Computer Vision
ENCS5343

Course Project

Arabic Handwritten Text Identification Using Deep Learning Techniques

Prepared by:

Name: Jouwana Daibes	ID: 1210123	Section: 2
Name: Lana Mussafer	ID: 1210455	Section: 1

Instructor:
Dr. Aziz Qaroash

Date:
January 23, 2024

1 Introduction

This project aims to create a deep learning solution for writer detection from Arabic-word images through the AHAWP dataset. Using CNNs which are proven at capturing image patterns helps our research correctly determine where each word image was written.

The project has four separate tasks to reach its objective. Our first step requires creating new CNNs starting with their basic structure and tuning their technical properties for our database. Our next step helps the chosen model perform better by adding data variations which boost both training data quantity and reliability. The third task tests an available deep learning model with applied data augmentation methods to examine its value versus customized architectures. In the last phase we use transfer learning to adapt a pre-trained CNN network for our dataset while retaining its knowledge from similar tasks. Our standardized methodology tests different training methods with different network architectures to find the best solution for writer identification tasks.

2 Dataset Description

The dataset used in this assignment consists of handwritten words from multiple writers. It includes 53199 alphabet images organized into folders corresponding to each writer. The data set was divided into 80included resizing all images to a uniform size of 128x128 pixels, converting them to grayscale, and applying Gaussian blur for noise reduction. This consistent preprocessing ensured that the features extracted from the images were reliable and standardized for classification tasks.

3 Experimental Setup and results

We conducted a series of experiments to evaluate CNN models for identifying writers of word images. In Task 1, we designed and trained custom CNN architectures by experimenting with various convolutional, pooling, and fully connected layers. After hyperparameter tuning, we selected the best-performing model. For Task 2, we retrained the chosen model using data augmentation techniques such as rotations, flips, shifts, and brightness adjustments to improve generalization. In Task 3, we trained well-known CNN architecture (Resnet18), with augmented data, comparing their performance to the results from Task 2. Finally, in Task 4, after reviewing several papers, we found a paper linked in [1] that presented a pre-trained model for Handwritten Character Recognition using Deep Learning (Convolutional Neural Networks). Additionally, we used a pre-trained ResNet18 model, trained on ImageNet, to further improve our results. Based on their work, we applied transfer learning by fine-tuning the pre-trained model on our dataset. For each task, we plotted accuracy and loss against epochs to analyze performance, enabling a comprehensive comparison across all experiments. The experiments were implemented using PyTorch, with data augmentation facilitated by torchvision.transforms.

Note: The results for all cases (accuracy and loss plots vs. epochs) across all tasks are presented in the Appendix below.

3.1 Task 1

3.1.1: Designing and Training Custom CNNs with Hyperparameter Tuning

Parameter	**Type/Value**	**Case 1**	**Case 2**	**Case 3**	**Case 4**	**Case 5**	**Case 6**	**Case 7**	**Case 8**
Architecture	Number of Layers	2 Conv, 1 FC	2 Conv, 1 FC	3 Conv, 1 FC	3 Conv, 2 FC	4 Conv, 2 FC	4 Conv, 1 FC	4 Conv, 3 FC	4 Conv, 3 FC
	Activation Function	Sigmoid	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU
Convolutional Layers	Number of Filters	32, 64	32, 64	32, 64, 128	32, 64, 128, 256	32, 64, 128	32, 64, 128, 256	32, 64, 128, 256	32, 64, 128, 256
	Filter Size	3x3	3x3	3x3	3x3	3x3	3x3	3x3	3x3
	Stride	1	1	1	1	1	1	1	1
	Padding	1	1	1	1	1	1	1	1
Pooling Layers	Pool Size	2x2, Stride: 2	2x2, Stride: 2	2x2, Stride: 2	2x2, Stride: 2	2x2, Stride: 2	2x2, Stride: 2	2x2, Stride: 2	2x2, Stride: 2
	Pool Type	Max	Max	Max	Max	Max	Max	Avg	Max
Fully Connected Layers	Number of Neurons In Flatten Layer	64×37×37	64×37×37	128×18×18	128×18×18	256×9×9	256×9×9	256x9x9	256×9×9
Training Parameters	Learning Rate	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
	Batch Size	32	32	32	32	32	32	32	32
	Epochs	25	25	25	25	25	25	25	25
	Optimizer	Adam	Adam	Adam	Adam	Adam	Adam	Adam	Adam
	Regularization	Dropout (0.5)	Dropout (0.5)	Dropout (0.5)	Dropout (0.5)	Dropout (0.5)	Dropout (0.5)	Dropout (0.3)	Dropout (0.3)

Parameter	**Type/Value**	**Case 9**	**Case 10**	**Case 11**	**Case 12**	**Case 13**
Architecture	Number of Layers	5 Conv, 3 FC	4 Conv, 3 FC	4 Conv, 3 FC	4 Conv, 3 FC	4 Conv, 3 FC
	Activation Function	ReLU	ReLU	ReLU	ReLU	ReLU
Convolutional Layers	Number of Filters	32, 64, 128, 256, 512	32, 64, 128, 256	32, 64, 128, 256	32, 64, 128, 256	32, 64, 128, 256
	Filter Size	3x3 (Conv1, Conv3, conv5), 5x5 (Conv2, Conv4)	3x3 (Conv1, Conv3), 5x5 (Conv2, Conv4)	3x3 (Conv1, Conv3), 5x5 (Conv2, Conv4)	3x3 (Conv1, Conv3), 5x5 (Conv2, Conv4)	3x3 (Conv1, Conv3), 5x5 (Conv2, Conv4)
	Stride	1 (Conv1, Conv3, conv5), 2 (Conv2, Conv4)	1 (Conv1, Conv3), 2 (Conv2, Conv4)	1 (Conv1, Conv3), 2 (Conv2, Conv4)	1 (Conv1, Conv3), 2 (Conv2, Conv4)	1 (Conv1, Conv3), 2 (Conv2, Conv4)
	Padding	1 (Conv1, Conv3, conv5), 2 (Conv2, Conv4)	1 (Conv1, Conv3), 2 (Conv2, Conv4)	1 (Conv1, Conv3), 2 (Conv2, Conv4)	1 (Conv1, Conv3), 2 (Conv2, Conv4)	1 (Conv1, Conv3), 2 (Conv2, Conv4)
Pooling Layers	Pool Size	2x2, Stride: 2	2x2, Stride: 2	2x2, Stride: 2	2x2, Stride: 2	2x2, Stride: 2
	Pool Type	Max	Max	Max	Max	Max
Fully Connected Layers	Number of Neurons In Flatten Layer	512×4×4	256×9×9	256×9×9	256×9×9	256×9×9
Training Parameters	Learning Rate	0.001	0.001	0.001	0.001	0.001
	Batch Size	32	32	32	32	32
	Epochs	21	15	15	21	15
	Optimizer	Adam	Adam	Adam	Adam	Adam
	Regularization	Dropout (0.3)	Dropout (0.7)	Dropout (0.3)	Dropout (0.3)	Dropout (0.3)

To build our CNN, we followed a systematic approach by experimenting with progressively complex architectures across 13 cases, guided by the accuracy results at each step. We began with a simple architecture in Case 1, comprising two convolutional layers and one fully connected (FC) layer, max pooling, Sigmoid activation function, which achieved an accuracy of 3.74%: Although the initial results were modest, they provided a baseline for further experimentation.

In Case 2, we replaced the Sigmoid activation function with ReLU and retained the same architecture, leading to a slight improvement in accuracy to 4.48%. Building on this, we introduced a third convolutional layer in Case 3, which boosted accuracy to 10.62%. To further enhance performance, we added an additional FC layer in Case 4, achieving an accuracy of 27.07%.

Recognizing the value of increased depth, we introduced a fourth convolutional layer in Case 5, which improved accuracy to 31.80%. In Case 6, we reduced the number of FC layers back to one, slightly increasing accuracy to 34.50%. To evaluate the impact of pooling strategies, we switched to average pooling in Case 7, but this resulted in a marginally lower accuracy of 34.07%. Reverting to max pooling in Case 8 we got back to three FC layers improved accuracy to 34.93%.

In Case 9, we expanded the architecture to five convolutional layers and three FC layers, incorporating alternating filter sizes (3×3 and 5×5) with varying strides and padding. This configuration achieved a notable accuracy of 36.59%. The results motivated us to fine-tune dropout rates and epoch settings, leading to improvements in subsequent cases. Case 10, with a similar architecture but fewer epochs, achieved 37.88%. Cases 11 and 12 continued the trend with accuracies of 39.35% and 40.64%, respectively.

Finally, in Case 13, we arrived at the optimal architecture, featuring four convolutional layers with alternating filter sizes and three FC layers. This configuration, trained with carefully tuned parameters, achieved the highest accuracy of 41.38%. This iterative process of refining the architecture based on accuracy insights allowed us to systematically identify the most effective design for our task.

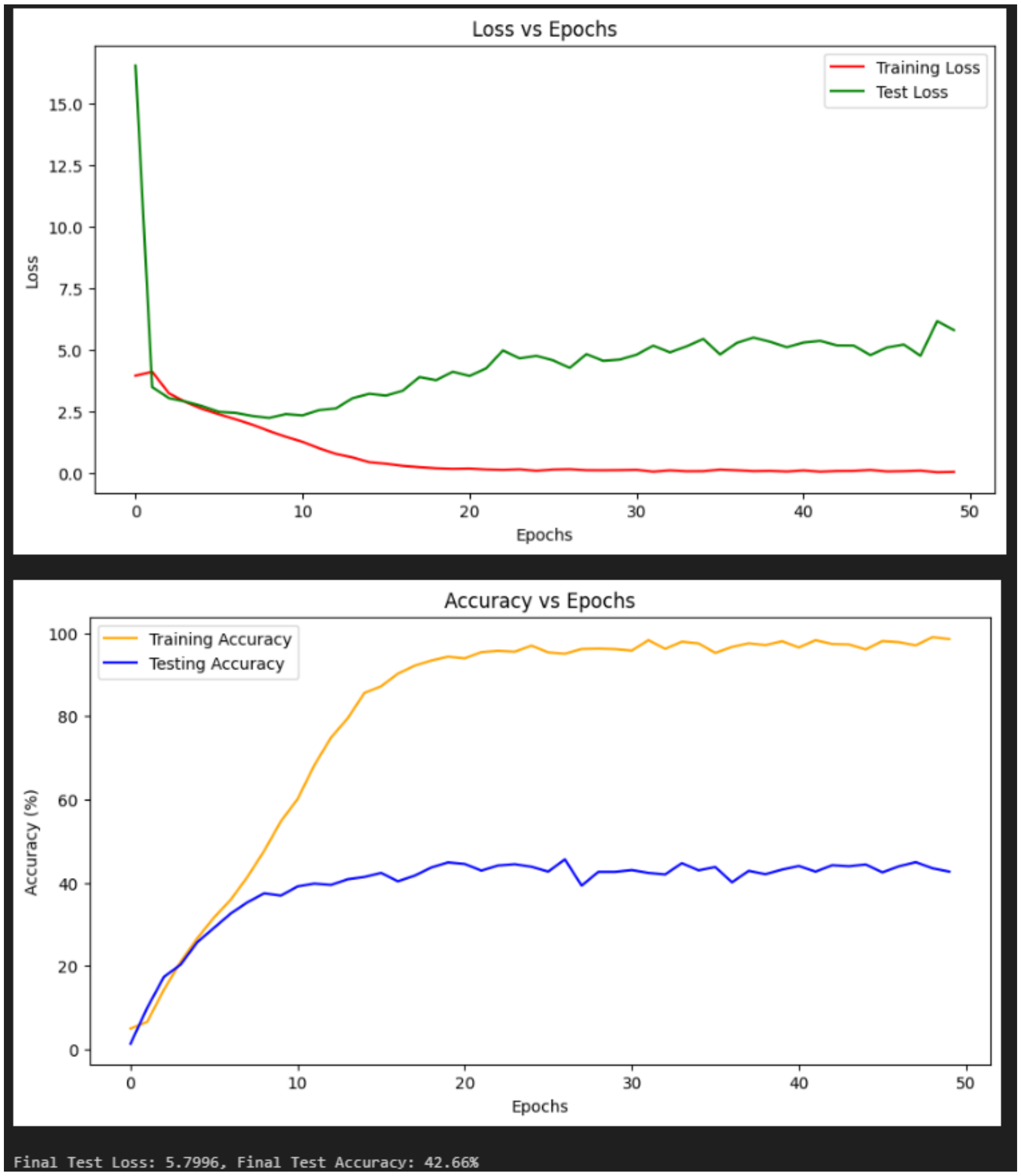
3.1.2 Results and Discussion

Case Number	Train Loss	Train Accuracy (%)	Test Loss	Test Accuracy (%)
1	4.3168	5.23	4.3482	3.74
2	4.0259	7.12	4.2256	4.48
3	0.6142	82.99	7.6855	10.62
4	0.2034	93.35	8.0770	27.07
5	0.0086	99.82	7.6028	31.80
6	0.0359	98.96	7.1511	34.50
7	0.2488	92.00	6.3772	34.07
8	0.1581	94.87	7.0768	34.93
9	0.3964	86.83	4.0670	36.59
10	0.5972	80.48	3.1497	37.88
11	0.7098	76.90	2.9585	39.35
12	0.1297	95.76	4.8791	40.64
13	0.4391	85.59	3.2131	41.38

The table provides an overview of the training and testing performance across 13 experimental cases, highlighting the iterative refinement of the CNN architecture and hyperparameters. Initially, the training loss was high (4.3168 in Case 1), with a modest training accuracy of 5.23%. As the architecture evolved, the training loss decreased significantly, and training accuracy improved, peaking at 99.82% in Case 5. However, the exceptionally low training loss and high accuracy in Case 5 indicate significant overfitting, as the corresponding test accuracy was only 31.80%. Overfitting persisted as a challenge, particularly in cases with deeper architectures, where the training accuracy remained disproportionately high compared to the test accuracy.

To address this, we made deliberate efforts to reduce overfitting by adjusting the architecture, fine-tuning hyperparameters, and incorporating regularization techniques such as dropout. These adjustments led to more balanced training and testing performance, with Case 13 achieving a test accuracy of 41.38% and a training accuracy of 85.59%, reflecting a substantial improvement in generalization. Despite this progress, overfitting was not completely resolved in Task 1. However, the problem was effectively mitigated in Task 2 through the application of data augmentation techniques, which enhanced the model's ability to generalize to unseen data by introducing variability into the training set. This underscores the critical role of both architectural design and data-driven strategies in overcoming overfitting and achieving robust performance.

The figures below illustrate the accuracy and loss for the best-performing model, achieved in Case 13:



```
Final Test Loss: 5.7996, Final Test Accuracy: 42.66%
```

3.2 Task 2

3.2.1 Train The Best Network after Data Augmentation

From the previous task, we chose a network consisting of 4 convolutional layers and 3 fully connected layers, using the ReLU activation function, max pooling layers, a learning rate of 0.001, 15 epochs, and a dropout rate of 0.3. This setup achieved an accuracy of 41.38% at 15 epochs. After that, we applied data augmentation techniques, including random rotation within ±10 degrees, random resizing with a scale range of 0.8 to 1.0, random shearing of up to 10 degrees, random perspective distortion with a distortion scale of 0.5, random brightness, contrast, saturation, and hue adjustments, along with normalization. In this case [Case 5.2.1, go to appendix part for the results], the model is still overfitting and continues to perform poorly on the test set with 12.03% accuracy, even after applying augmentation.

