جامعة بيرزيت
BIRZEIT UNIVERSITY

Faculty of Engineering and Technology
Department of Electrical and Computer Engineering
ENCS5141—Intelligent Systems Laboratory

Assignment #2—Comparative Analysis of CNN and Patch-based LSTM Architectures for Image Classification

Prepared by: Mohammad Salem—1200651

Instructor: Dr : Yazan Abu Farha
Assistant: Ahmad Abbas
Date: June 8, 2025

Abstract

Utilizing two benchmark datasets, MNIST and CIFAR-10, this work evaluates the classification accuracy and computational efficiency of three deep-learning architectures: a unique Patch-based LSTM, fine-tuned AlexNet (pretrained on ImageNet), and SimpleCNN (a lightweight customized CNN). Under the same setup (batch size 128, Adam optimizer, CrossEntropy loss), each model was trained for 10 epochs.

We measured four key metrics: test accuracy, total training time, total inference time, and confusion matrices.

Our results show that AlexNet attains the highest accuracy on both datasets (e.g., 99% on MNIST, 75% on CIFAR-10), while SimpleCNN offers the fastest training and inference speeds with moderate accuracy (95% on MNIST, 72% on CIFAR-10). The Patch-LSTM achieves competitive performance on MNIST (96%) but incurs longer inference times due to sequential patch processing. These findings highlight the trade-offs between model complexity, predictive performance, and computational cost, and provide guidance for selecting appropriate architectures in resource-constrained environments.

Contents

List of Figures

1 Introduction

In computer vision, classification of images is still a fundamental task, and convolutional neural networks (CNNs) are the most often used method.

However, additional sequence-based techniques have come up, such as patch-based LSTMs, which handle pictures as ordered token streams as opposed to 2D grids.

In order to determine their respective advantages across datasets of different complexity, this work compares a patch-based LSTM classifier, a pretrained AlexNet focused on target datasets, and a lightweight 2-layer CNN.

2 Methodology

2.1 Datasets

MNIST: 70,000 grayscale handwritten digit images (28×28), 10 classes.
CIFAR-10: 60,000 color images (32×32) in 10 object categories.

