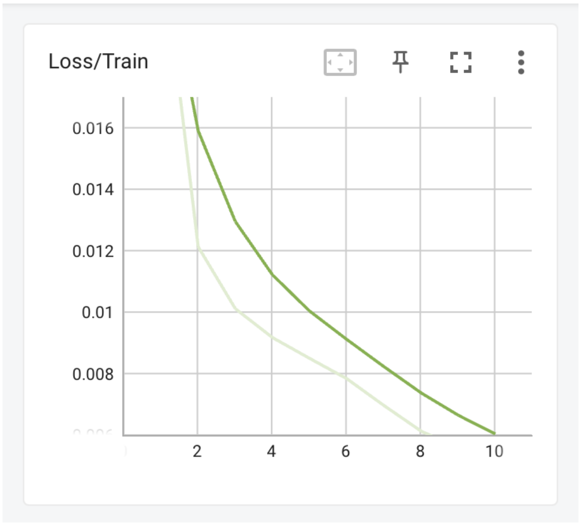




**6.**
Learning rate: 1e-3
Optimizer: Adam
Number of nodes in hidden layer: **20**
Test accuracy: 90.75%

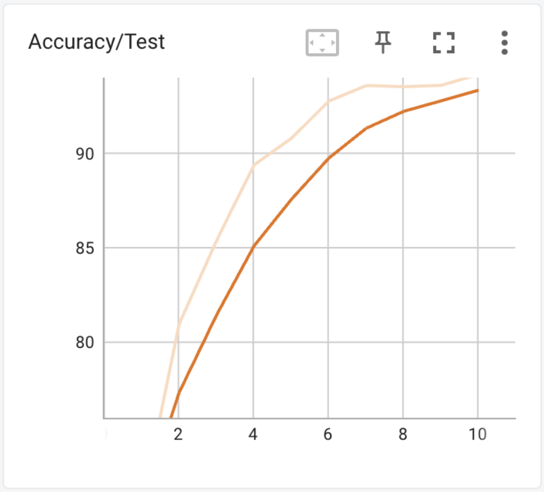
Training accuracy, training loss, test accuracy vs. epoch:

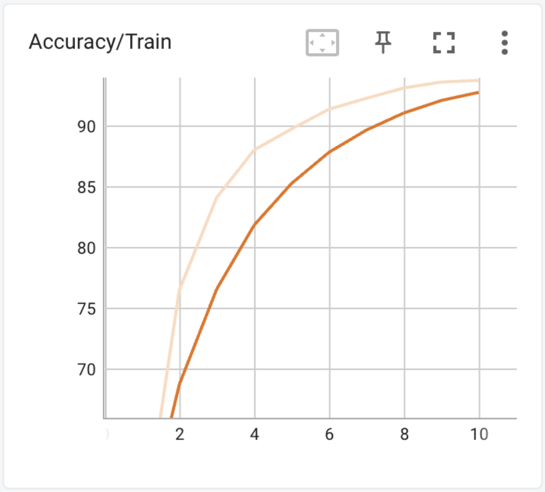


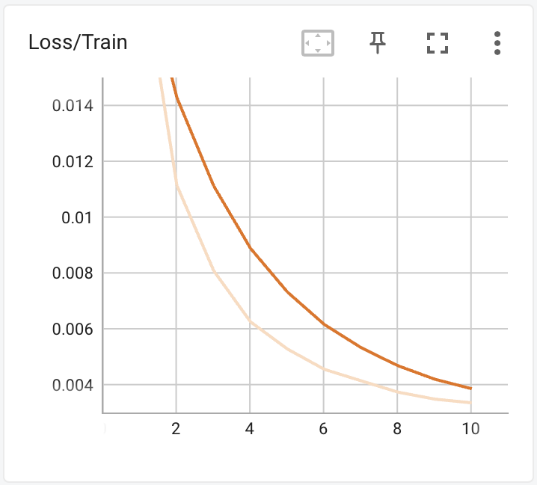


**7.**
Learning rate: 1e-3
Optimizer: Adam
Number of nodes in hidden layer: **30**
Test accuracy: 94.17%

Training accuracy, training loss, test accuracy vs. epoch:





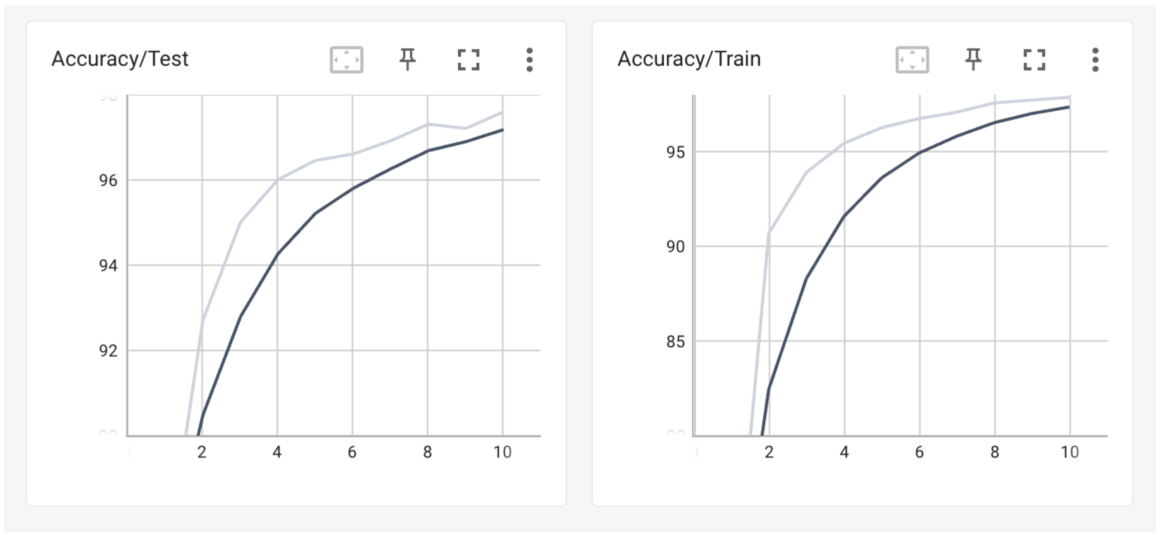


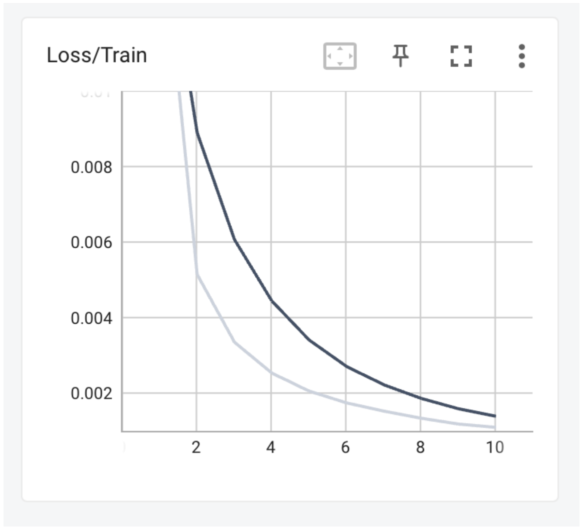
***Training & Testing GRU***

Using GRU, the following performance metrics were logged for 20 and 30 nodes in the hidden layer using an Adam optimizer and 1e-3 learning rate for 10 epochs. The 30 node hidden layer configuration produced a marginally better test accuracy of 98.32%.

**1.**
Number of nodes in hidden layer: **20**
Test accuracy: 97.60%

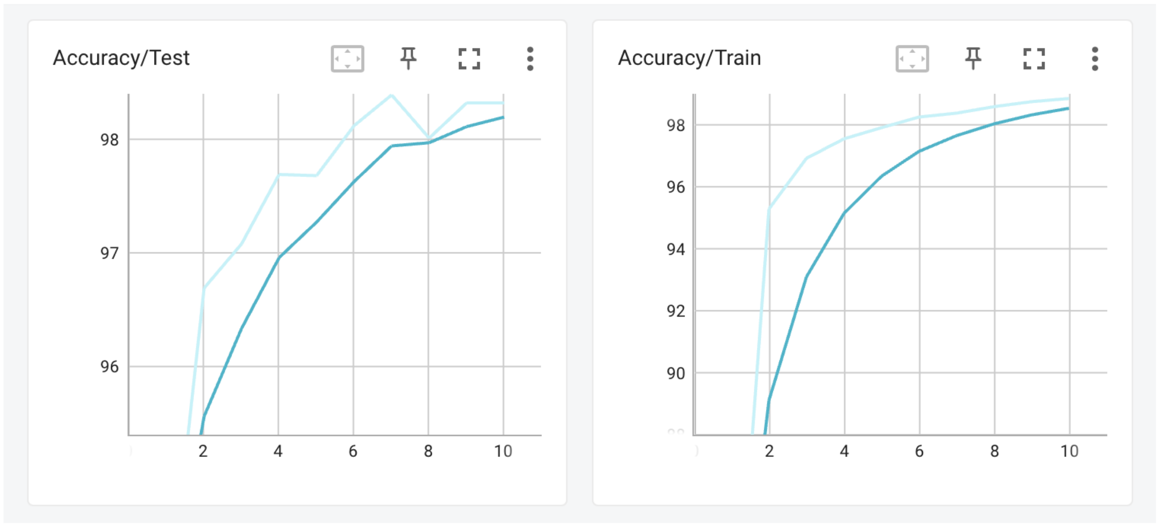
Training accuracy, training loss, test accuracy vs. epoch:

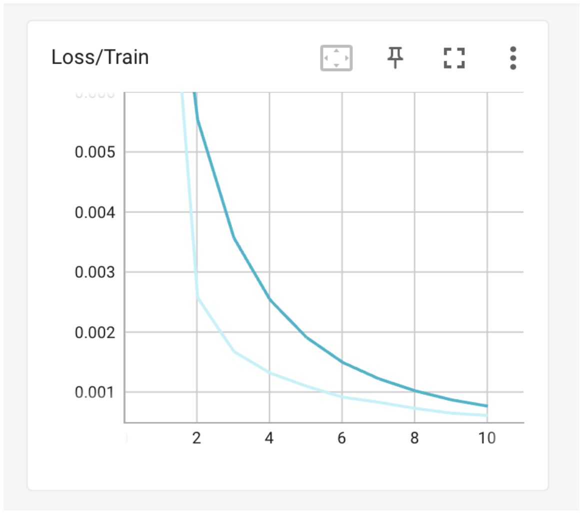




**2.**
Number of nodes in hidden layer: **30**
Test accuracy: 98.32%

Training accuracy, training loss, test accuracy vs. epoch:



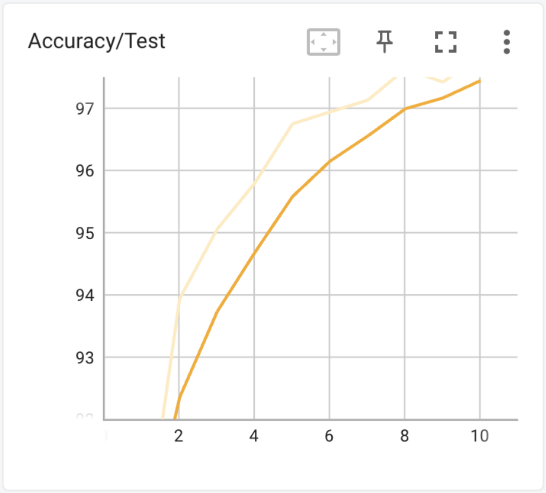


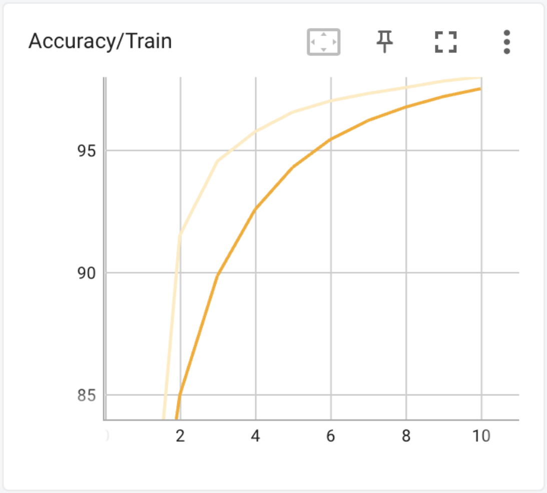
***Training & Testing LSTM***

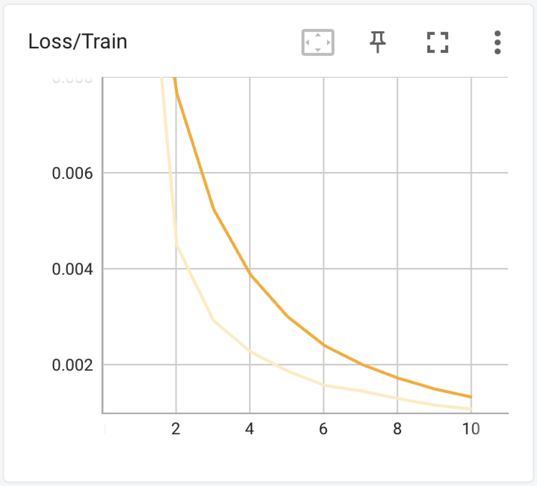
Using LSTM, the following performance metrics were logged for 20 and 30 nodes in the hidden layer using an Adam optimizer and 1e-3 learning rate for 10 epochs. Both 20 and 30 node hidden layer configurations produced nearly identical test accuracies of ~97.85%.

