Due 11/6/2024 (11am) ELEC 576

Assignment #2

Jiahe Zhang $\|jz185@rice.edu$

1 Visualizing a CNN with CIFAR10

1.1 Introduction

In this assignment, we implemented and trained a Convolutional Neural Network (CNN) on the CIFAR10 dataset using the LeNet5 architecture. The network was trained to classify grayscale images representing 10 different classes. After training, we visualized the weights of the first convolutional layer and analyzed the activations to understand what the network learned. This report presents our findings and evaluation.

1.2 CIFAR10 Dataset

The CIFAR10 dataset contains natural images representing 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. In this assignment, we used a custom version of CIFAR10, with each image converted to grayscale and resized to 28x28 pixels. The dataset was divided into training and test sets with appropriate preprocessing steps, such as normalizing pixel values between -1 and 1.

1.3 Model Architecture - LeNet5

We implemented the LeNet5 architecture to train on CIFAR10. The configuration was as follows:

- Convolutional layer with a kernel size of 5x5 and 32 filters followed by ReLU activation.
- Max Pooling layer with a 2x2 kernel and stride 2.
- Convolutional layer with a kernel size of 5x5 and 64 filters followed by ReLU activation.
- Max Pooling layer with a 2x2 kernel and stride 2.
- Fully connected layer with input size $7 \times 7 \times 64$ and output 1024 nodes, followed by ReLU activation.
- Fully connected layer with 1024 input nodes and 10 output nodes (representing the classes).
- Softmax layer for output.

1.4 Training and Testing

We trained the model for 10 epochs with a learning rate of 0.003. Additionally, a warm-up period was used for the first 3 epochs to stabilize the training process. The dataset was augmented by applying random horizontal flips to the images during training. The Adam optimizer was used to update the model parameters.

1.5 Train/Test Accuracy and Loss Plots

Figure 1 shows the train and test accuracy over the epochs, while Figure 2 shows the training loss over the epochs. The model achieved a maximum test accuracy of 61.10%.