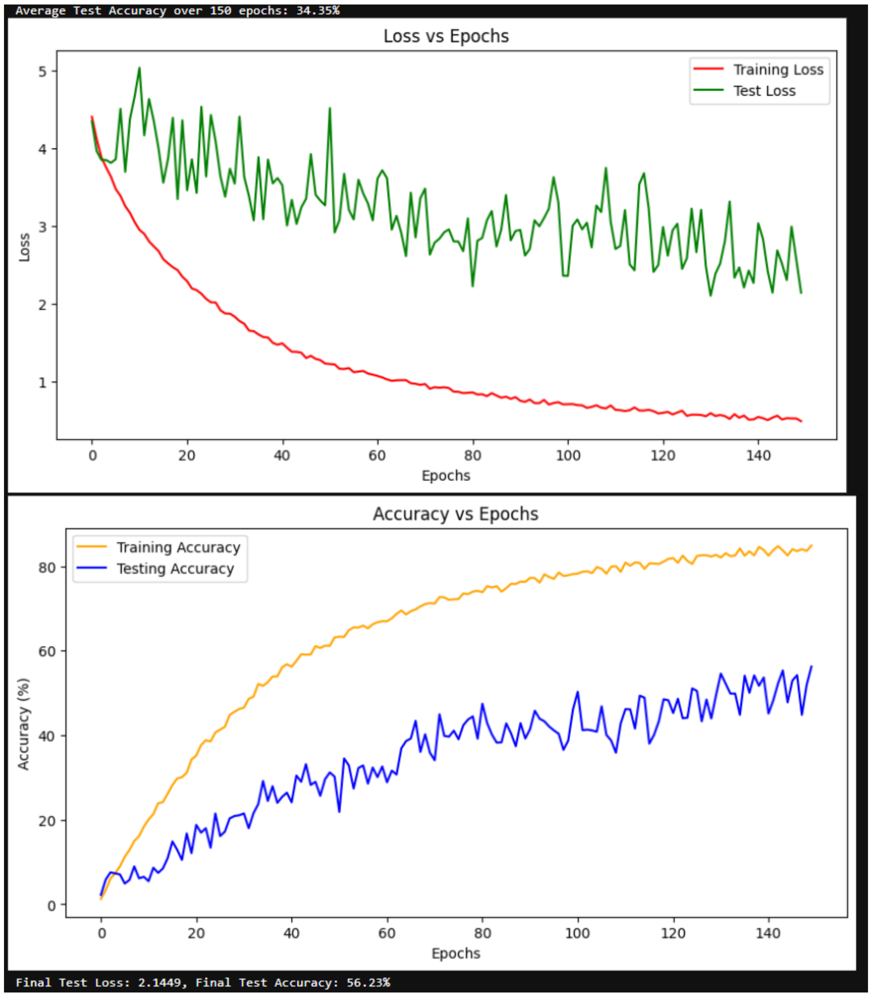
We then adjusted some parameters by increasing the dropout rate from 0.3 to 0.5 for better regularization, adding a learning rate scheduler (StepLR) to dynamically adjust the learning rate during training for improved convergence, and increasing the number of epochs to 20. However, the test accuracy dropped to 8.04% and the model still suffers from overfitting. [Case 5.2.2]

Next, we increased the dropout rate from 0.5 to 0.7 while keeping all other parameters unchanged. Although the accuracy worsened, dropping to 1.47% the model addressed the overfitting issue, as the testing accuracy surpassed the training accuracy. [Case 5.2.3]

We then reset the dropout rate to 0.5, introduced early stopping based on testing accuracy, and implemented improved monitoring. However, the results remained poor, with a test accuracy of 4.67% and persistent overfitting. [Case 5.2.4]

Finally, we increased the number of epochs, allowing training to continue even after the test loss started rising. With the setup unchanged from the first case except for setting the number of epochs to 100, the test accuracy improved to 36.40% though the overfitting issue remained unresolved. [Case 5.2.5]

Further increasing the number of epochs to 150 resulted in a test accuracy of 56.23% [Case 5.2.6], as shown in the below figure:


```
Average Test Accuracy over 150 epochs: 34.35%
```

```
Final Test Loss: 2.1449, Final Test Accuracy: 56.23%
```

Despite applying data augmentation to improve the model's generalization, overfitting remained an issue, as the testing accuracy consistently fell behind the training accuracy. While data augmentation helped introduce variations to the training data, it wasn't enough to completely address the problem. However, increasing the number of epochs after applying the data augmentation showed noticeable improvements in the testing accuracy. This suggests that giving the model more time to learn the augmented patterns helped to improve its performance.

3.2.2 Comparing Task 2 with Task 1 Results

In the case without data augmentation, the training loss decreases steadily, and the training accuracy quickly reaches 98.53% indicating the model is fitting well to the training data. However, the test loss stabilizes, while the testing accuracy remains stuck at 42.66%Ṫhis large gap between training and testing performance indicates significant overfitting, as the model struggles to generalize to unseen data.

The second figure shows the results with data augmentation. The training loss decreases steadily but is slightly higher due to the increased data variability. The test loss fluctuates but generally decreases, suggesting better generalization. Training accuracy rises more gradually and stays below 100% reflecting improved regularization. Most importantly, testing accuracy improves significantly to 56.23% showing better performance on unseen data and reduced overfitting.

With data augmentation, the dataset becomes larger by creating more variations of the original data. This makes it harder for the model to memorize specific details, reducing overfitting and helping it generalize better to new data. However, despite the improvements, there is still a noticeable gap between training and testing accuracy, indicating that overfitting has not been completely resolved.

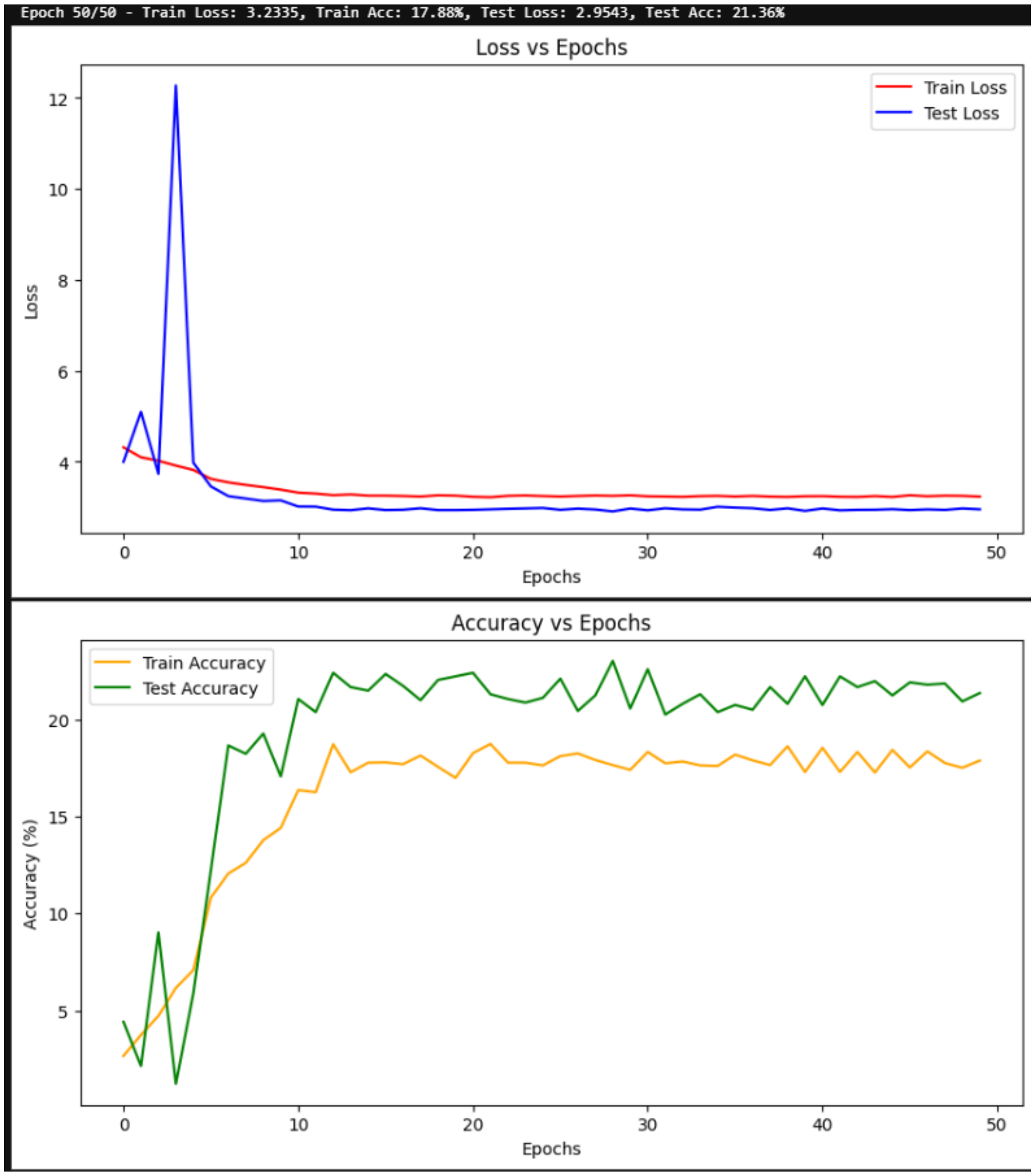
3.3 Task 3

3.3.1 Using Published CNN Network

In this task, we initially chose an untrained VGG-16 model because it is similar to our custom CNN. Both models use the ReLU activation function and incorporate max pooling layers to reduce spatial dimensions and effectively extract hierarchical features. However, the results were unexpected, with the accuracy remaining fixed at 1.23% across all epochs. As a result, we decided to experiment with another architecture, ResNet.

We trained and evaluated the ResNet-18 model on the AHAWP dataset, customizing the final layer to match the number of classes in our dataset (82 classes). Initially, we trained the model for 15 epochs using Cross Entropy Loss and the Adam optimizer, while tracking training and validation accuracy and loss. Unfortunately, the accuracy was very low, around 2%. To address this, we increased the number of epochs to 50, which improved the accuracy to approximately 20%. Despite these efforts, we were unable to achieve 21.36% accuracy within the scope of this task.

The figure below illustrates the accuracy and loss trends across epochs



```
Epoch 50/50 - Train Loss: 3.2335, Train Acc: 17.88%, Test Loss: 2.9543, Test Acc: 21.36%
```

3.3.2 Comparing Task 3 with Task 2 Results

In the case with data augmentation, the training loss steadily decreases over 150 epochs, while the test loss varies but gradually decreases overall, indicating improved generalization due to the diverse variations introduced by augmentation. The training accuracy increases gradually and stays below 100% reflecting better regularization. Meanwhile, the testing accuracy steadily improves, reaching 56.23% showing better performance on unseen data compared to models without augmentation.

In the case with ResNet18 trained from scratch, the training loss decreases steadily but stabilizes at a higher value, while the test loss drops initially and stabilizes with minimal variation, indicating limited generalization. Training accuracy remains consistently lower than testing accuracy, suggesting underfitting or a mismatch with the dataset. The testing accuracy peaks at 21.36% far below the 56.23% achieved with data augmentation.

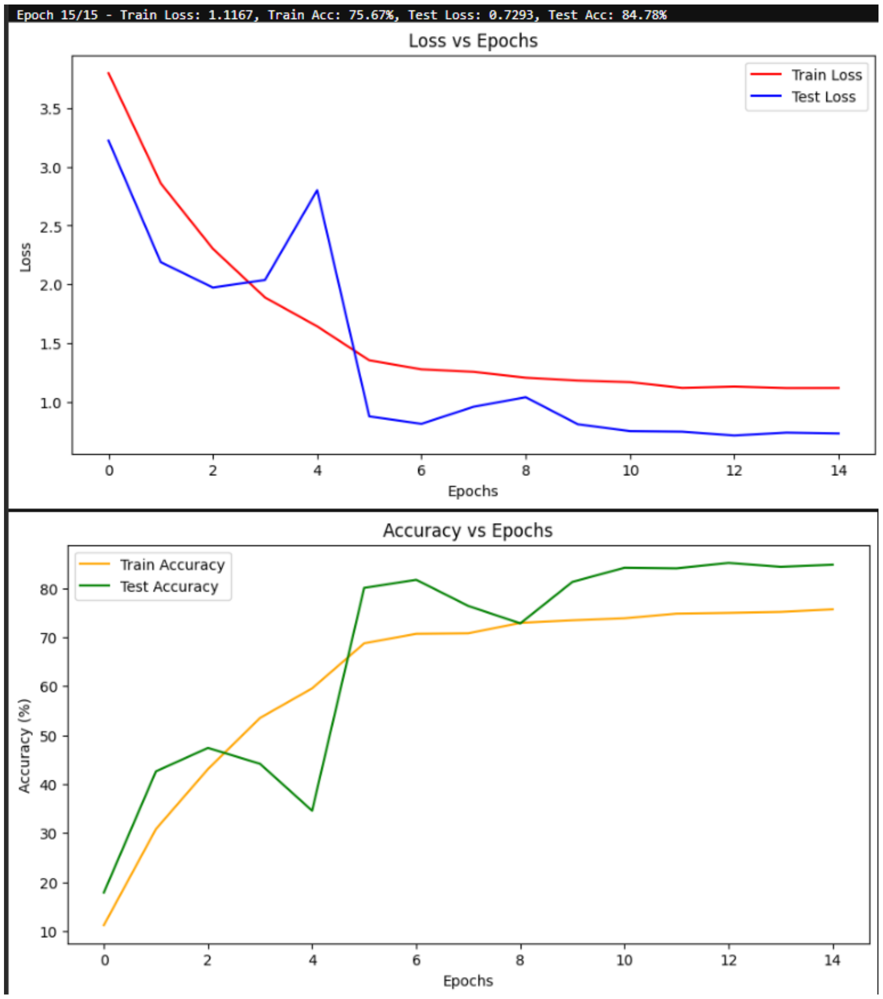
Overall, the model trained with data augmentation performs much better, showing superior generalization and accuracy. The published network, while stable, underperforms due to its insufficient complexity or lack of alignment with the dataset.

3.4 Task 4

3.4.1 Using Pre-trained CNN Network

In this task, we also used the ResNet18 architecture, initializing it with pre-trained weights from the ImageNet dataset. We then fine-tuned the network on our specific dataset, adjusting the final fully connected layer to match the number of classes in our classification task. This approach resulted in a test accuracy of 71.52% with no signs of overfitting.

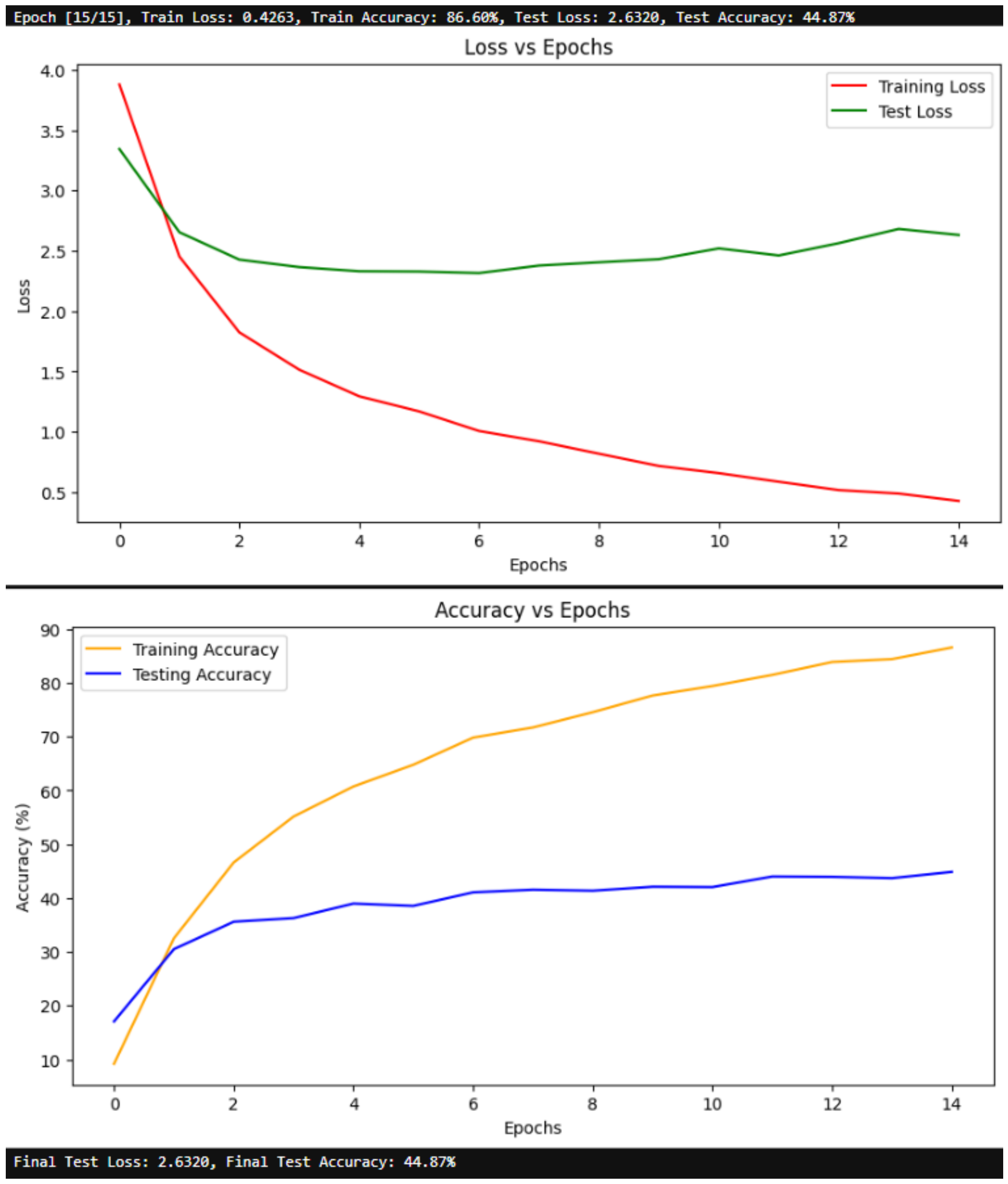
Subsequently, we made minor adjustments to the architecture to improve performance further. First, we reduced the learning rate from 0.001 to 0.0001, which increased the test accuracy to 83.92%Ȧdditionally, we incorporated weight decay to enhance regularization, and got test accuracy equal to 84.78% with improved performance on unseen data. The figure below shows the results of this final approach:



```
Epoch 15/15 - Train Loss: 1.1167, Train Acc: 75.67%, Test Loss: 0.7293, Test Acc: 84.78%
```

Additionally, we searched for models trained on tasks similar to ours and found a paper presenting a model specifically designed for handwritten character recognition. The model consists of three convolutional layers with ReLU activation and max pooling to extract features from input images, followed by a flattening layer to prepare these features for further processing. It includes two fully connected layers with ReLU activation for high-level feature learning and a final output layer with 10 neurons and softmax activation for classification.