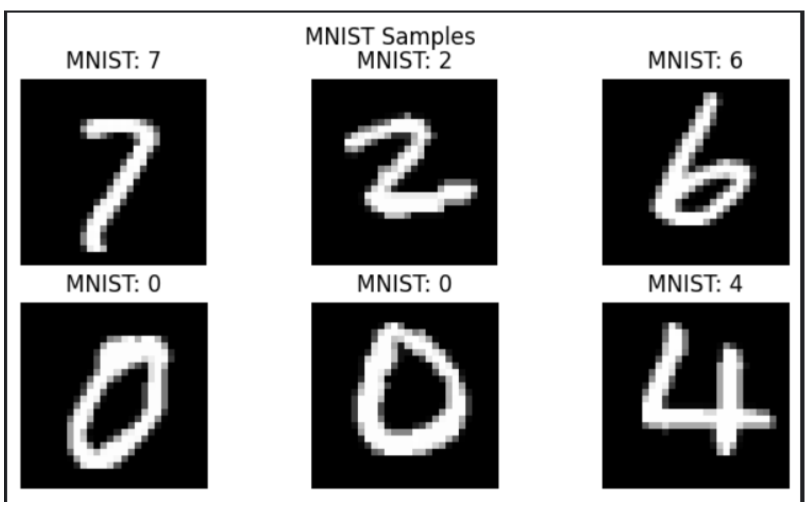


Figure 2.1: MNIST Samples

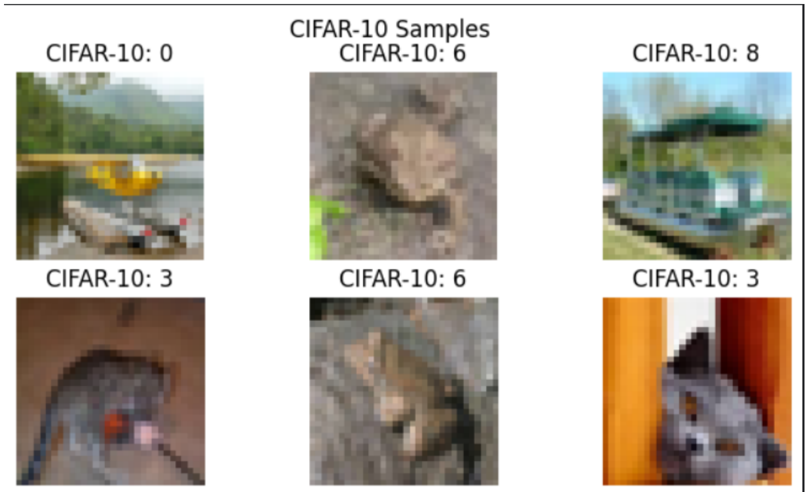


Figure 2.2: CIFAR-10 Samples

2.2 Preprocessing

Normalization: Pixel values scaled to [−1, 1] using channel-wise mean=0.5, std=0.5.

Augmentation (none): For fairness, no augmentations were applied.

Patch extraction (Architecture C): Images divided into 16 equal patches (4×4 grid); each patch flattened into a feature vector.

2.3 Model Architectures

2.3.1 SimpleCNN (Architecture A)

A custom CNN with two 2D convolutional layers (32→64 filters), ReLU activations, max-pooling, followed by a two-layer MLP head for classification.

```
MNIST CNN: SimpleCNN(
  (features): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=3136, out_features=128, bias=True)
    (2): ReLU()
    (3): Linear(in_features=128, out_features=10, bias=True)
  )
)
CIFAR CNN: SimpleCNN(
  (features): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=4096, out_features=128, bias=True)
    (2): ReLU()
    (3): Linear(in_features=128, out_features=10, bias=True)
  )
)
Output shape MNIST: torch.Size([1, 10])
Output shape CIFAR: torch.Size([1, 10])
```

Figure 2.3: Architecture A

2.3.2 Fine-tuned AlexNet (Architecture B)

Pretrained AlexNet on ImageNet; feature extractor layers frozen, classifier head replaced with a 10-way linear layer, trained for 10 epochs.

2.3.3 Patch-based LSTM (Architecture C)

Transforms each image into a sequence of patch tokens, processes them through a single-layer LSTM (hidden size=128), and applies a dense+softmax classifier to the final hidden state.

2.4 Experimental Setup

Framework: PyTorch on GPU-enabled Kaggle.

Hyperparameters:

- Batch size: 128
- Epochs: 10
- Optimizer: Adam (lr=1e-3)
- Loss function: CrossEntropyLoss

Evaluation Metrics:
- Accuracy,
- Total training time,
- Total inference time,
- Confusion matrix.

```
Epoch 1/10  Train: loss=0.1281, acc=0.9565  Test:  loss=1.4115, acc=0.7086
Epoch 2/10  Train: loss=0.0906, acc=0.9709  Test:  loss=1.5158, acc=0.7057
Epoch 3/10  Train: loss=0.0835, acc=0.9727  Test:  loss=1.6511, acc=0.7063
Epoch 4/10  Train: loss=0.0825, acc=0.9716  Test:  loss=1.6679, acc=0.7109
Epoch 5/10  Train: loss=0.0558, acc=0.9820  Test:  loss=1.7365, acc=0.7123
Epoch 6/10  Train: loss=0.0570, acc=0.9810  Test:  loss=1.8746, acc=0.7106
Epoch 7/10  Train: loss=0.0612, acc=0.9793  Test:  loss=1.9215, acc=0.7151
Epoch 8/10  Train: loss=0.0477, acc=0.9841  Test:  loss=1.9880, acc=0.7159
Epoch 9/10  Train: loss=0.0432, acc=0.9861  Test:  loss=2.0320, acc=0.7055
Epoch 10/10  Train: loss=0.0501, acc=0.9827  Test:  loss=2.0542, acc=0.7104
```

Figure 2.4: Training loop on CIFAR-10 data set

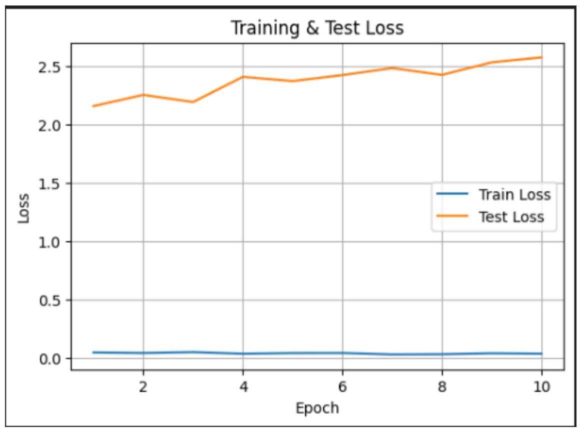


Figure 2.5: Training and Testing loss for SimpleCNN on CIFAR-10 data set

2.5 Results

2.5.1 SimpleCNN Performance

Simple tabular training loop CIFAR-10 as shown on Figure 2.4

Evaluation of SimpleCNN on CIFAR-10 as shown on Figure 2.10 and Figure 2.11
Also, Evaluation of SimpleCNN on MNIST as shown on Figure 2.12