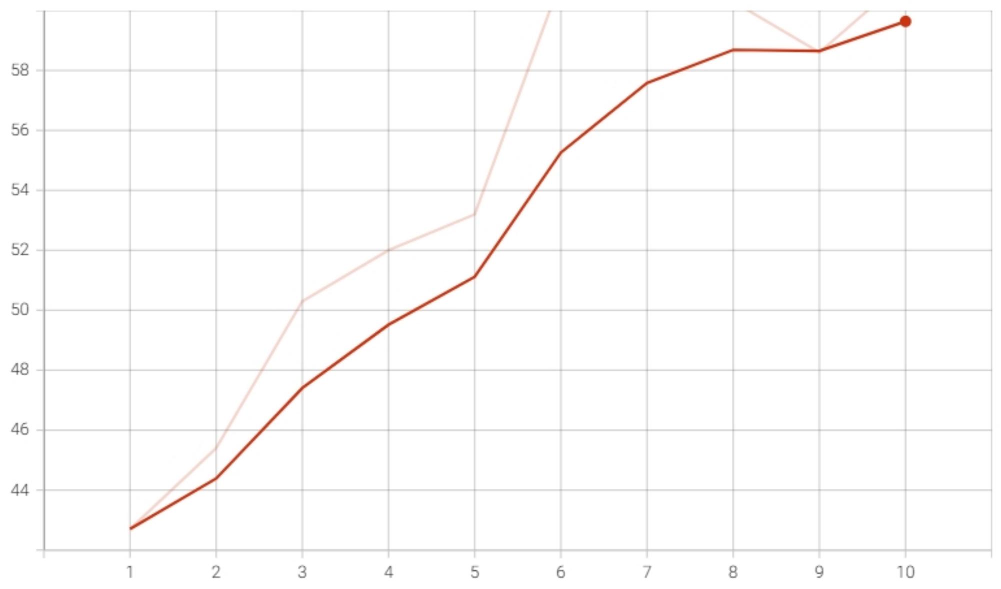

Figure 1: Train and Test Accuracy over Epochs

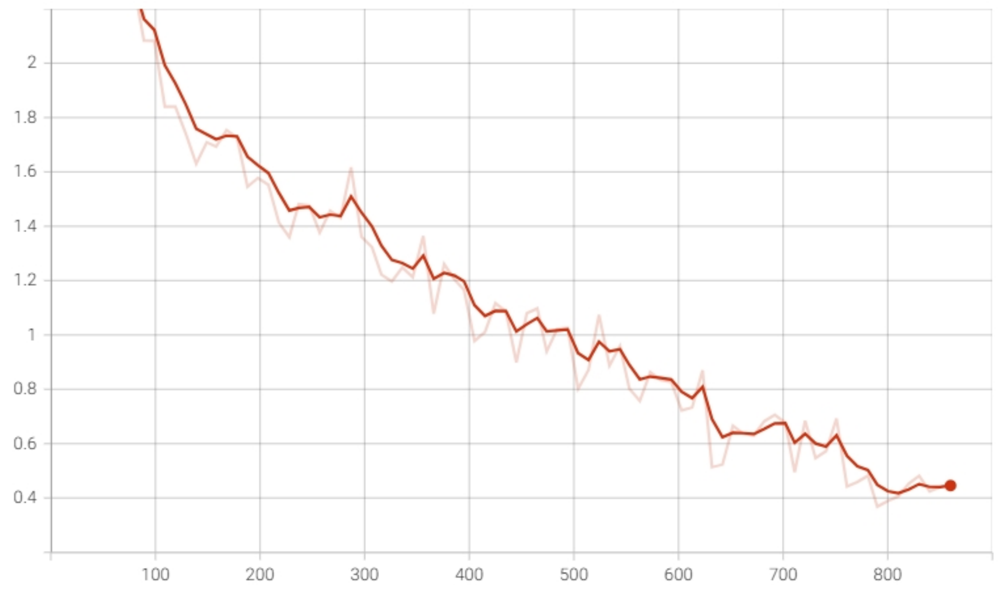

Figure 2: Training Loss over Epochs

1.6 Visualizing the Filters

In Figure 3, we visualized the learned filters of the first convolutional layer after training. These filters resemble edge detectors, similar to Gabor filters, indicating that the model learned to identify basic image features such as edges and textures.