Hence, we used their code and applied transfer learning by freezing the convolutional layers of the pre-trained network, allowing only the fully connected layers to be updated during training. Here's the results:



```
Epoch [15/15], Train Loss: 0.4263, Train Accuracy: 86.60%, Test Loss: 2.6320, Test Accuracy: 44.87%
```

```
Final Test Loss: 2.6320, Final Test Accuracy: 44.87%
```

The results show that the model is learning well from the training data, as evidenced by the steadily decreasing training loss and the high training accuracy of 86.60%However, the testing accuracy levels off at 44.87% revealing a significant gap between training and testing performance, which clearly points to overfitting.

3.4.2 Comparing Task 4 with Task 3 Results

In the case with ResNet18 trained from scratch, the training loss decreases over 50 epochs but remains high at 3.2335, reflecting the model's difficulty in learning effectively. Similarly, the test loss stabilizes at 2.9543, and the testing accuracy slightly exceeds the training accuracy (21.36% vs. 17.88%, indicating that there is underfitting behavior. The model struggles to capture patterns in the data, as it has to learn all features from scratch, which is challenging given the limited data and complex architecture.

In the case with ResNet18 with pre-trained weights, the training loss steadily decreases to 1.1167 over 15 epochs, while the test loss drops to 0.7293, indicating effective learning and generalization. The training accuracy reaches 75.67% with the testing accuracy exceeding it at 84.78% showing the model's ability to use pre-trained features for better performance on unseen data. This efficiency, achieved in fewer epochs, shows the benefits of using pre-trained weights.

The pre-trained ResNet18 outperforms the non-pre-trained version in every aspect, including accuracy (84.78% vs. 21.36%, test loss (0.7293 vs. 2.9543), and training efficiency (15 epochs vs. 50 epochs). The pre-trained model demonstrates excellent generalization due to its ability to transfer learned features from a similar dataset, while the non-pre-trained model struggles to learn meaningful representations, leading to underfitting and poor overall performance.

4 Conclusion

In conclusion, our extensive evaluation of CNN architectures for writer identification in word images highlighted the critical interplay between architectural design, regularization techniques, and data-driven strategies in achieving optimal performance. Task 1 demonstrated the iterative process of refining custom CNN models, showcasing both the potential and challenges of deeper architectures, particularly issues like overfitting. Task 2 emphasized the importance of data augmentation in improving generalization, although some overfitting persisted.

In Task 3, we trained a well-known CNN model, ResNet18, from scratch to establish a benchmark for comparing custom architectures. However, the accuracy achieved was only 20%, falling short of the desired performance and underscoring the challenges of training such models from scratch on our dataset. Finally, Task 4 explored the potential of transfer learning by leveraging a pretrained ResNet18 model. By using a model pretrained on ImageNet and referencing research papers with tasks similar to ours, we achieved significant performance improvements.

The results underscore that while custom CNNs offer flexibility and control, pretrained models and transfer learning provide a robust foundation for achieving higher accuracy and efficiency, especially with limited training data. Our custom CNN achieved 41% accuracy, which improved to 56% with data augmentation. In contrast, using the pretrained ResNet18 model boosted accuracy to 84%, and applying a pretrained ResNet model based on a relevant paper achieved 44%. These findings highlight the critical advantages of transfer learning, demonstrating that starting from scratch is often unnecessary when pretrained models are available. Future efforts could focus on more advanced augmentation techniques, architectural innovations, or ensemble approaches to further mitigate overfitting and enhance generalization across diverse datasets.

5 References

[1] Pre-trained CNN network on similar task: Handwritten Character Recognition Using Deep Learning Convolutional Neural Network

[2] Pre-trained Convolutional Neural Networks on MathWorks

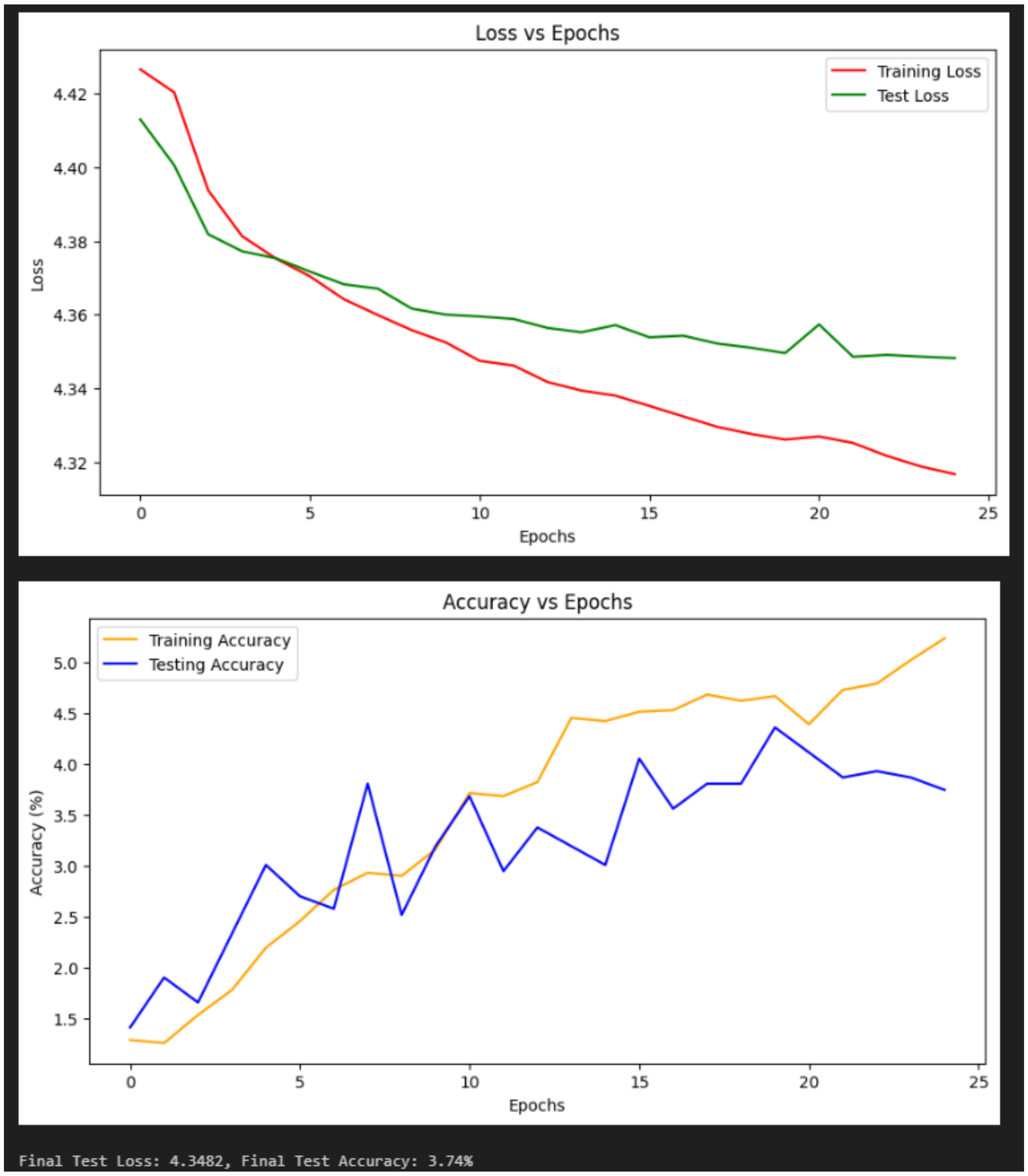
[3] ResNet Paper: Deep Residual Learning for Image Recognition

[4] Dr. Aziz Qaroush Slides Via Ritaj

6 Appendix

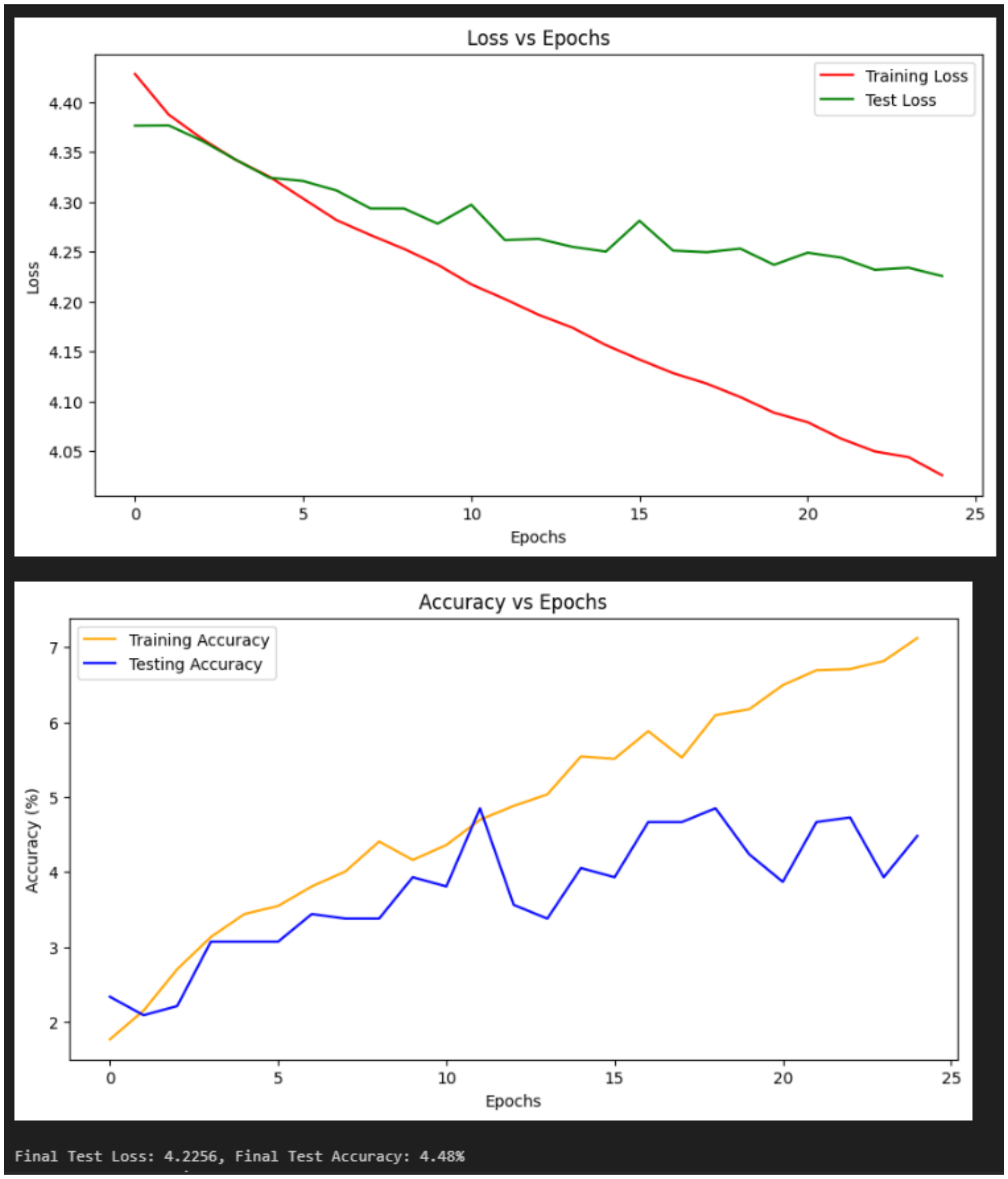
6.1 Task1

6.1.1 Case 1



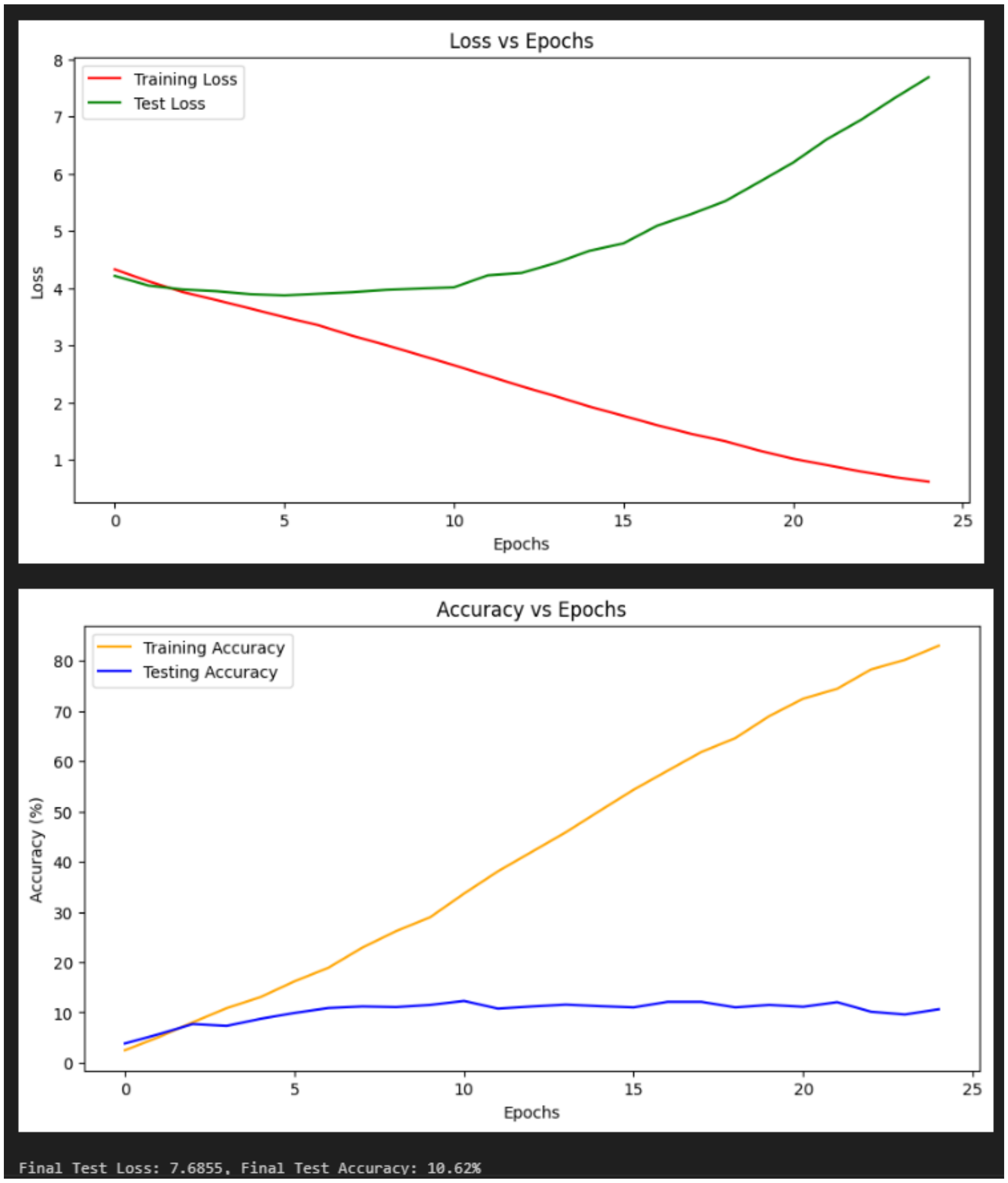
```
Final Test Loss: 4.3482, Final Test Accuracy: 3.74%
```

6.1.2 Case 2



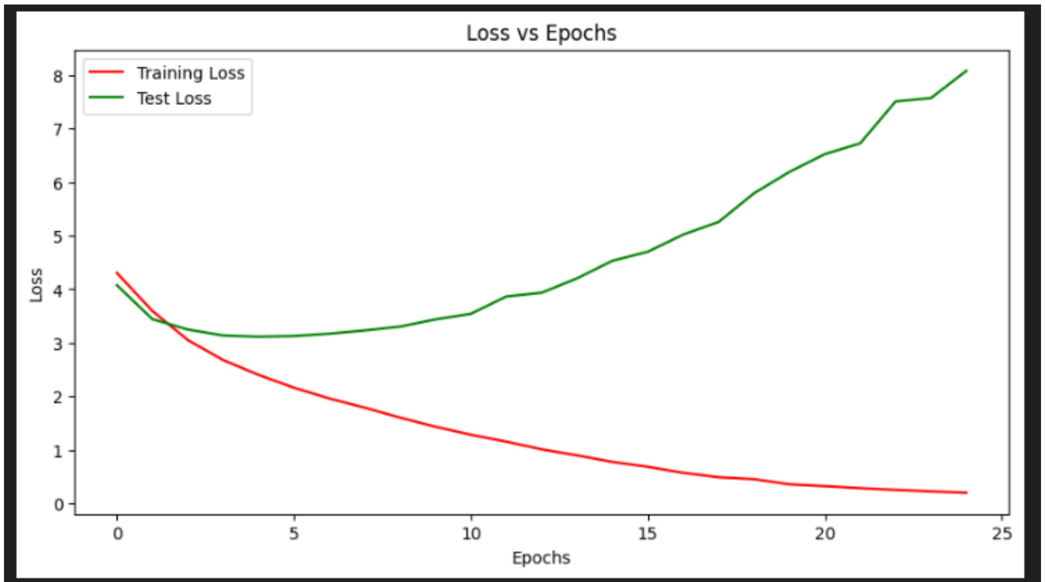
```
Final Test Loss: 4.2256, Final Test Accuracy: 4.48%
```