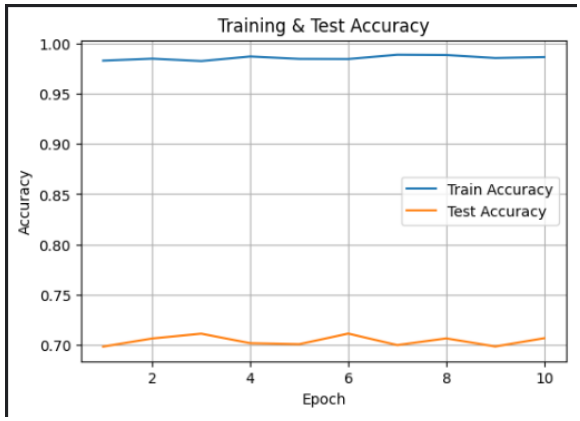


Figure 2.6: Training and Testing Accuracy for SimpleCNN on CIFAR-10 data set

Epoch	Train Loss	Train Acc	Test Loss	Test Acc
1	0.2116	0.9378	0.0604	0.9796
2	0.0530	0.9837	0.0357	0.9881
3	0.0356	0.9885	0.0381	0.9875
4	0.0270	0.9911	0.0347	0.9878
5	0.0221	0.9929	0.0318	0.9900
6	0.0162	0.9948	0.0308	0.9912
7	0.0147	0.9951	0.0324	0.9900
8	0.0105	0.9966	0.0304	0.9911
9	0.0093	0.9970	0.0351	0.9899
10	0.0093	0.9969	0.0339	0.9899

Figure 2.7: Train 10 epochs SimpleCNN on the MNIST dataset

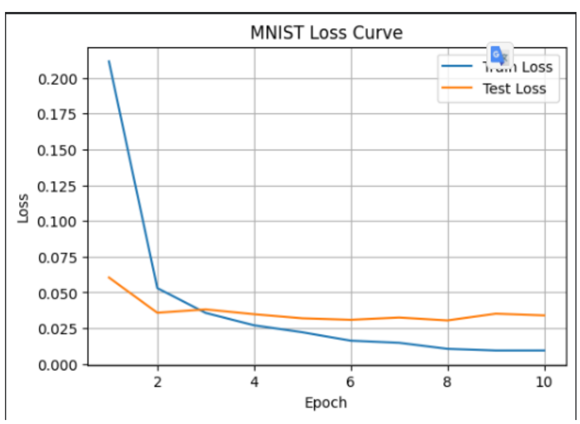


Figure 2.8: MNIST Loss carve for SimpleCNN

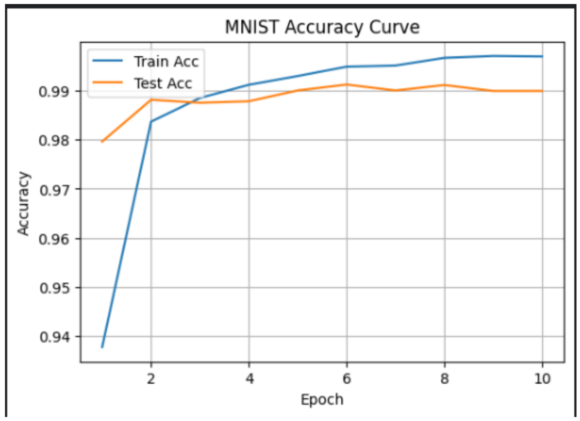
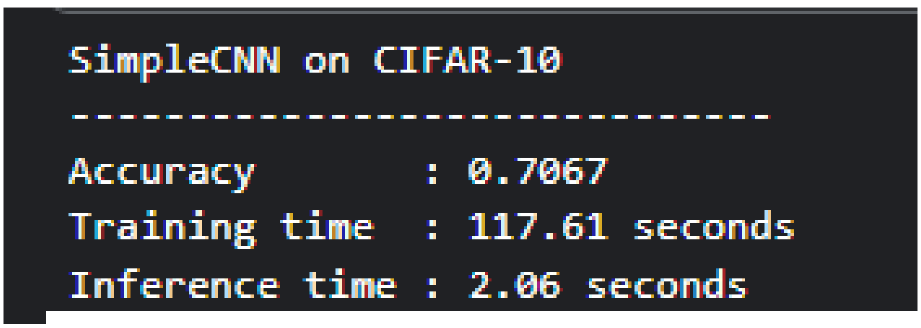