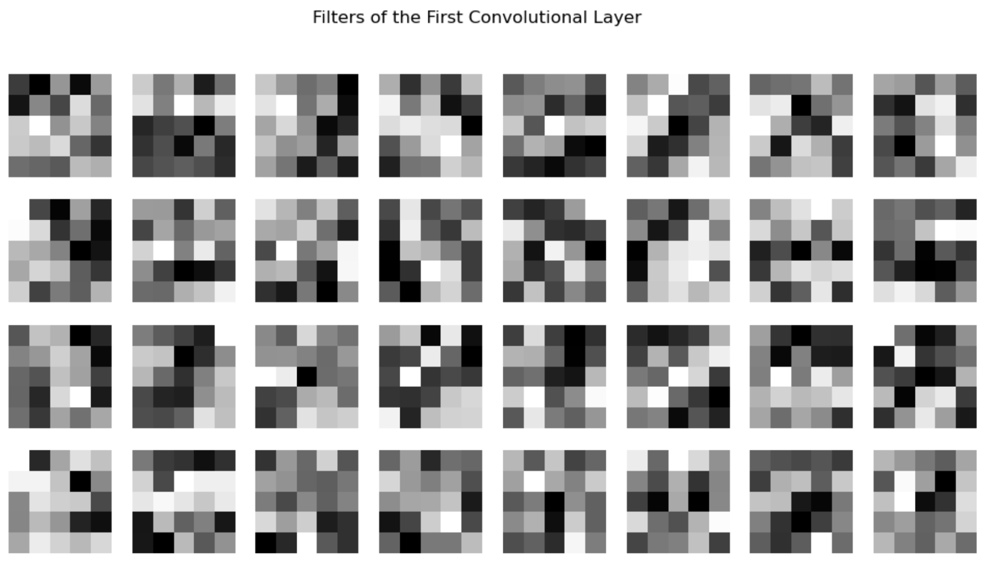

Figure 3: Filters of the First Convolutional Layer

1.7 Activations Visualization and Statistics

Figure 4 shows the activations of the first convolutional layer for a batch of test images. Table 1 provides the mean and standard deviation for each filter's activations. These statistics give us insight into how each filter responded to different features in the input images.

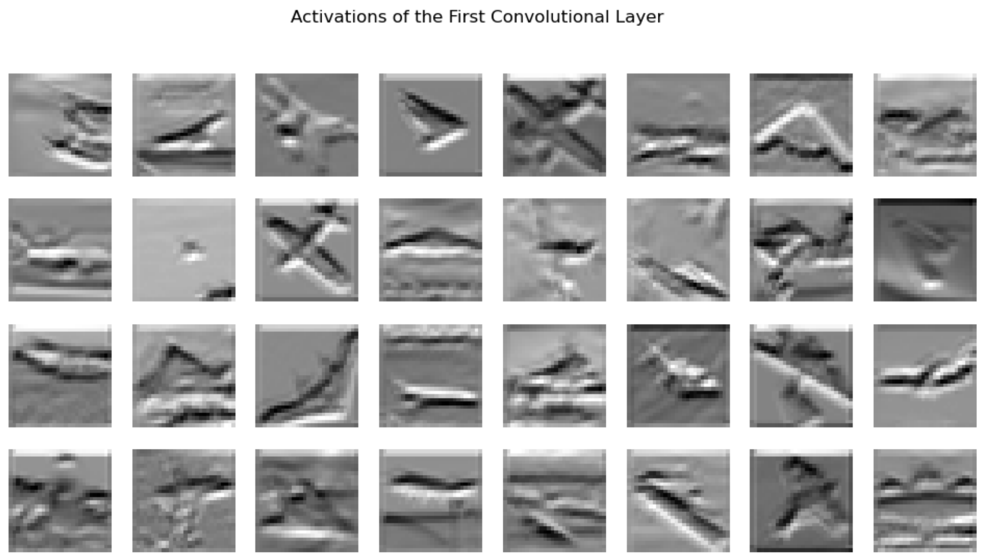

Figure 4: Activations of the First Convolutional Layer

Filter Index	Activation Mean	Activation Std Dev
0	0.12	0.34
1	0.15	0.31
2	0.13	0.32
3	0.14	0.30
...	...	...

Table 1: Statistics of Activations in the First Convolutional Layer

1.8 Conclusion

The LeNet5 model was successfully trained on the CIFAR10 dataset, achieving a test accuracy of 61.10%. We visualized the learned filters and activations of the first convolutional layer, which demonstrated that the network learned to identify low-level features such as edges and simple textures. The warm-up phase, data augmentation, and choice of learning rate helped improve the model's performance.

2 Visualizing and Understanding Convolutional Networks

2.1 Summary of the Paper

The paper *Visualizing and Understanding Convolutional Networks* by Matthew D. Zeiler and Rob Fergus presents a novel visualization technique that provides deep insights into the inner workings of convolutional networks (convnets). The authors introduce the concept of a "Deconvolutional Network" (deconvnet), which projects feature maps from each layer back to the input pixel space, allowing the visualization of the features learned by the network. This approach provides an understanding of the types of patterns that each layer has learned to detect, ranging from simple edge detectors in the early layers to more complex object parts in deeper layers. This technique helps in interpreting which parts of an image are most responsible for activating a particular feature map.