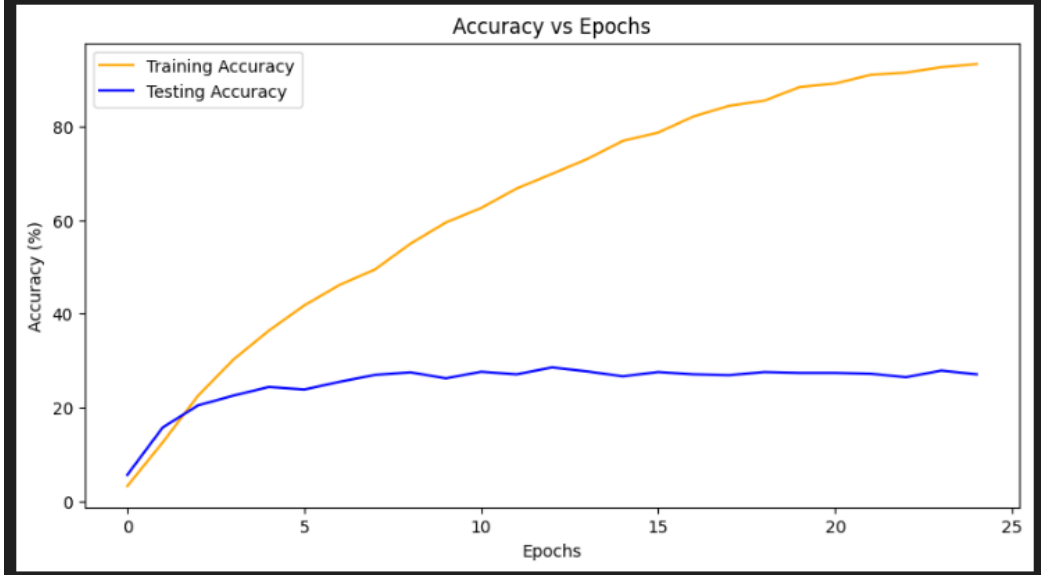
6.1.3 Case 3



```
Final Test Loss: 7.6855, Final Test Accuracy: 10.62%
```

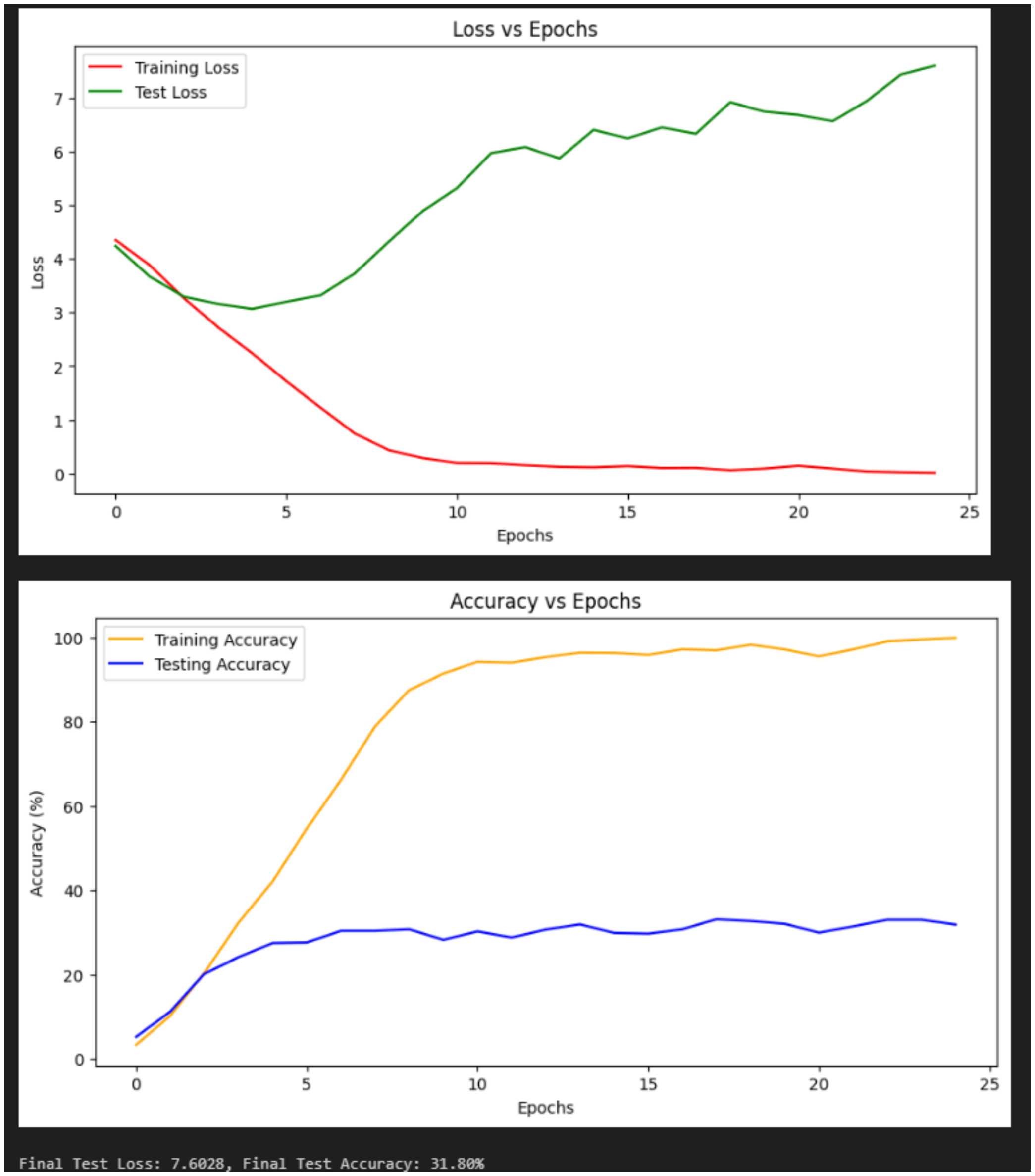
6.1.4 Case 4





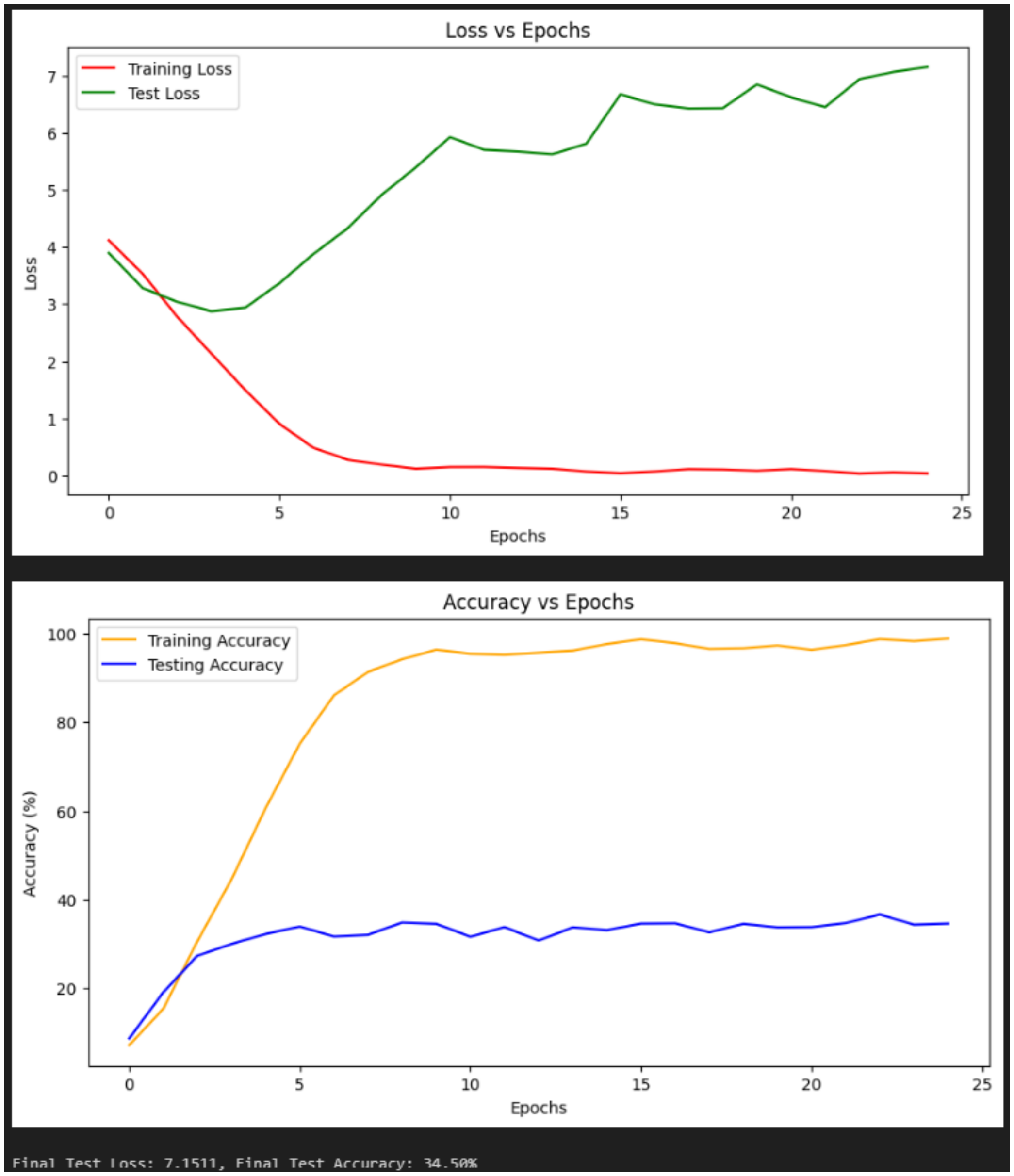
```
Final Test Loss: 8.0770, Final Test Accuracy: 27.07%
```

6.1.5 Case 5



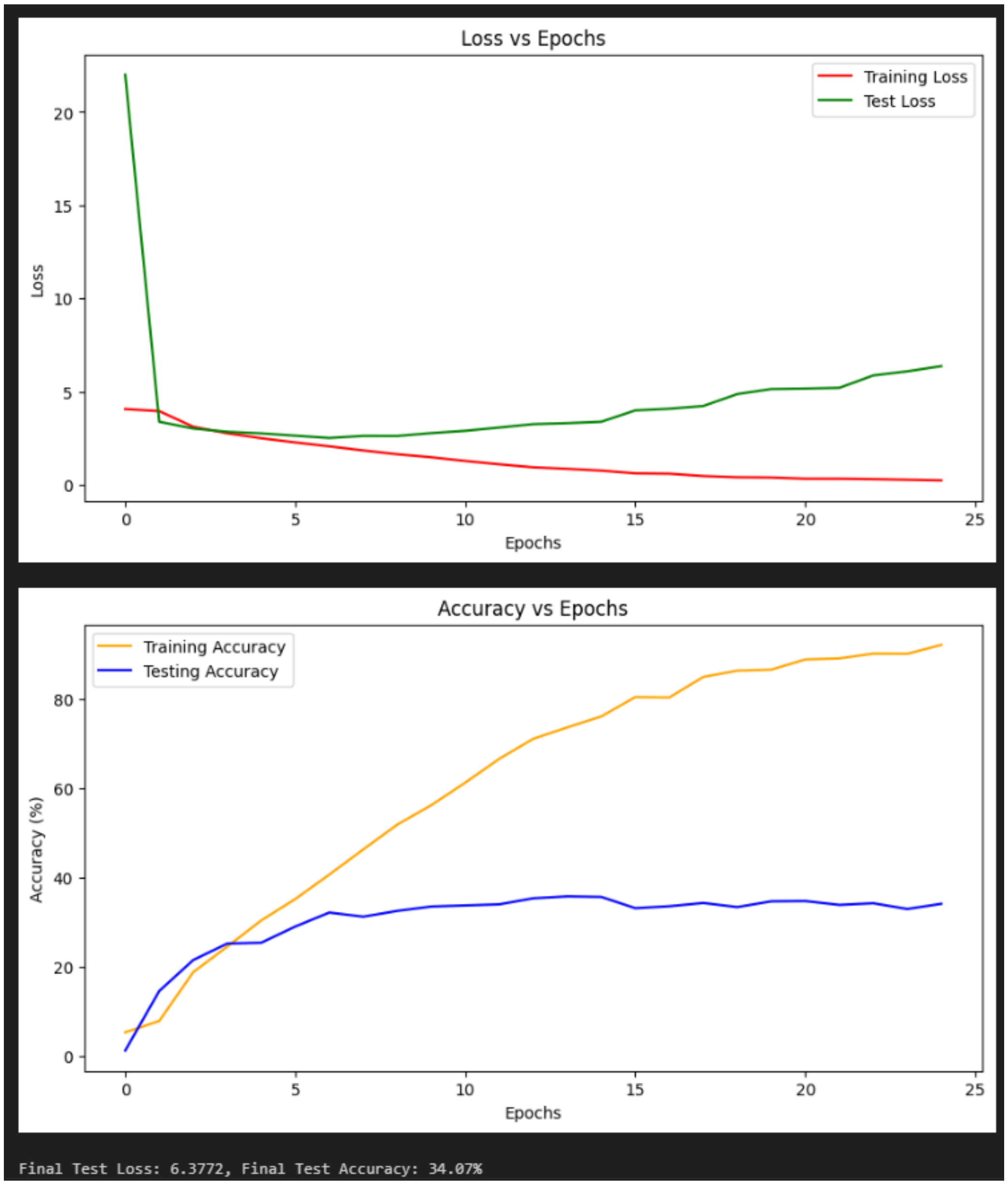
```
Final Test Loss: 7.6028, Final Test Accuracy: 31.80%
```

6.1.6 Case 6



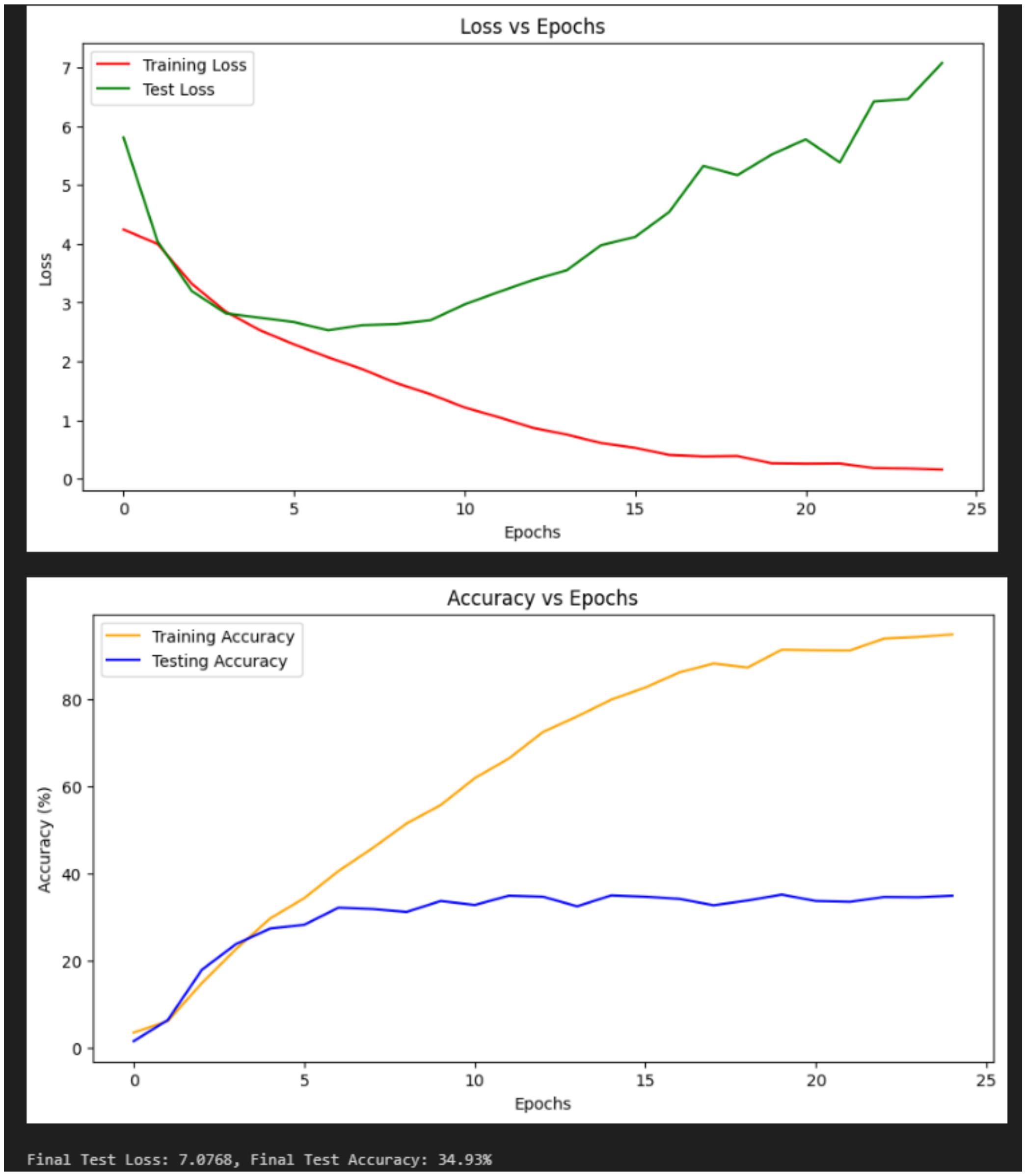
```
Final Test Loss: 7.1511, Final Test Accuracy: 34.50%
```