Figure 2.9: MNIST Accuracy carve for SimpleCNN


SimpleCNN on CIFAR-10

Accuracy   : 0.7067

Training time : 117.61 seconds

Inference time : 2.06 seconds

Figure 2.10: Evaluation of SimpleCNN on CIFAR-10

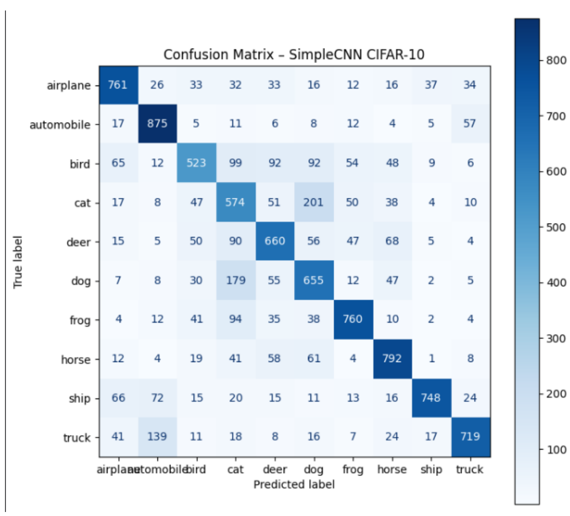


Figure 2.11: Confusion Matrix for SimpleCNN on CIFAR-10

```
SimpleCNN on MNIST
------------------------------
Accuracy       : 0.9903
Training time  : 121.72 seconds
Inference time : 1.75 seconds
```