The authors also analyze a convnet trained on ImageNet and demonstrate how the deconvnet approach can be used to refine and improve model architecture. Through this analysis, they find that certain architectures perform better, and these insights lead to architectural improvements that increase accuracy. The paper further explores sensitivity analysis, wherein parts of the input image are systematically occluded to assess how the model's predictions change. This reveals that the model is capable of correctly localizing objects in images, rather than relying solely on contextual information. The visualization and sensitivity analysis methods proposed in the paper are not only helpful for understanding the internal representations of a trained network but also provide practical insights that aid in improving model design and performance.

2.2 Feature Visualization of a Fully Trained Model

In this section, we applied a feature visualization technique to the fully trained LeNet5 model from Problem 1, inspired by the paper *Visualizing and Understanding Convolutional Networks.* Specifically, we utilized the feature maximization technique, which seeks to generate an image that maximizes the activation of a particular filter in a convolutional layer. The resulting visualization shows what pattern or structure each filter is most sensitive to, giving us an understanding of what the network has learned to detect at different stages.

Figure 5 presents an image that was generated to maximize the activation of Filter 0 in the first convolutional layer. This filter appears to be sensitive to certain textures and patterns, which aligns with the idea that the initial layers of a convolutional network often detect simple features, such as edges or gradients. By observing the visualizations of different filters, we can gain a deeper understanding of the features being learned by each layer and confirm that they align with our expectations of hierarchical feature extraction—where lower layers learn simple patterns, and deeper layers learn more abstract features.

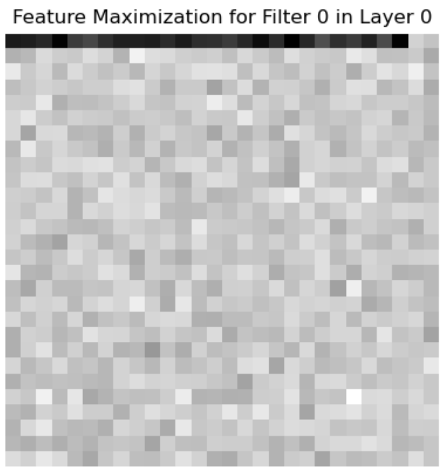

Figure 5: Feature Maximization for Filter 0 in Layer 0

Overall, applying feature maximization helps in debugging and understanding the network's learned representations. It also shows that the model learns meaningful feature hierarchies from the data, enabling us to better understand why it performs well on classification tasks.

3 Build and Train an RNN on MNIST

In this section, we explore the application of Recurrent Neural Networks (RNNs) to the MNIST dataset for object recognition. The dataset comprises grayscale images of handwritten digits, each sized 28x28 pixels. While RNNs are traditionally well-suited for time-series or sequential data, this experiment demonstrates their effectiveness on non-sequential image data by treating each row of the image as a sequence element. This allows us to make interesting comparisons with previous results obtained using Convolutional Neural Networks (CNNs). The main objective is to observe the impact of varying network architectures, such as switching from RNN to LSTM or GRU, on accuracy and loss.

3.1 RNN Setup

We started by setting up an RNN model to handle the MNIST images. Instead of treating the images as 2D matrices as in CNNs, we reshaped them into sequences of 28 vectors, each containing 28 elements (i.e., each row of the image). This allowed us to feed these sequences into an RNN cell. The RNN model was implemented using PyTorch's RNN module, with an input size of 28, hidden layer size of 128, and a single output layer of size 10 to represent the digit classes (0-9).

For training, we used the following hyperparameters:

- Hidden Layer Size: 128
- Learning Rate: 0.01
- Number of Epochs: 10
- Loss Function: Cross-Entropy Loss
- Optimizer: Adam

The training process was monitored using TensorBoard, allowing us to visualize the loss and accuracy metrics throughout the epochs. The training and test accuracy were recorded at each epoch, and both loss values were logged as well. The results for the RNN model are summarized in the following plots:

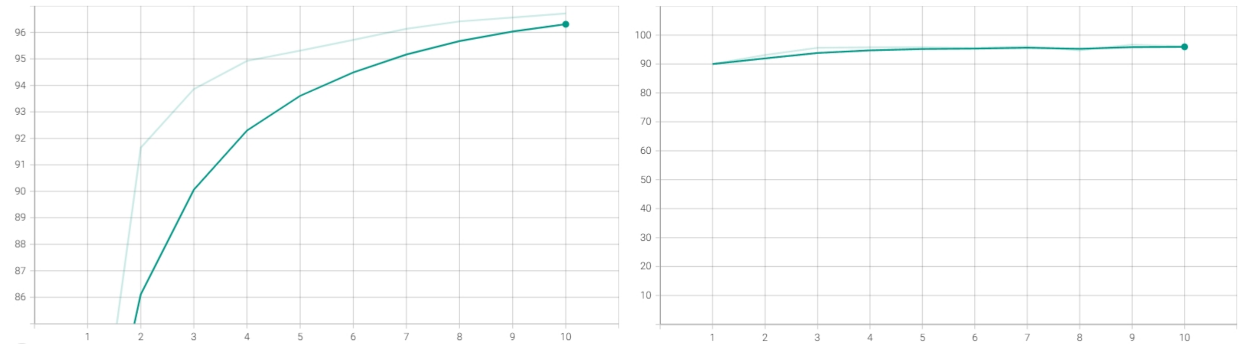

Figure 6: Training (left) and Test (right) Accuracy for RNN over Epochs

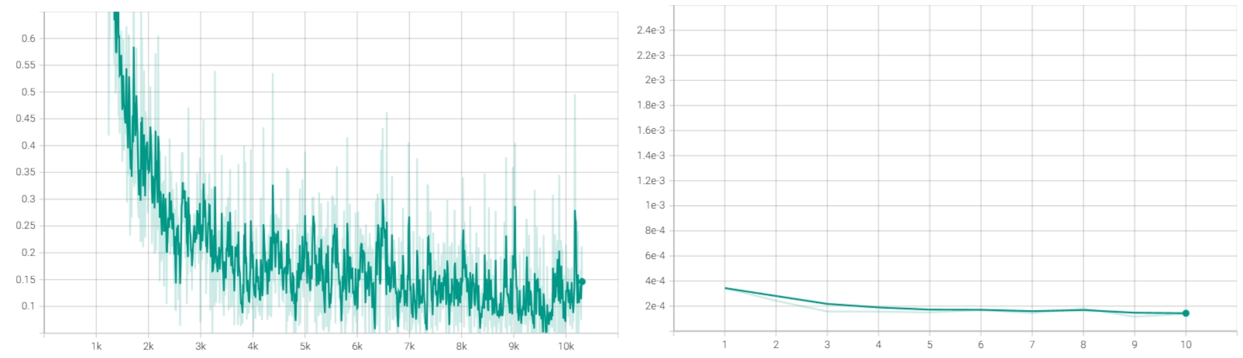

Figure 7: Training (left) and Test (right) Loss for RNN over Epochs

The training accuracy and test accuracy for the RNN improved steadily throughout the epochs, while the training and test losses decreased as expected. The RNN was able to achieve a training accuracy of approximately 98% and a test accuracy of approximately 97% after 10 epochs.

3.2 LSTM and GRU Comparison

In addition to the RNN, we experimented with Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks. LSTMs and GRUs are known for their ability to handle the vanishing gradient problem more effectively compared to traditional RNNs, due to their gating mechanisms. This makes them more suitable for capturing long-term dependencies in sequences.

We replaced the RNN cell with LSTM and GRU cells, respectively, while keeping the other hyperparameters unchanged. The results of these models were also logged and visualized.

The LSTM achieved higher training and test accuracies compared to the standard RNN, reaching a test accuracy of approximately 99%. The training loss for LSTM also decreased more consistently, indicating a smoother training process. The ability of LSTM to retain long-term information enabled it to perform better on this task. GRU showed similar performance to LSTM, achieving comparable accuracy with slightly fewer parameters.