6.1.7 Case 7



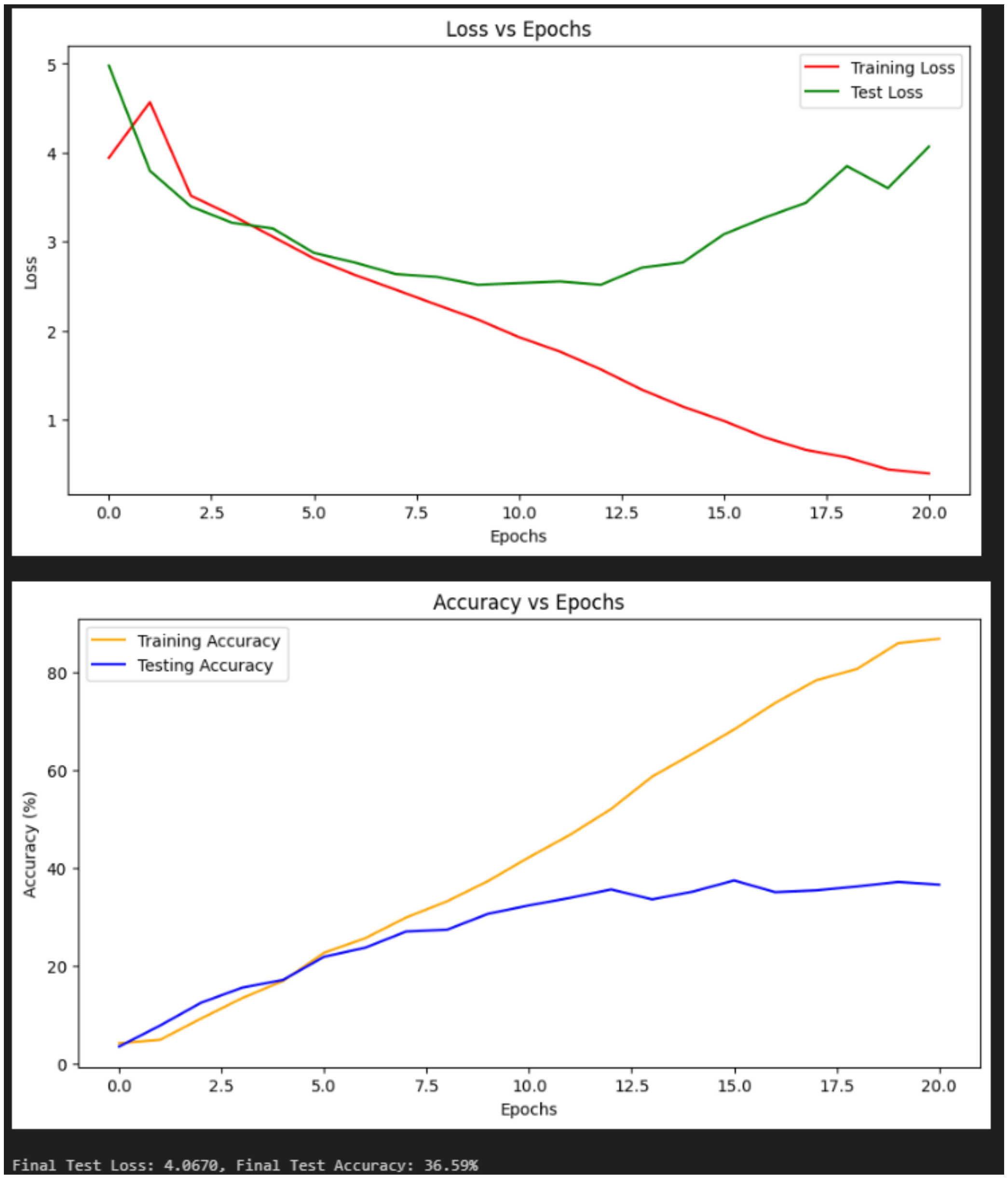
```
Final Test Loss: 6.3772, Final Test Accuracy: 34.07%
```

6.1.8 Case 8



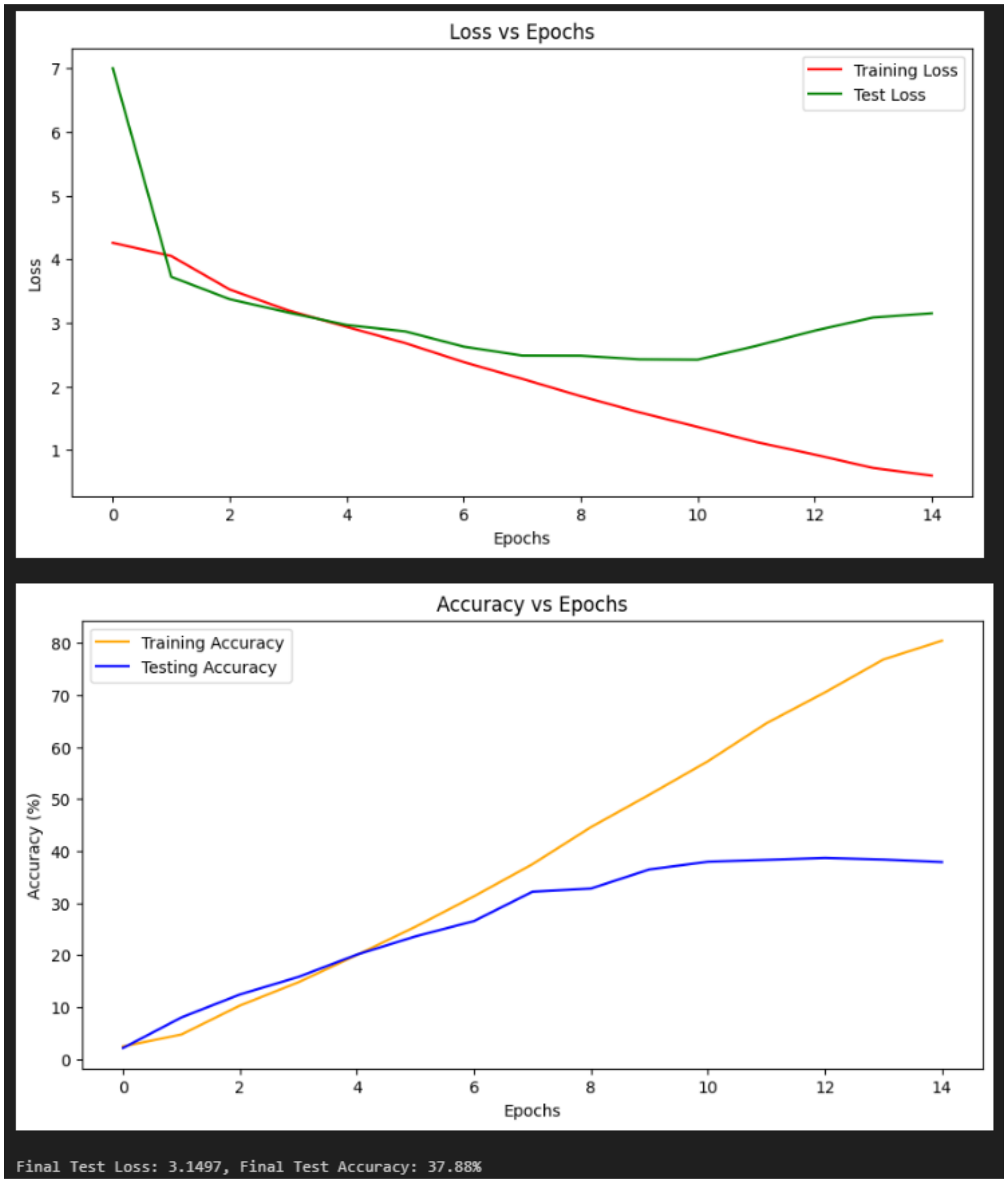
```
Final Test Loss: 7.0768, Final Test Accuracy: 34.93%
```

6.1.9 Case 9



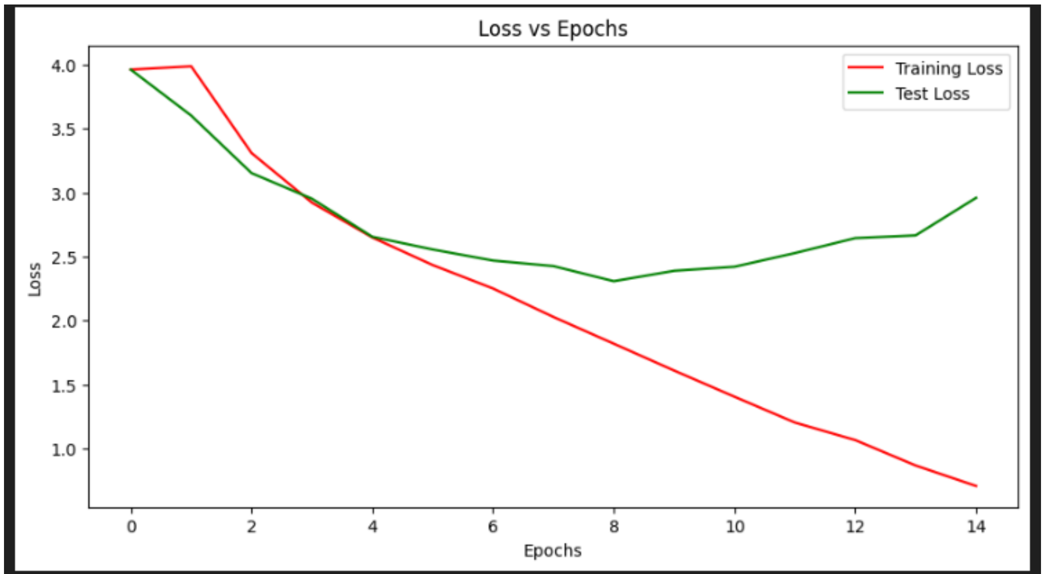
```
Final Test Loss: 4.0670, Final Test Accuracy: 36.59%
```

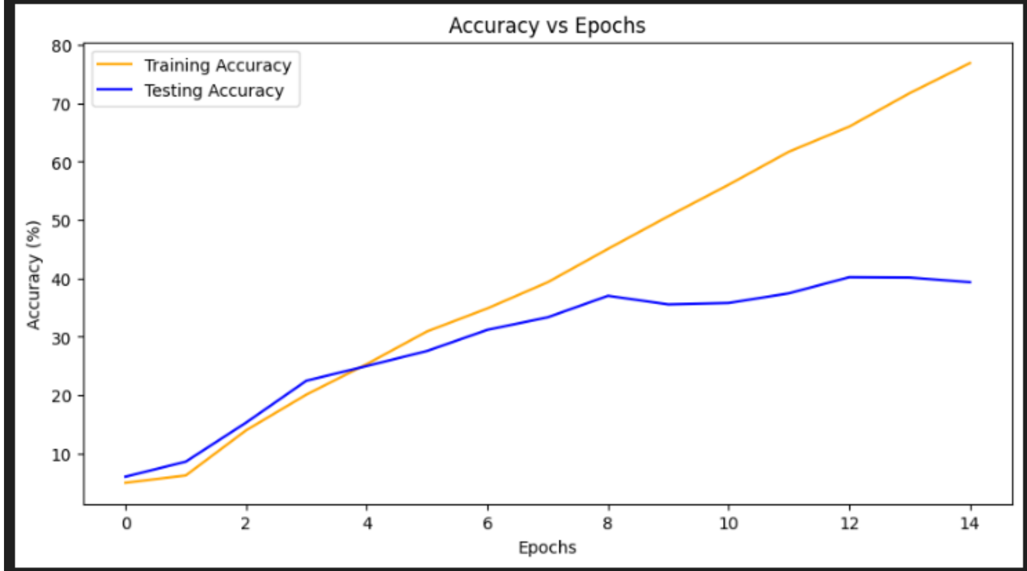
6.1.10 Case 10



```
Final Test Loss: 3.1497, Final Test Accuracy: 37.88%
```

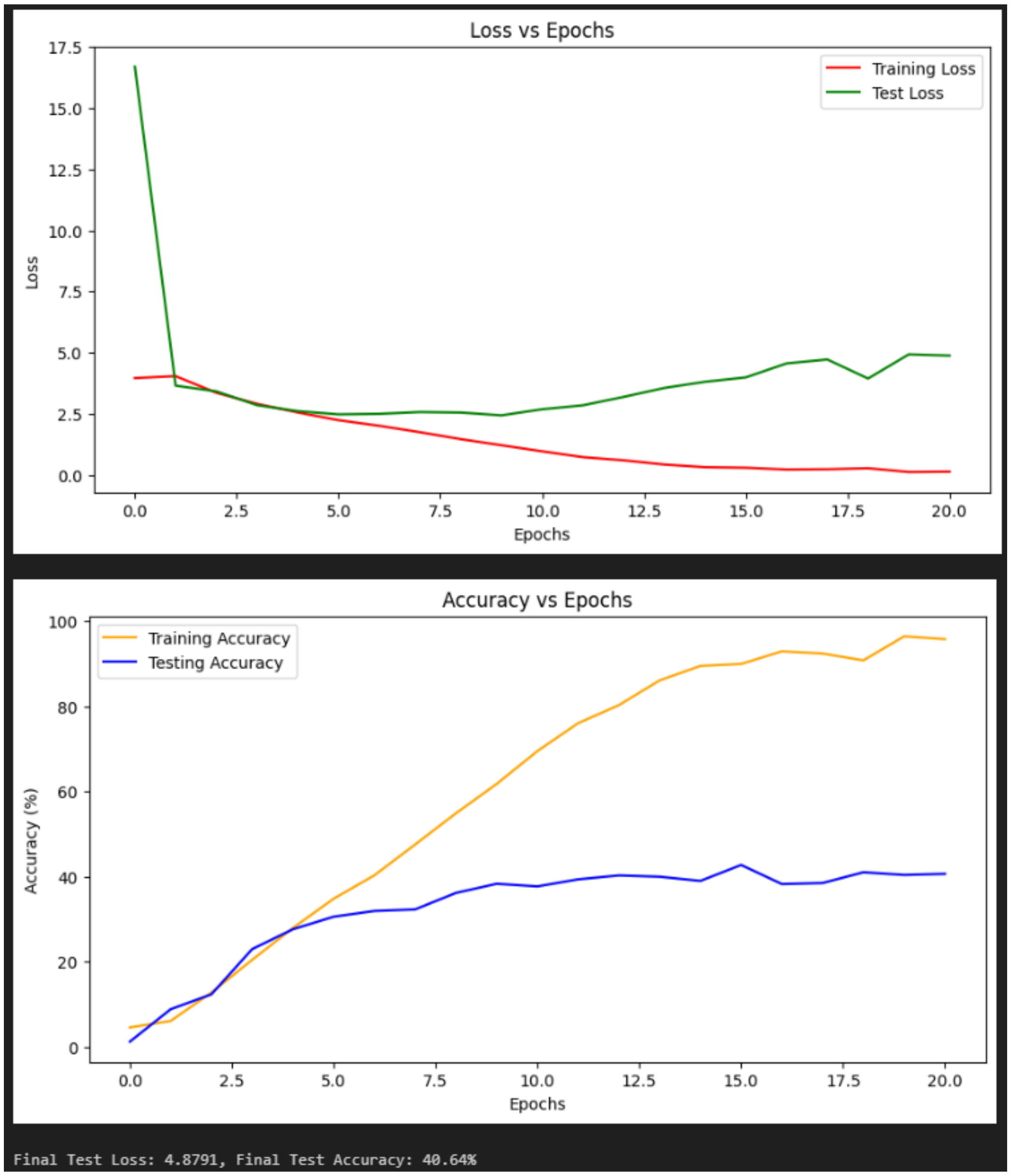
6.1.11 Case 11





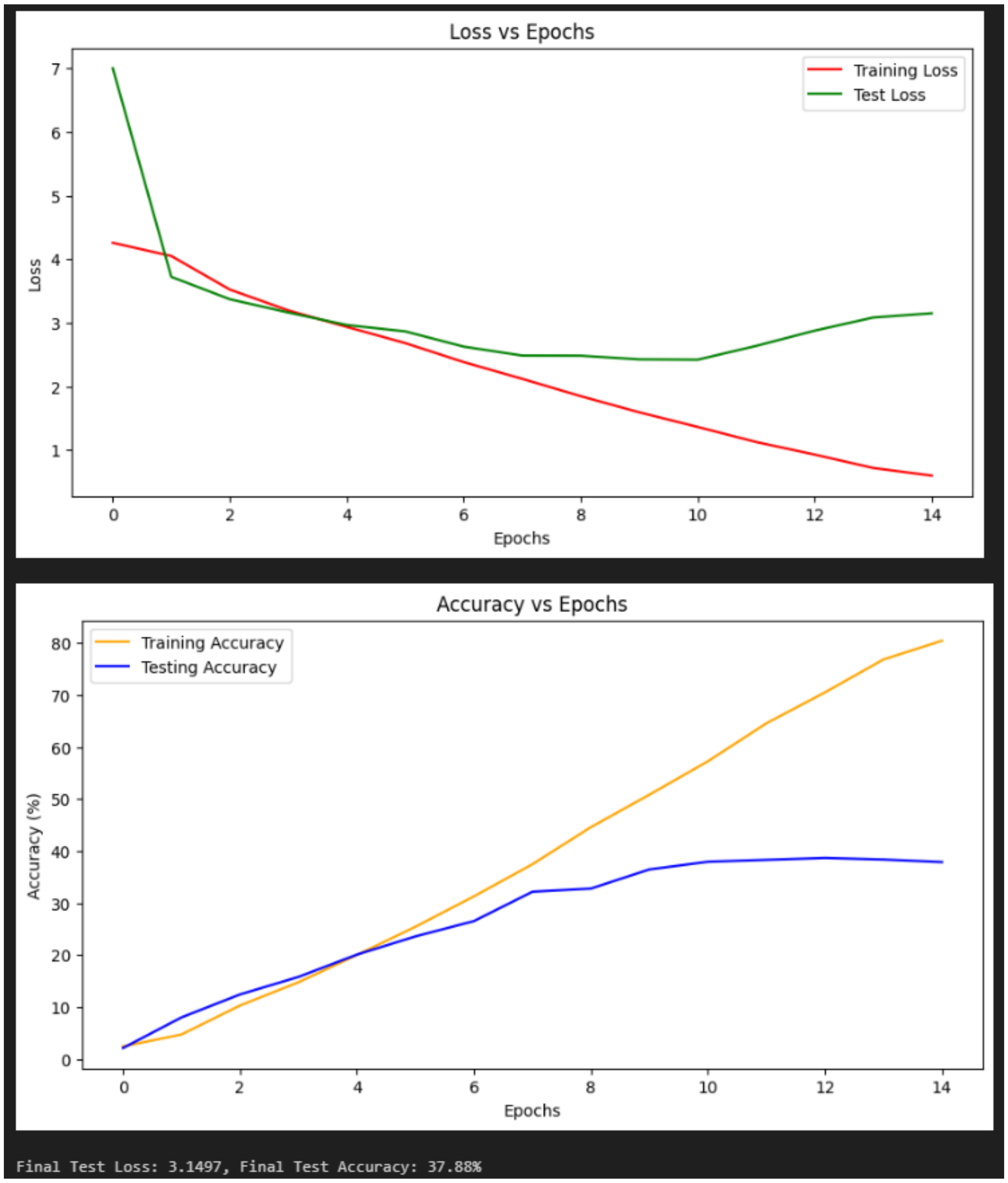
```
Final Test Loss: 2.9585, Final Test Accuracy: 39.35%
```