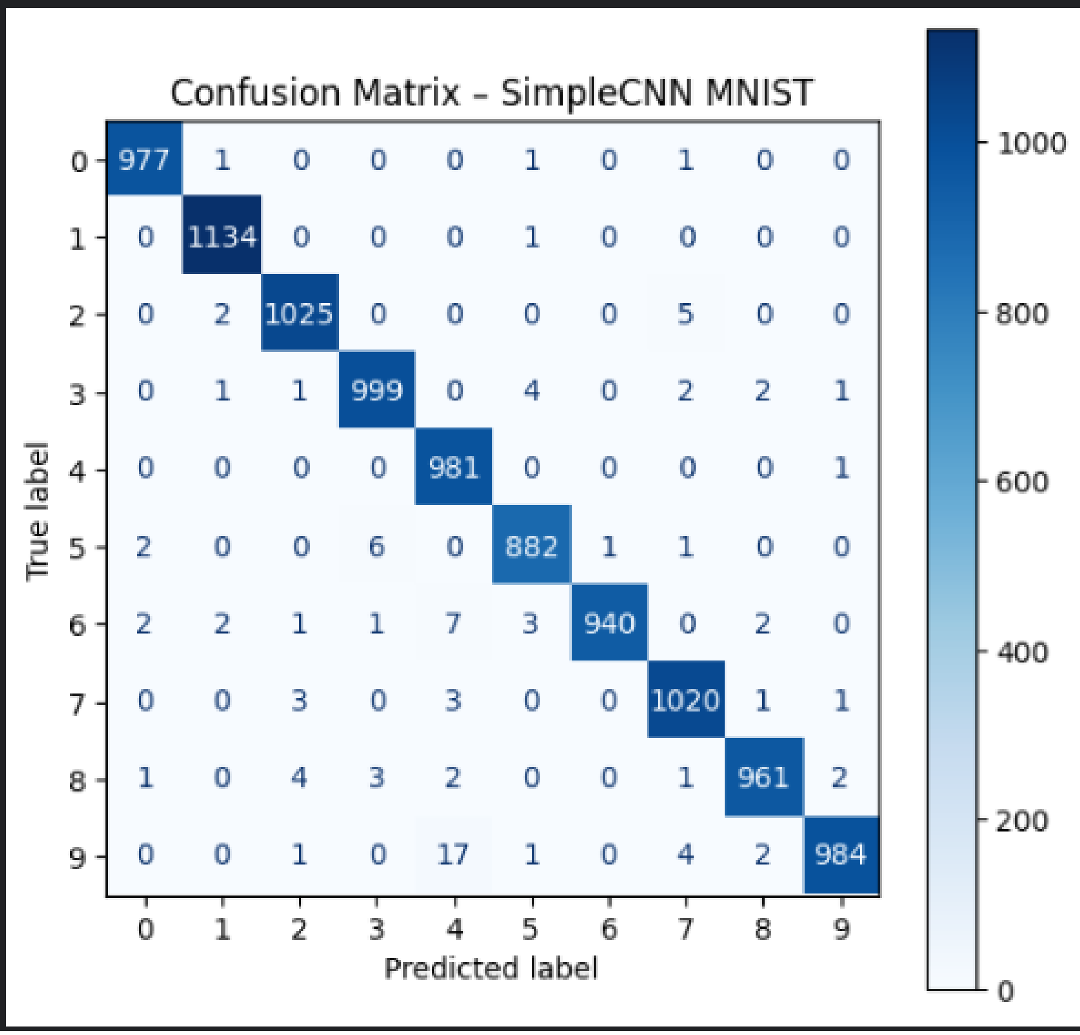


Figure 2.12: Evaluation of SimpleCNN on MNIST

```
AlexNet CIFAR-10 Epoch 1/10  Train: loss=0.7271, acc=0.7462  Test: loss=0.5601, acc=0.8082
AlexNet CIFAR-10 Epoch 2/10  Train: loss=0.5406, acc=0.8134  Test: loss=0.4933, acc=0.8303
AlexNet CIFAR-10 Epoch 3/10  Train: loss=0.4673, acc=0.8374  Test: loss=0.4484, acc=0.8510
AlexNet CIFAR-10 Epoch 4/10  Train: loss=0.4278, acc=0.8533  Test: loss=0.4241, acc=0.8536
AlexNet CIFAR-10 Epoch 5/10  Train: loss=0.3910, acc=0.8639  Test: loss=0.4406, acc=0.8497
AlexNet CIFAR-10 Epoch 6/10  Train: loss=0.3570, acc=0.8782  Test: loss=0.4231, acc=0.8584
AlexNet CIFAR-10 Epoch 7/10  Train: loss=0.3290, acc=0.8872  Test: loss=0.4244, acc=0.8589
AlexNet CIFAR-10 Epoch 8/10  Train: loss=0.3088, acc=0.8933  Test: loss=0.4040, acc=0.8688
AlexNet CIFAR-10 Epoch 9/10  Train: loss=0.2920, acc=0.8989  Test: loss=0.3971, acc=0.8700
AlexNet CIFAR-10 Epoch 10/10  Train: loss=0.2680, acc=0.9090  Test: loss=0.4194, acc=0.8664
```