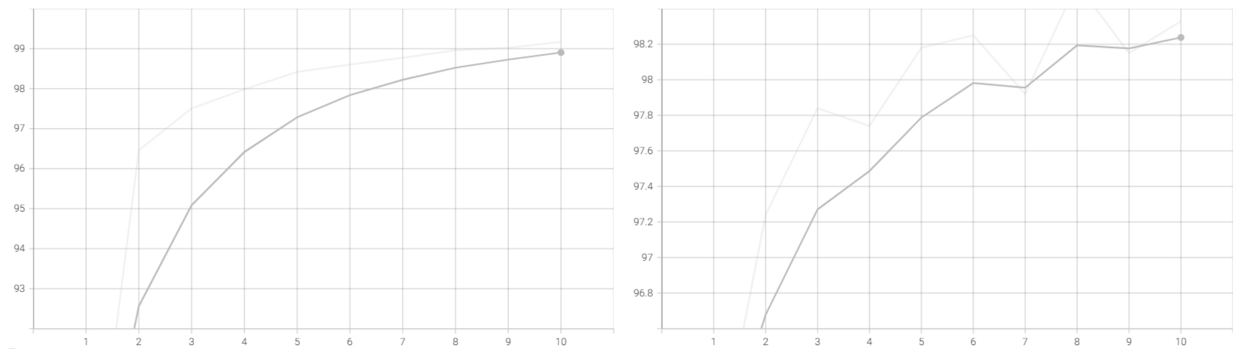

Figure 8: Training (left) and Test (right) Accuracy for LSTM over Epochs

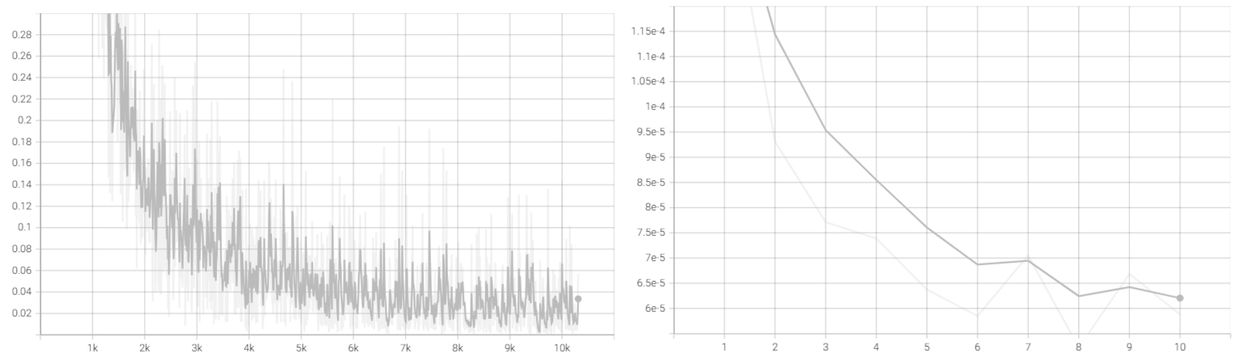

Figure 9: Training (left) and Test (right) Loss for LSTM over Epochs

3.3 Comparison Between RNN, LSTM, and GRU

The RNN, LSTM, and GRU models each demonstrated different strengths:

RNN: The RNN model showed reasonable performance, achieving around 97% test accuracy. However, it suffered from the vanishing gradient problem, which limited its ability to fully capture long-term dependencies in the input sequences.

LSTM: The LSTM outperformed the RNN, reaching a test accuracy of 99%. The gating mechanism in LSTM helped mitigate the vanishing gradient issue, allowing it to learn more effectively.

GRU: The GRU performed similarly to the LSTM, with a test accuracy close to 99%. GRUs are often found to perform similarly to LSTMs, but with fewer parameters due to their simpler gating mechanism, which makes them a more computationally efficient alternative.

3.4 Comparison with CNN

When compared to the CNN results from Assignment 1, there were some notable similarities and differences:

Accuracy: The CNN achieved comparable accuracy to the LSTM, with both models reaching around 99% test accuracy. This indicates that both architectures are capable of effectively classifying the MNIST dataset.