6.1.12 Case 12



```
Final Test Loss: 4.8791, Final Test Accuracy: 40.64%
```

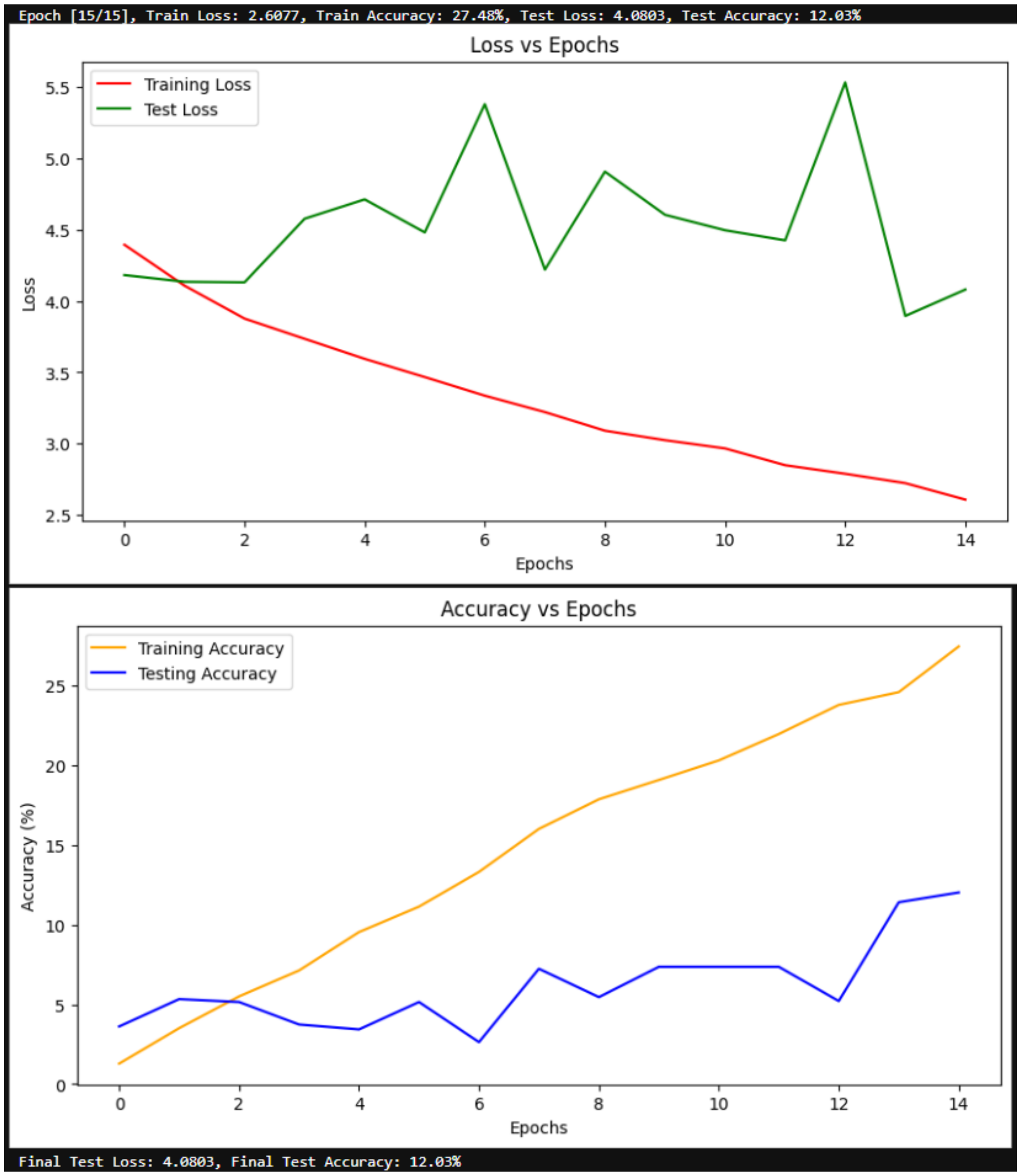
6.1.13 Case 13



```
Final Test Loss: 3.1497, Final Test Accuracy: 37.88%
```

6.2 Task2

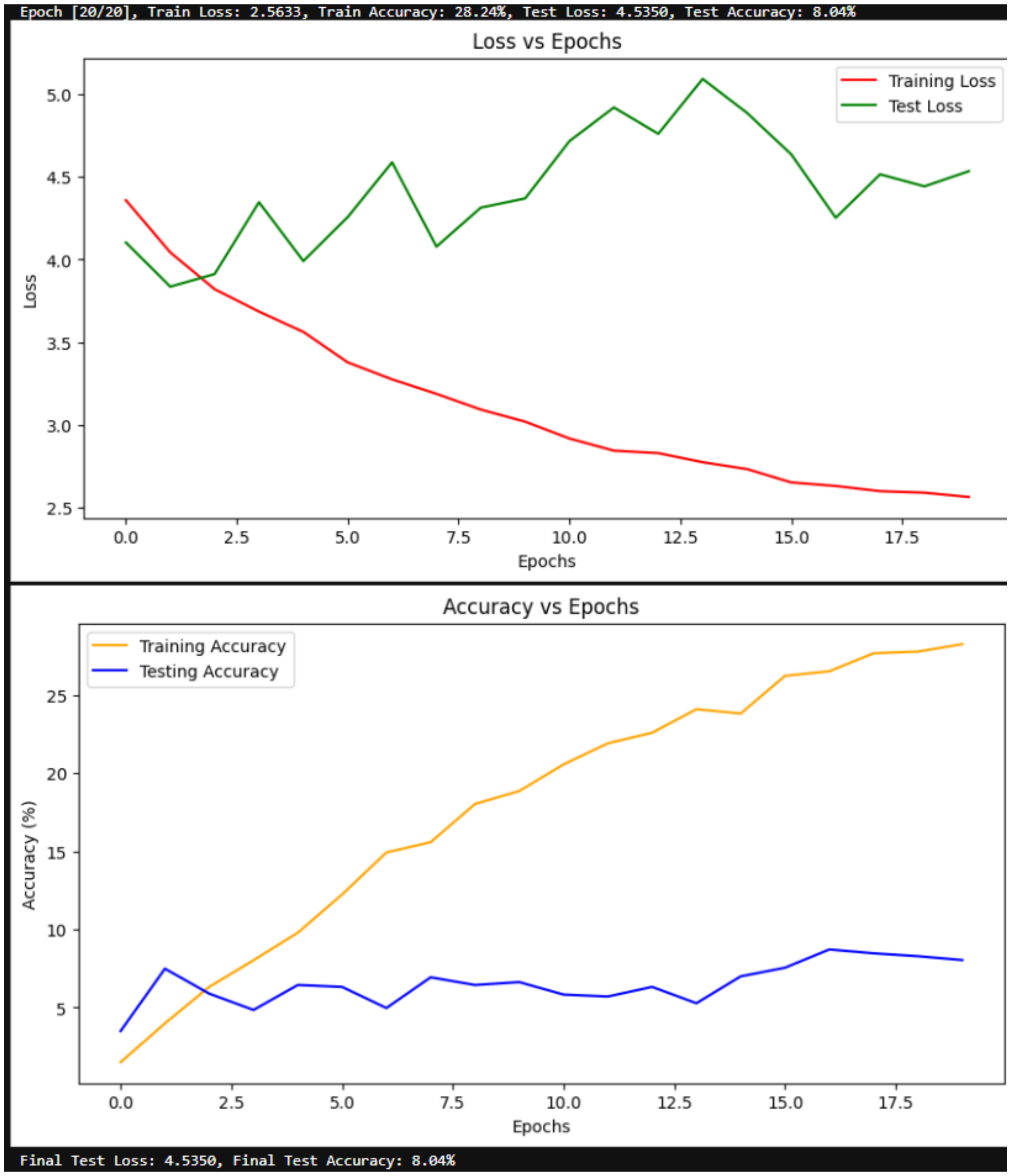
6.2.1 Case 1



```
Epoch [15/15], Train Loss: 2.6077, Train Accuracy: 27.48%, Test Loss: 4.0803, Test Accuracy: 12.03%
```

```
Final Test Loss: 4.0803, Final Test Accuracy: 12.03%
```

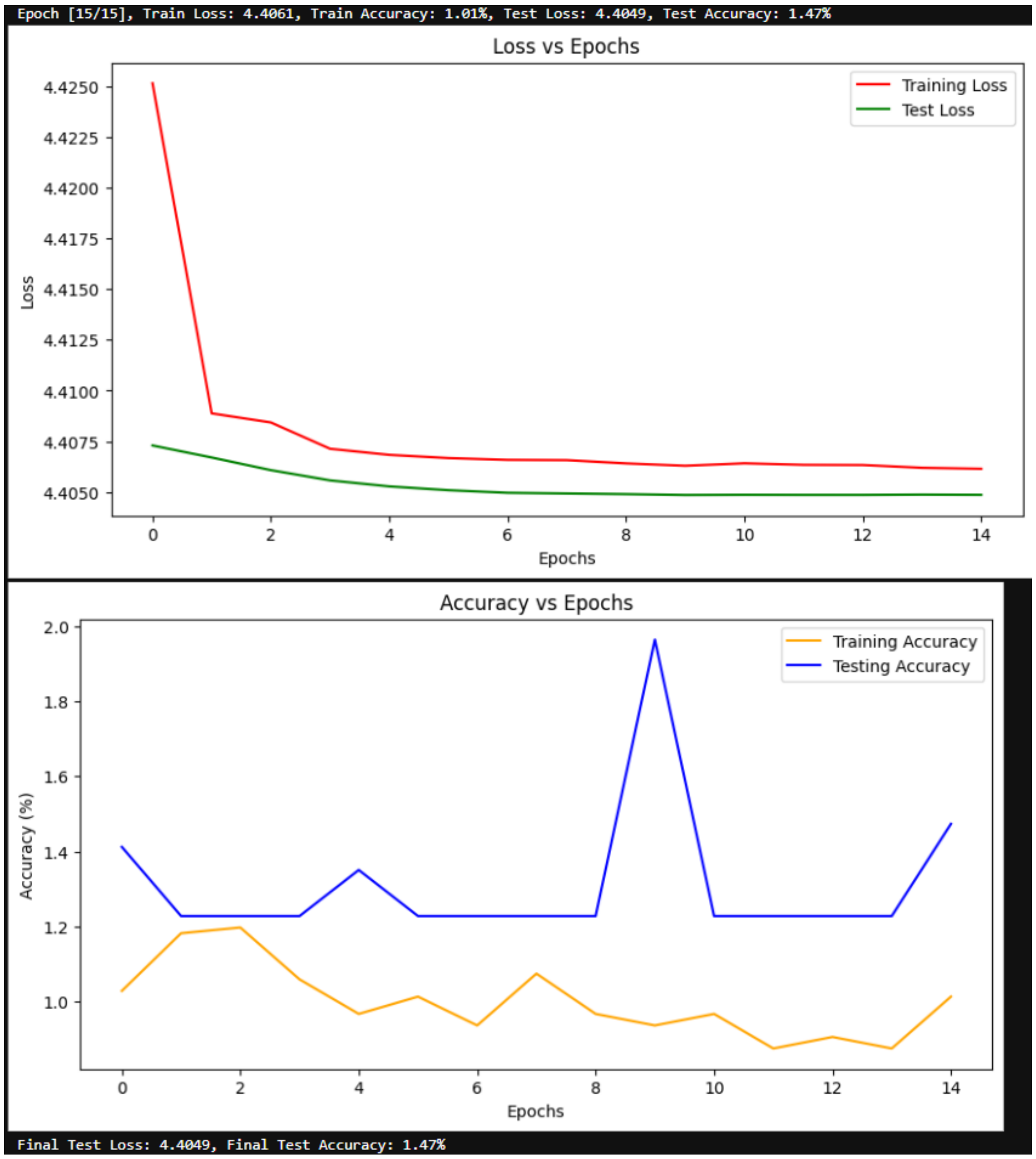
6.2.2 Case 2



```
Epoch [20/20], Train Loss: 2.5633, Train Accuracy: 28.24%, Test Loss: 4.5350, Test Accuracy: 8.04%
```

```
Final Test Loss: 4.5350, Final Test Accuracy: 8.04%
```

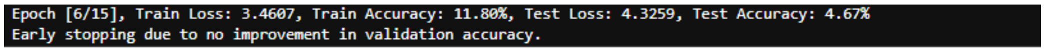
6.2.3 Case 3



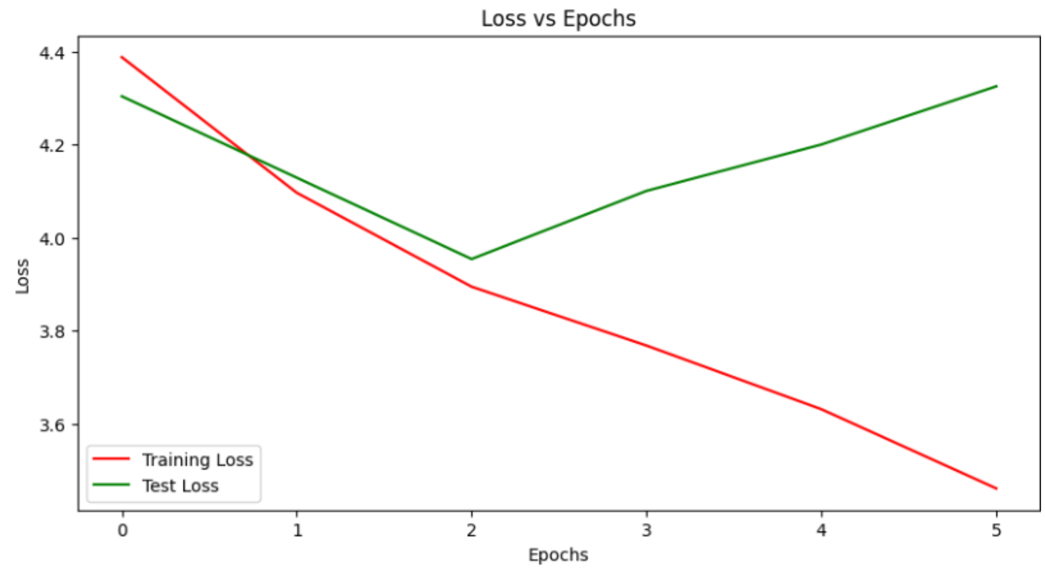
```
Epoch [15/15], Train Loss: 4.4061, Train Accuracy: 1.01%, Test Loss: 4.4049, Test Accuracy: 1.47%
```