**1.**
Number of nodes in hidden layer: **20**
Test accuracy: 97.86%

Training accuracy, training loss, test accuracy vs. epoch:

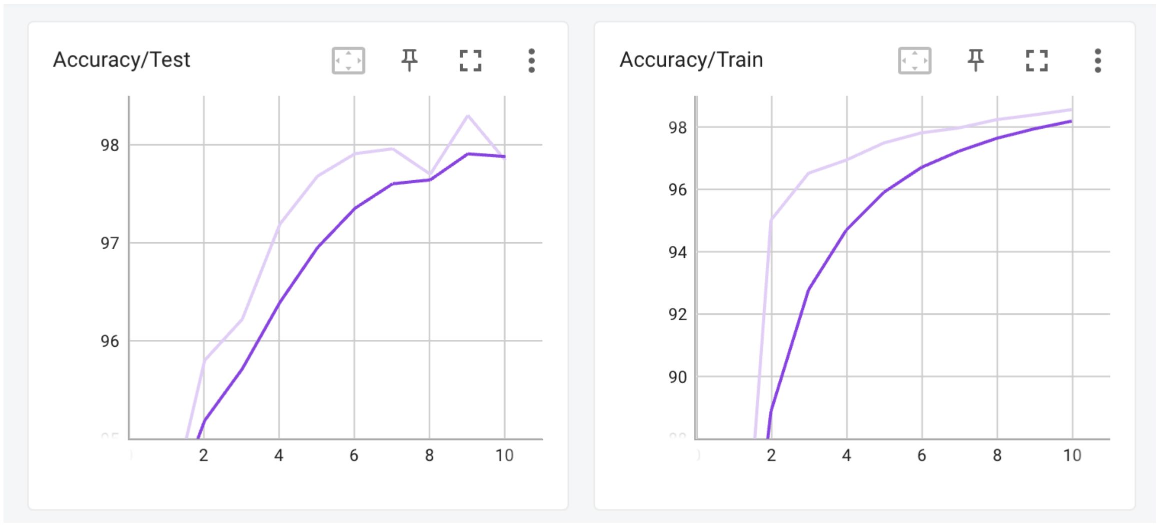


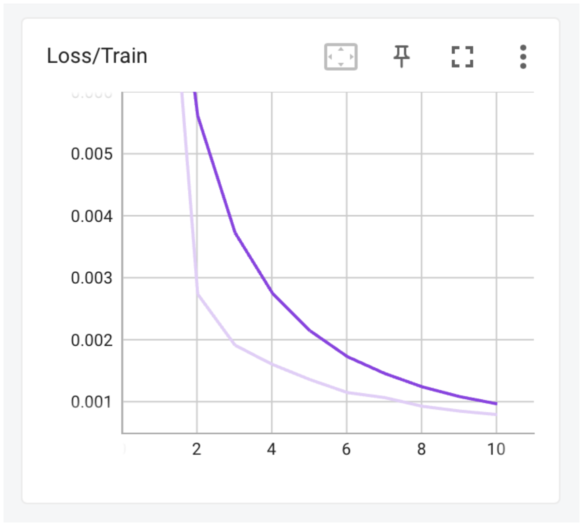




**2.**
Number of nodes in hidden layer: **30**
Test accuracy: 97.84%

Training accuracy, training loss, test accuracy vs. epoch:





***Comparisons between RNN, GRU, LSTM, & CNN***

Holistically, using more epochs and a greater number of nodes in the hidden layer (amongst the range from 5 to 30 tested here) led to greater test accuracy for all of the models. The Adam optimizer outperformed the SGD optimizer, and an ideal learning rate was found to be about 1e-3.

The GRU and LSTM models slightly outperformed the traditional RNN model; both more complex architectures had test accuracies >97%, while the highest RNN test accuracy was ~94%. This is likely due to the fact that the GRU and LSTM models address the 'vanishing gradient problem' in cases where there are long-range dependencies/patterns/relationships in the data. GRU and LSTM similarly handle this by using 'gating mechanisms' that allow for longer-range learning. LSTMs are slightly more computationally expensive to use than GRUs, however.

Compared to the convolutional neural network (CNN) model used in Problem Set #1, neither the RNN, GRU, or LSTM were able to outperform the CNN's 98.5% test accuracy, though the GRU's 98.3% test accuracy was nearly identical. In general, CNNs are better suited for image and video recognition (i.e., spatially-organized data); RNNs (and by extension GRUs and LSTMs) are better suited for text and speech recognition and natural language processing (i.e., time-series/temporal data) where long-range learning is required. Thus, in the case of the image MNIST dataset, it makes sense that CNNs would perform better, though due to the simplicity of this classification task, both perform remarkably well.