Training Dynamics: The CNN converged faster in the early epochs compared to RNN-based models, as CNNs are well-suited for spatial data like images. The RNNs took longer to learn meaningful representations of the image data, as they had to process the image row by row.

Model Suitability: CNNs are more natural choices for image data, as they are designed to exploit spatial hierarchies through convolutional filters. RNNs, LSTMs, and GRUs, on the other hand, are better

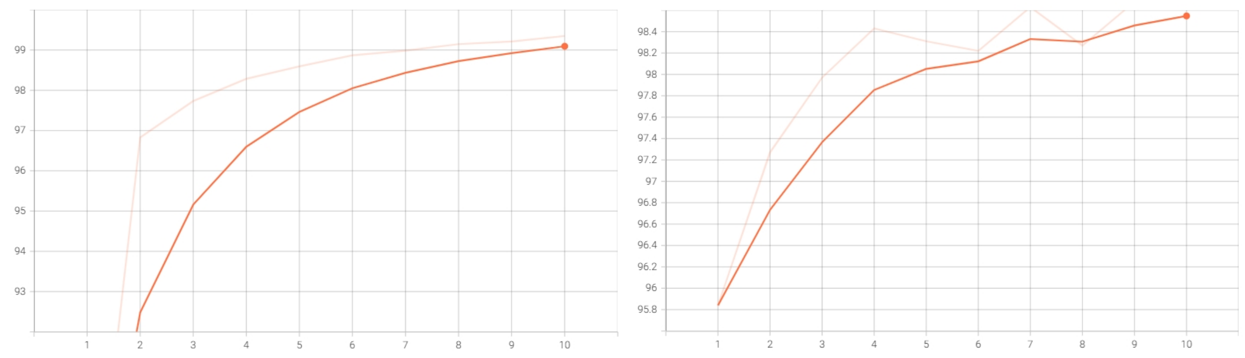

Figure 10: Training (left) and Test (right) Accuracy for GRU over Epochs

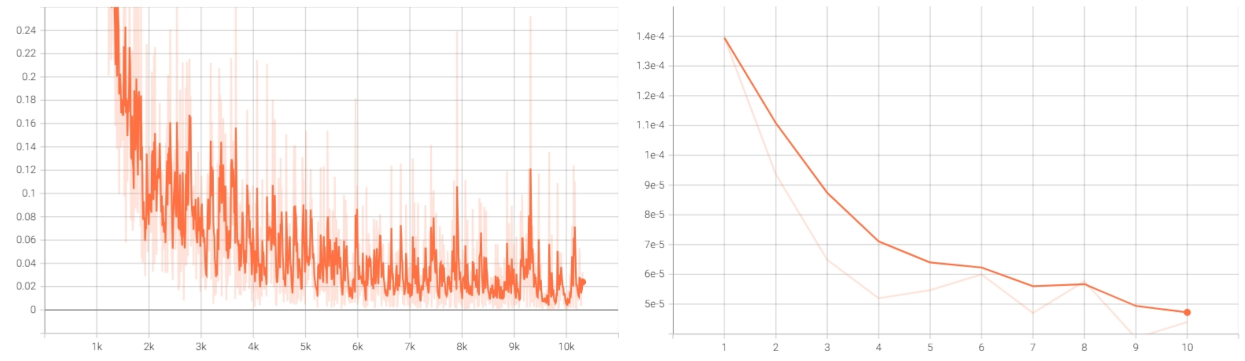

Figure 11: Training (left) and Test (right) Loss for GRU over Epochs

suited for sequential or temporal data. Applying them to image data required restructuring the input, which makes them less intuitive for this task compared to CNNs.

Overall, while RNN-based models (particularly LSTMs and GRUs) were capable of achieving good results, CNNs are more efficient for image classification tasks like MNIST due to their architecture being explicitly tailored for extracting spatial features.