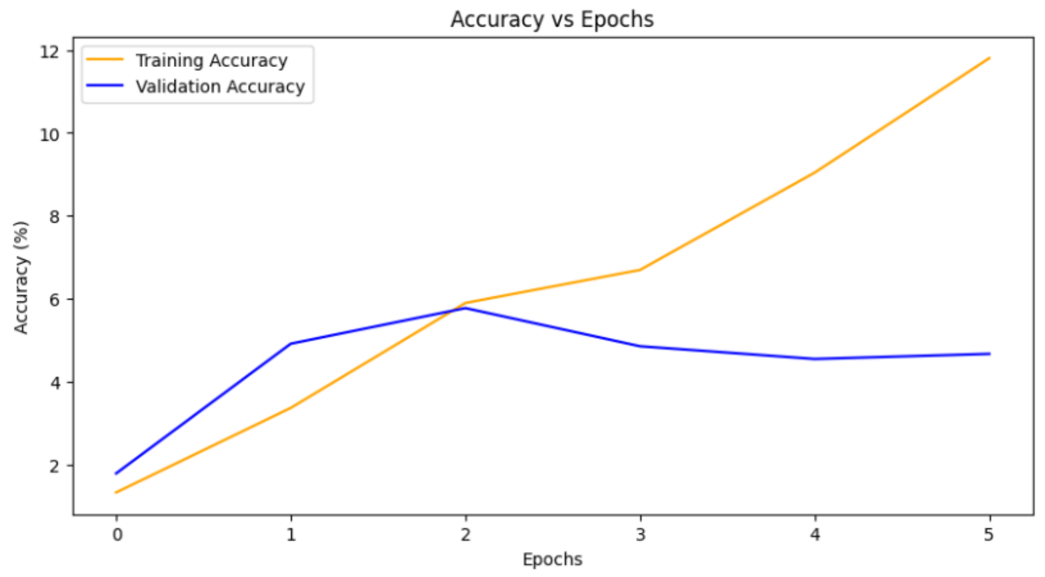
```
Final Test Loss: 4.4049, Final Test Accuracy: 1.47%
```

6.2.4 Case 4

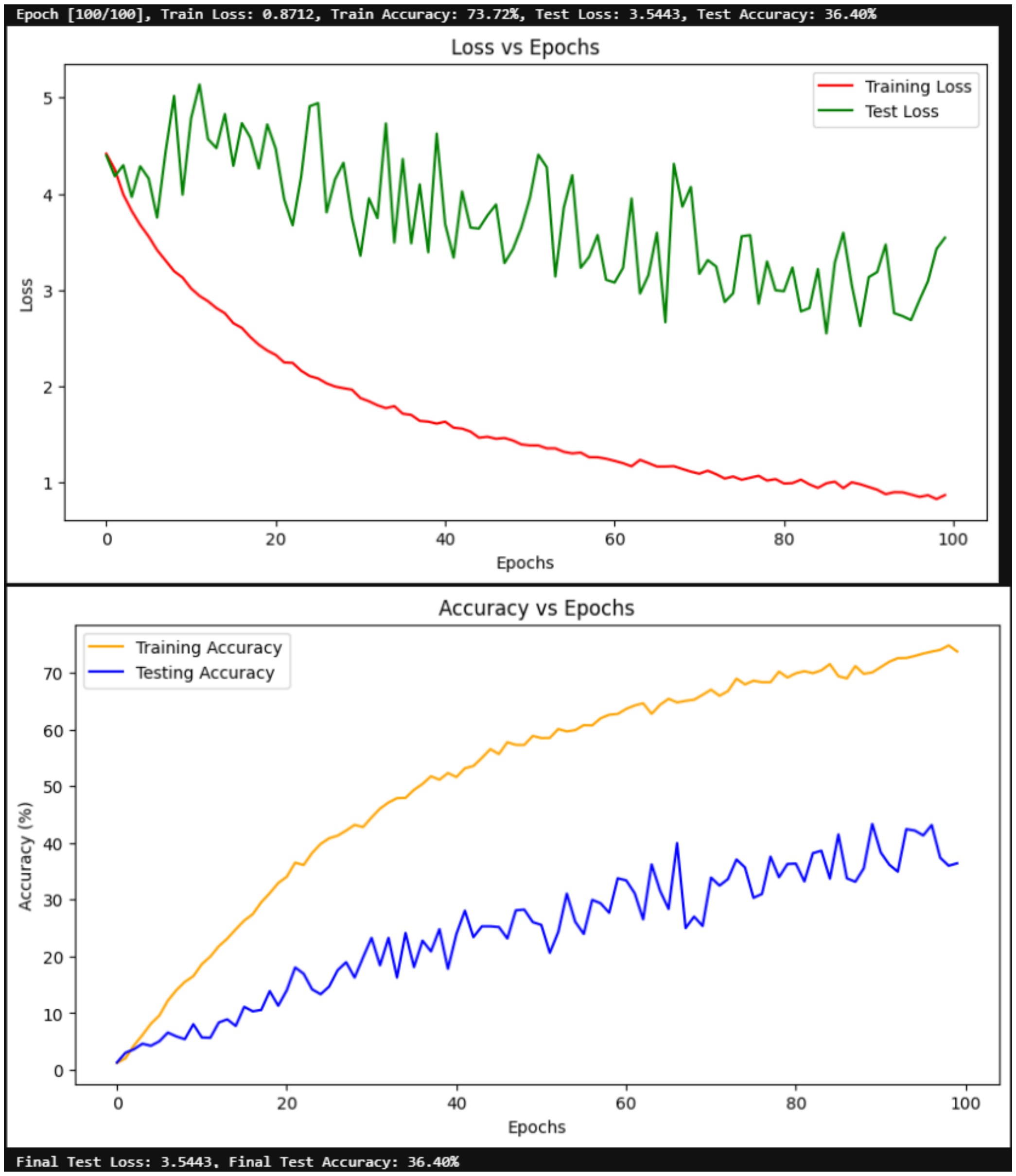


```
Epoch [6/15], Train Loss: 3.4607, Train Accuracy: 11.80%, Test Loss: 4.3259, Test Accuracy: 4.67%
Early stopping due to no improvement in validation accuracy.
```



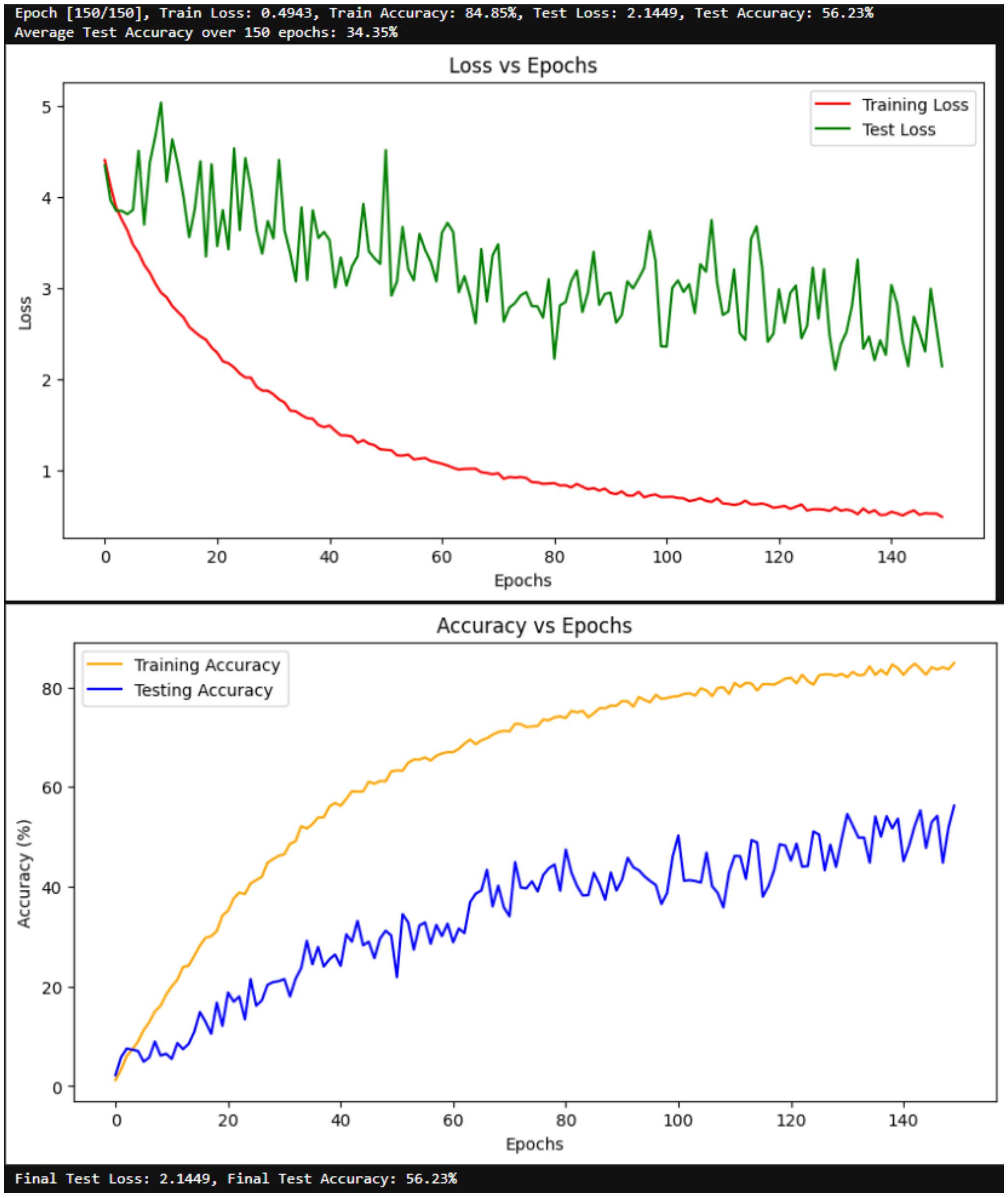


6.2.5 Case 5


```
Epoch [100/100], Train Loss: 0.8712, Train Accuracy: 73.72%, Test Loss: 3.5443, Test Accuracy: 36.40%
```

```
Final Test Loss: 3.5443, Final Test Accuracy: 36.40%
```

6.2.6 Case 6

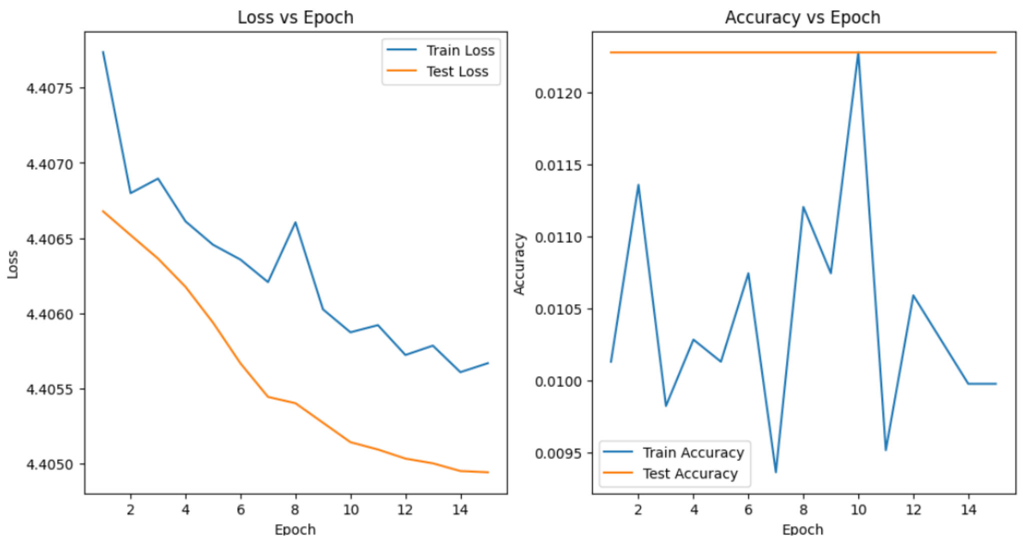


```
Epoch [150/150], Train Loss: 0.4943, Train Accuracy: 84.85%, Test Loss: 2.1449, Test Accuracy: 56.23%
Average Test Accuracy over 150 epochs: 34.35%
```

```
Final Test Loss: 2.1449, Final Test Accuracy: 56.23%
```

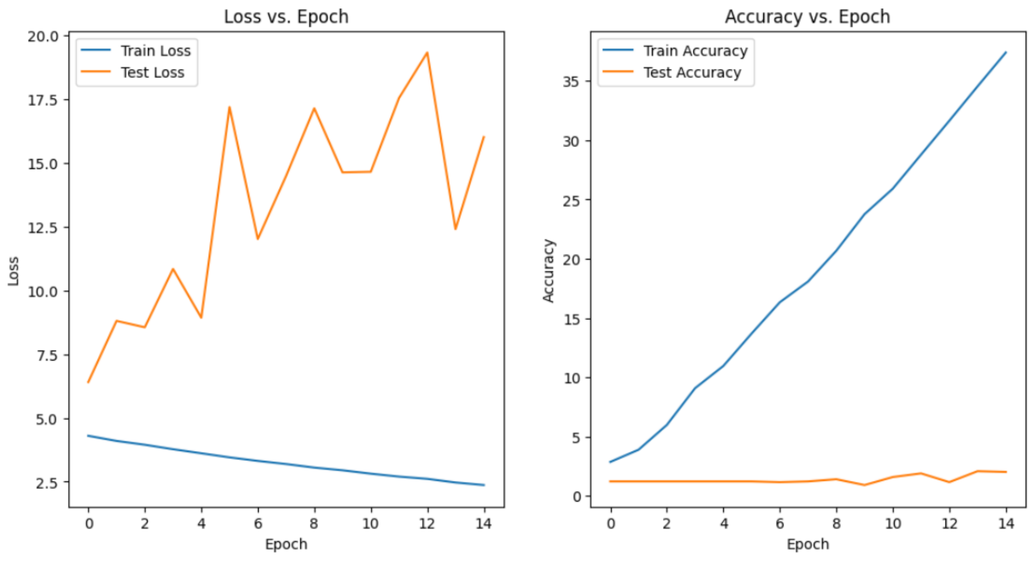
6.3 Task3

6.3.1 Case 1: VGG16 Net



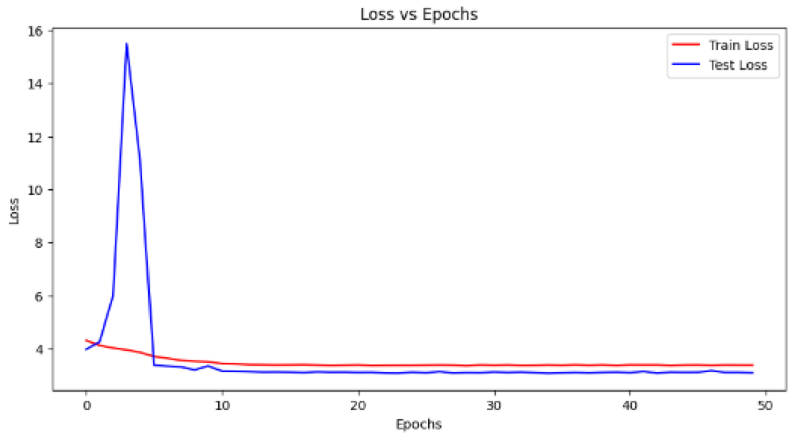
```
Epoch [1/15], Train Loss: 4.4077, Train Accuracy: 0.0101, Val Loss: 4.4067, Val Accuracy: 0.0123
Epoch [2/15], Train Loss: 4.4068, Train Accuracy: 0.0114, Val Loss: 4.4065, Val Accuracy: 0.0123
Epoch [3/15], Train Loss: 4.4069, Train Accuracy: 0.0098, Val Loss: 4.4064, Val Accuracy: 0.0123
Epoch [4/15], Train Loss: 4.4066, Train Accuracy: 0.0103, Val Loss: 4.4062, Val Accuracy: 0.0123
Epoch [5/15], Train Loss: 4.4065, Train Accuracy: 0.0101, Val Loss: 4.4059, Val Accuracy: 0.0123
Epoch [6/15], Train Loss: 4.4064, Train Accuracy: 0.0107, Val Loss: 4.4057, Val Accuracy: 0.0123
Epoch [7/15], Train Loss: 4.4062, Train Accuracy: 0.0094, Val Loss: 4.4054, Val Accuracy: 0.0123
Epoch [8/15], Train Loss: 4.4066, Train Accuracy: 0.0112, Val Loss: 4.4054, Val Accuracy: 0.0123
Epoch [9/15], Train Loss: 4.4060, Train Accuracy: 0.0107, Val Loss: 4.4053, Val Accuracy: 0.0123
Epoch [10/15], Train Loss: 4.4059, Train Accuracy: 0.0123, Val Loss: 4.4051, Val Accuracy: 0.0123
Epoch [11/15], Train Loss: 4.4059, Train Accuracy: 0.0095, Val Loss: 4.4051, Val Accuracy: 0.0123
Epoch [12/15], Train Loss: 4.4057, Train Accuracy: 0.0106, Val Loss: 4.4050, Val Accuracy: 0.0123
Epoch [13/15], Train Loss: 4.4058, Train Accuracy: 0.0103, Val Loss: 4.4050, Val Accuracy: 0.0123
Epoch [14/15], Train Loss: 4.4056, Train Accuracy: 0.0100, Val Loss: 4.4049, Val Accuracy: 0.0123
Epoch [15/15], Train Loss: 4.4057, Train Accuracy: 0.0100, Val Loss: 4.4049, Val Accuracy: 0.0123
```

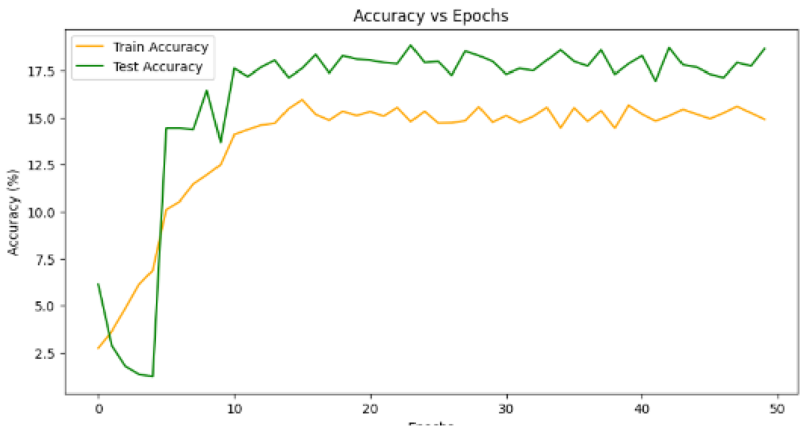
6.3.2 Case 2: ResNet18 with epochs = 15



6.3.3 Case 3: ResNet18 with epochs = 50

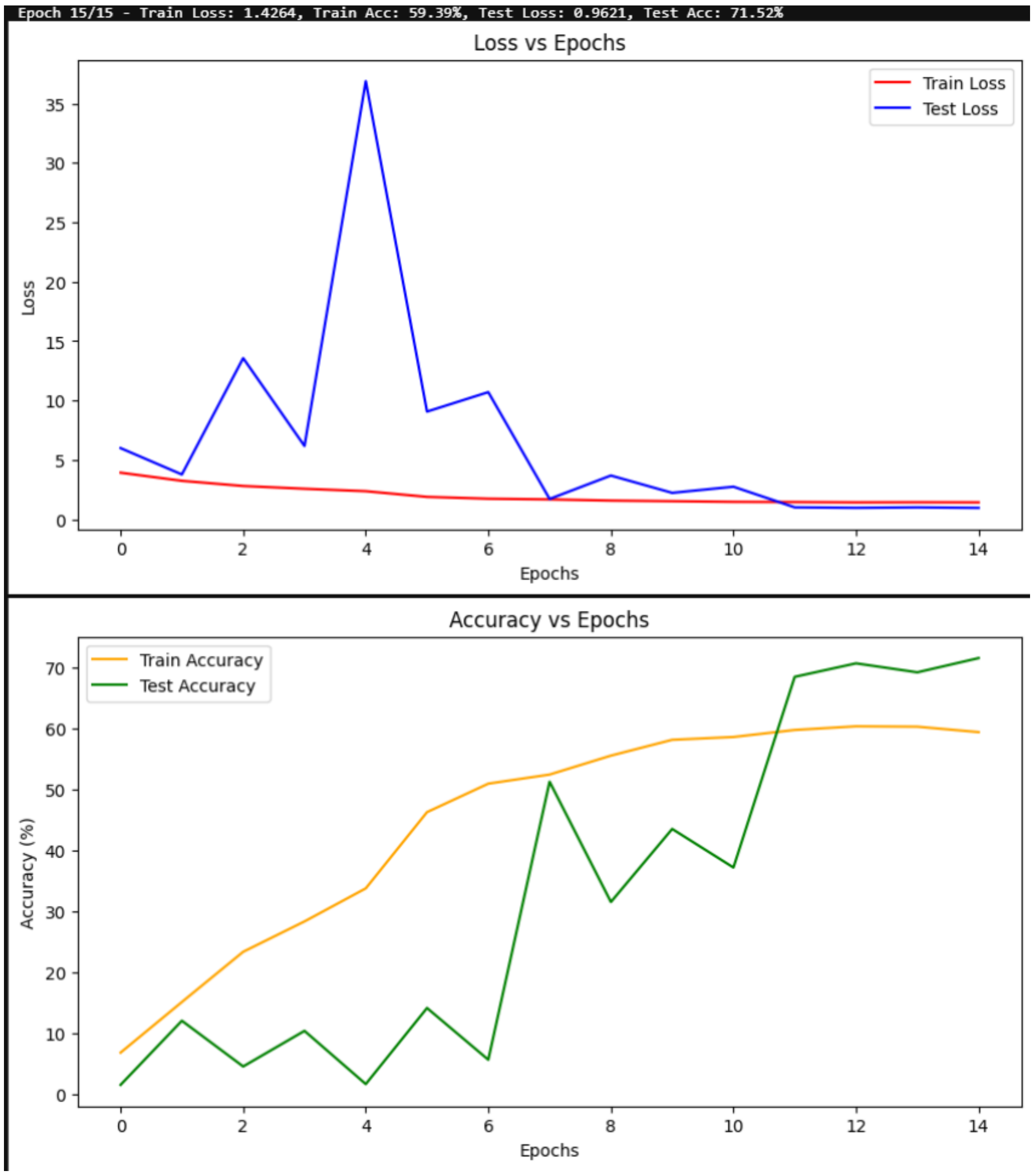
```
Epoch 50/50 - Train Loss: 3.3708, Train Acc: 14.90%, Test Loss: 3.0782, Test Acc: 18.66%
```





6.4 Task4

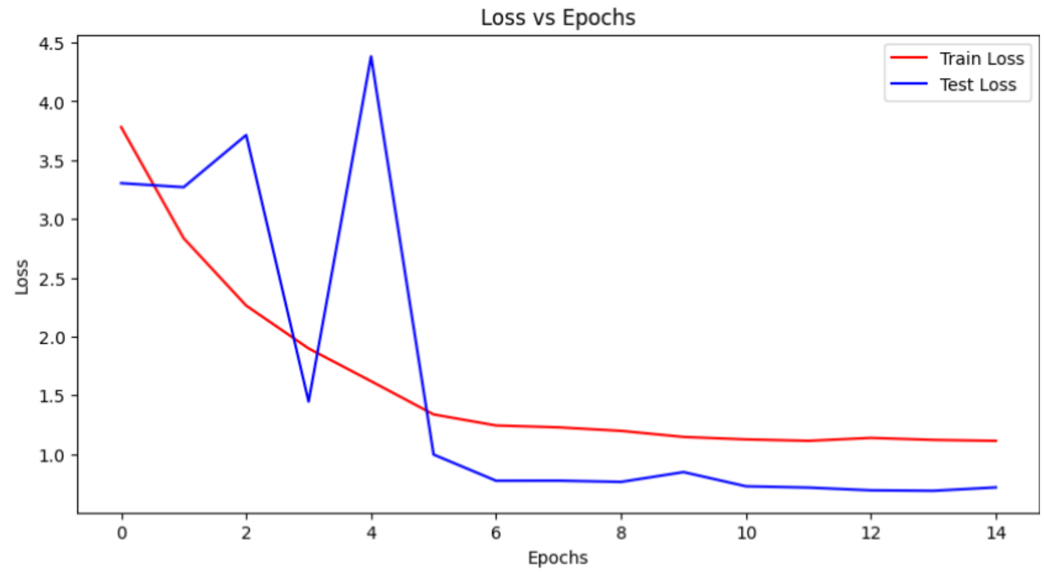
6.4.1 Case 1:

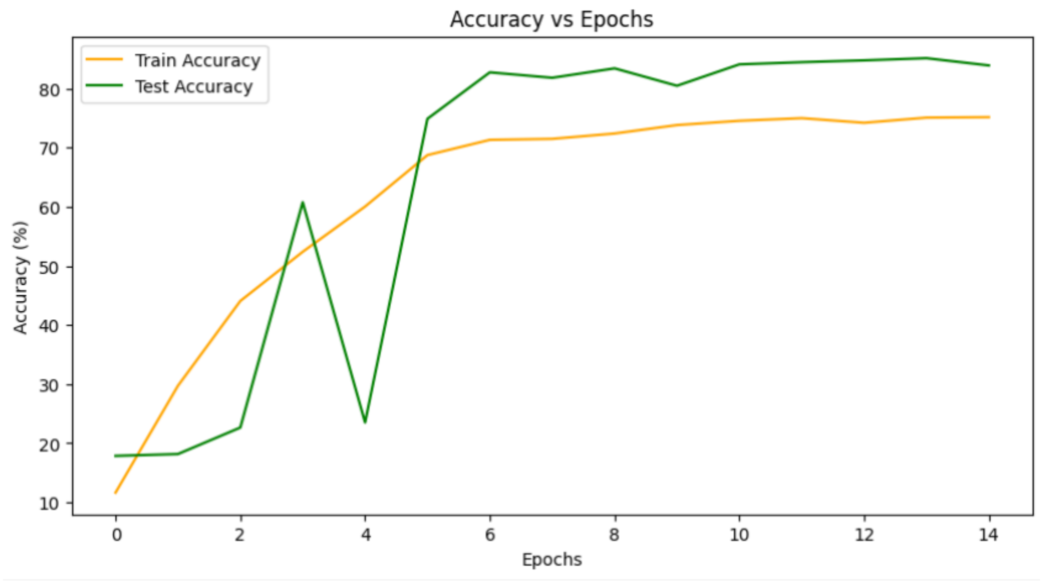


```
Epoch 15/15 - Train Loss: 1.4264, Train Acc: 59.39%, Test Loss: 0.9621, Test Acc: 71.52%
```

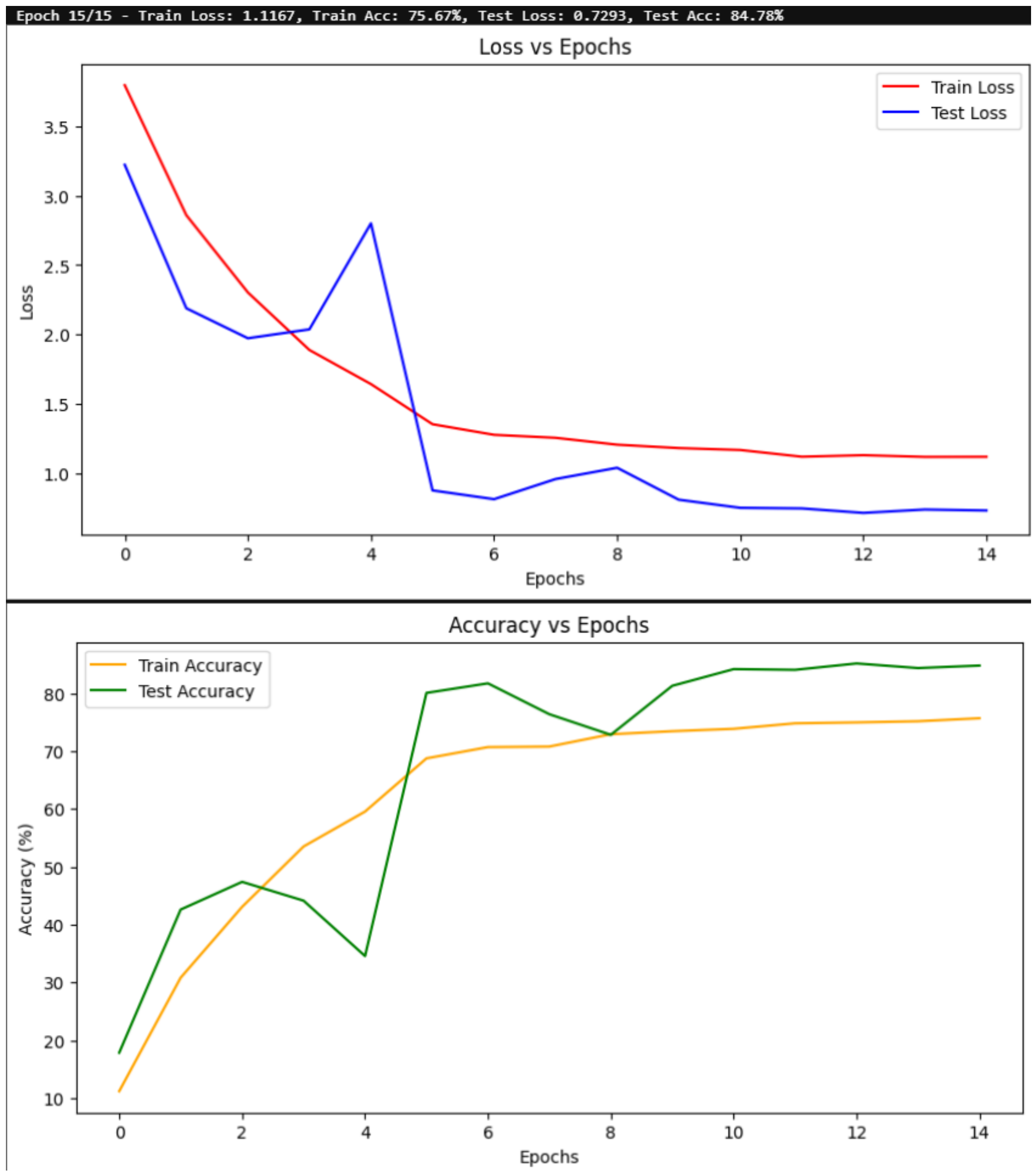
6.4.2 Case 2:

```
Epoch 15/15 - Train Loss: 1.1127, Train Acc: 75.17%, Test Loss: 0.7171, Test Acc: 83.92%
```

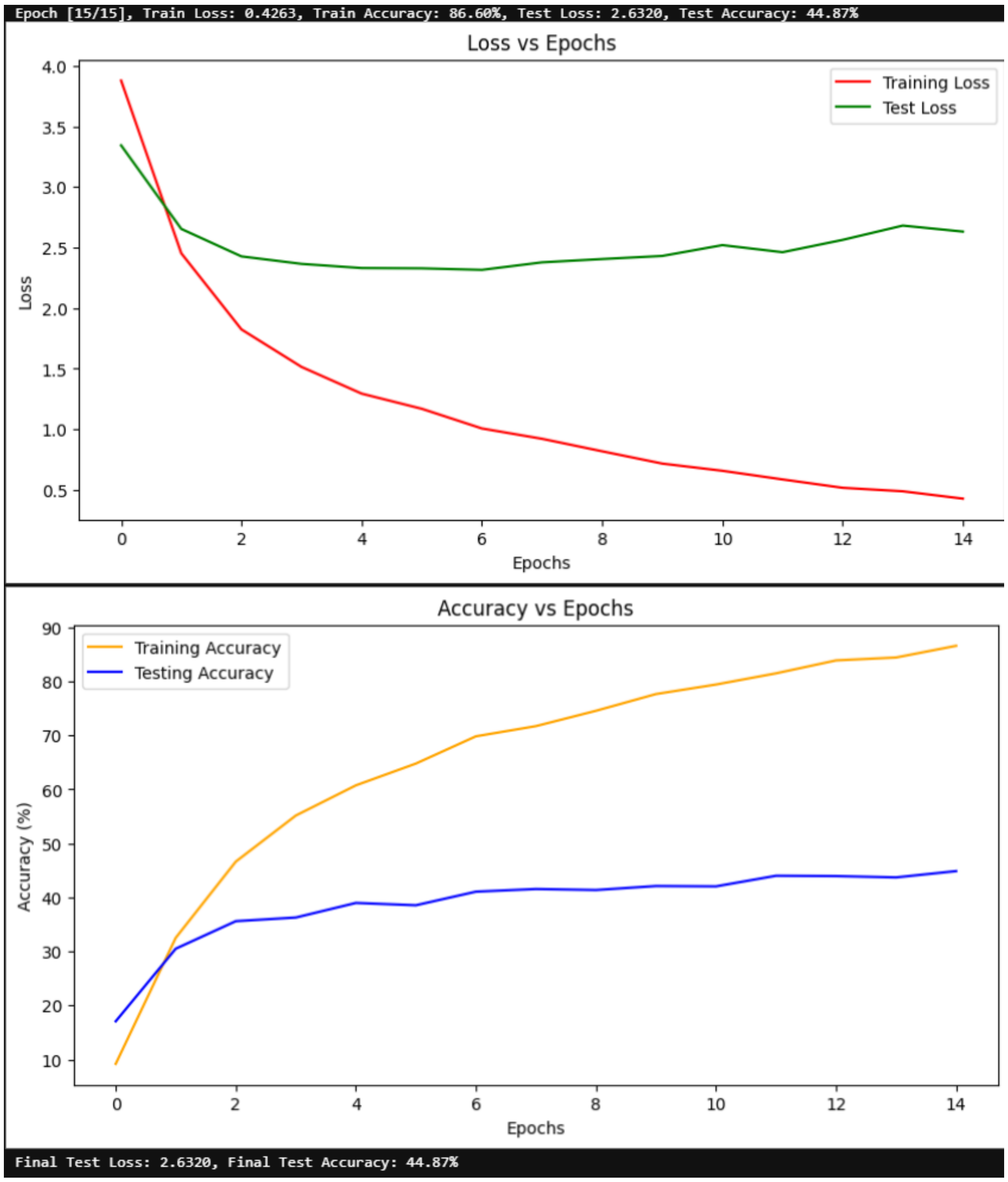




6.4.3 Case 3:


```
Epoch 15/15 - Train Loss: 1.1167, Train Acc: 75.67%, Test Loss: 0.7293, Test Acc: 84.78%
```

6.4.4 Case 4:



```
Epoch [15/15], Train Loss: 0.4263, Train Accuracy: 86.60%, Test Loss: 2.6320, Test Accuracy: 44.87%
```

```
Final Test Loss: 2.6320, Final Test Accuracy: 44.87%
```