Figure 2.13: Training 10 epochs for AlexNet on CIFAR-10

2.5.2 AlexNet Performance

Fine-tune AlexNet on CIFAR-10 as shown on Figure 2.13

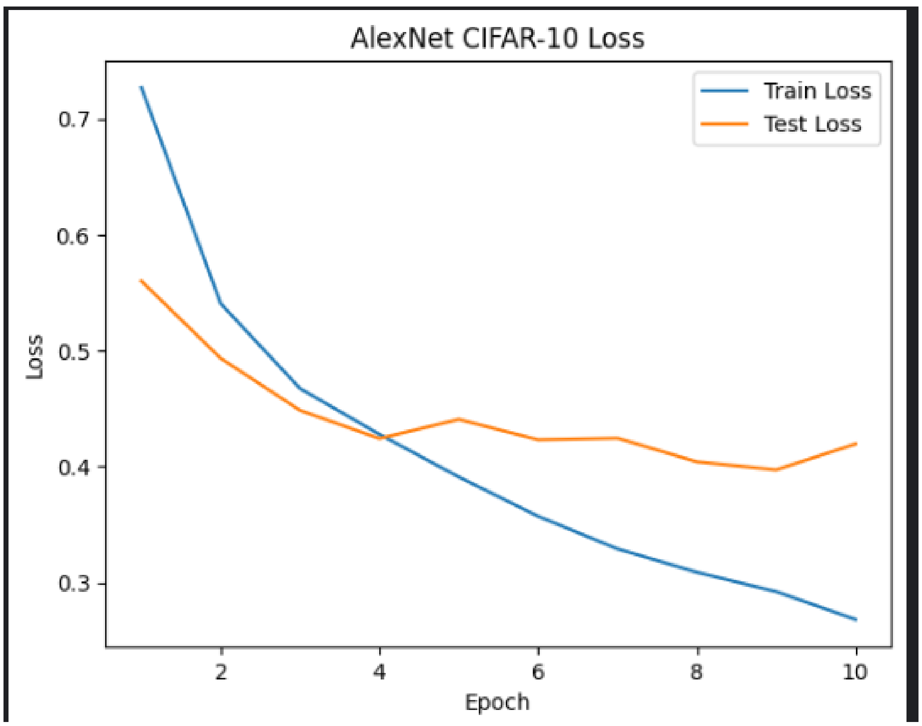


Figure 2.14: AlexNet CIFAR-10 loss carve

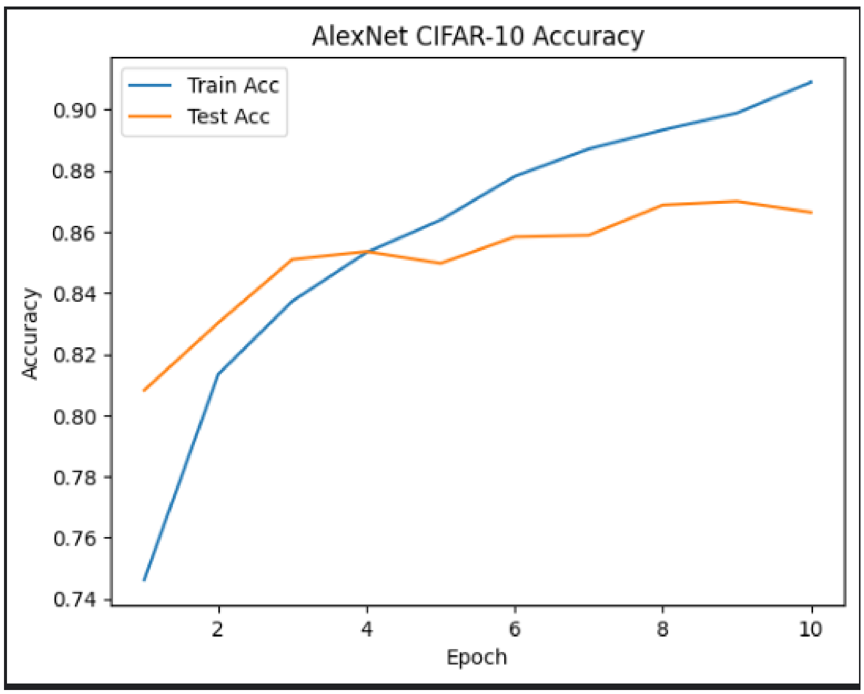


Figure 2.15: AlexNet CIFAR-10 Accuracy

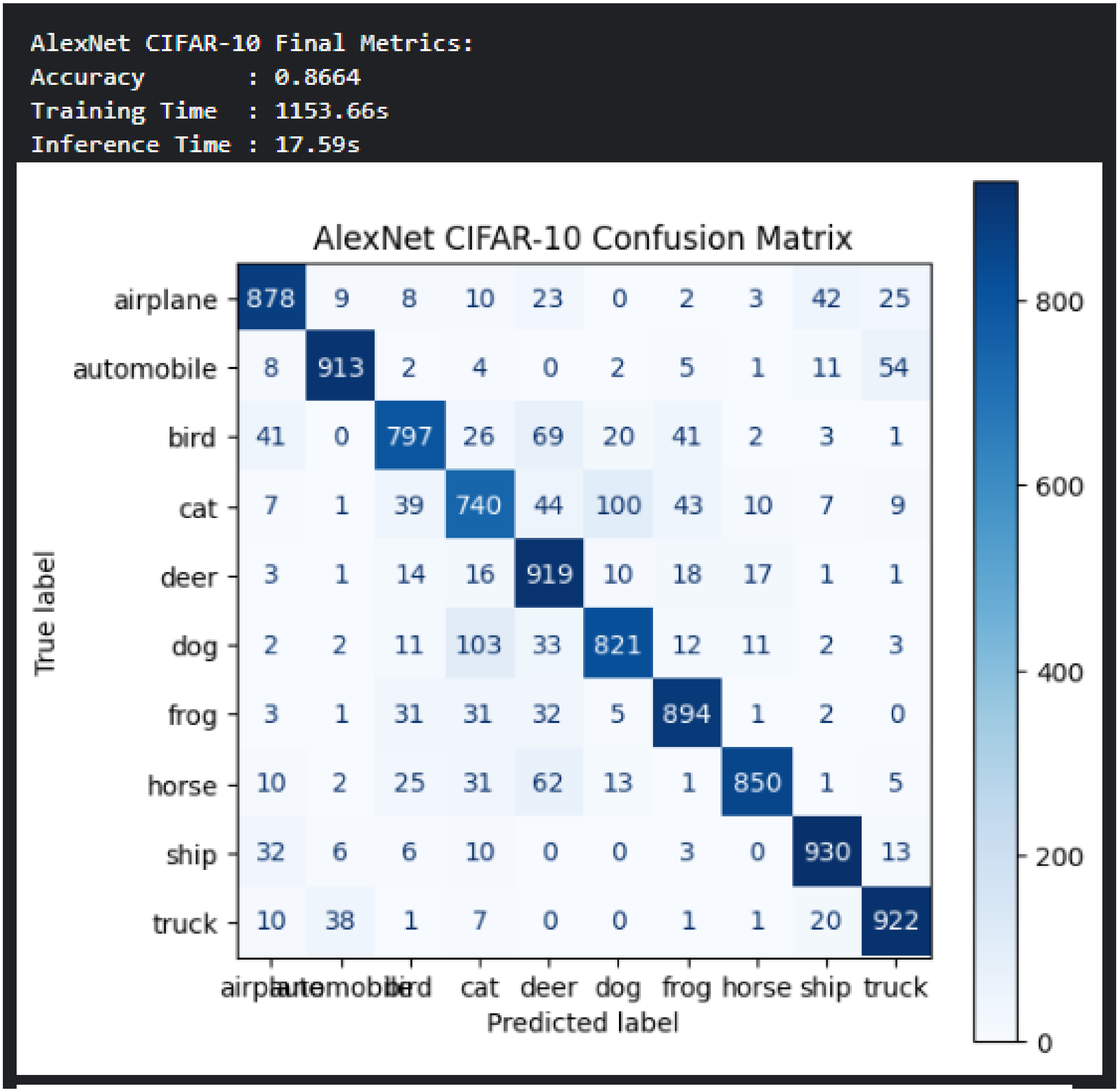


Figure 2.16: Evaluation for AlexNet on CIFAR-10

```
AlexNet MNIST Epoch 1/10  Train: loss=0.1532, acc=0.9551  Test: loss=0.0393, acc=0.9875
AlexNet MNIST Epoch 2/10  Train: loss=0.0784, acc=0.9785  Test: loss=0.0591, acc=0.9830
AlexNet MNIST Epoch 3/10  Train: loss=0.0790, acc=0.9787  Test: loss=0.0442, acc=0.9865
AlexNet MNIST Epoch 4/10  Train: loss=0.0648, acc=0.9828  Test: loss=0.0434, acc=0.9870
AlexNet MNIST Epoch 5/10  Train: loss=0.0697, acc=0.9822  Test: loss=0.0353, acc=0.9896
AlexNet MNIST Epoch 6/10  Train: loss=0.0628, acc=0.9839  Test: loss=0.0414, acc=0.9879
AlexNet MNIST Epoch 7/10  Train: loss=0.0541, acc=0.9860  Test: loss=0.0304, acc=0.9915
AlexNet MNIST Epoch 8/10  Train: loss=0.0490, acc=0.9868  Test: loss=0.0379, acc=0.9895
AlexNet MNIST Epoch 9/10  Train: loss=0.0476, acc=0.9879  Test: loss=0.0274, acc=0.9929
AlexNet MNIST Epoch 10/10  Train: loss=0.0473, acc=0.9879  Test: loss=0.0360, acc=0.9906
```

Figure 2.17: Training 10 epochs AlexNet on MNIST data set

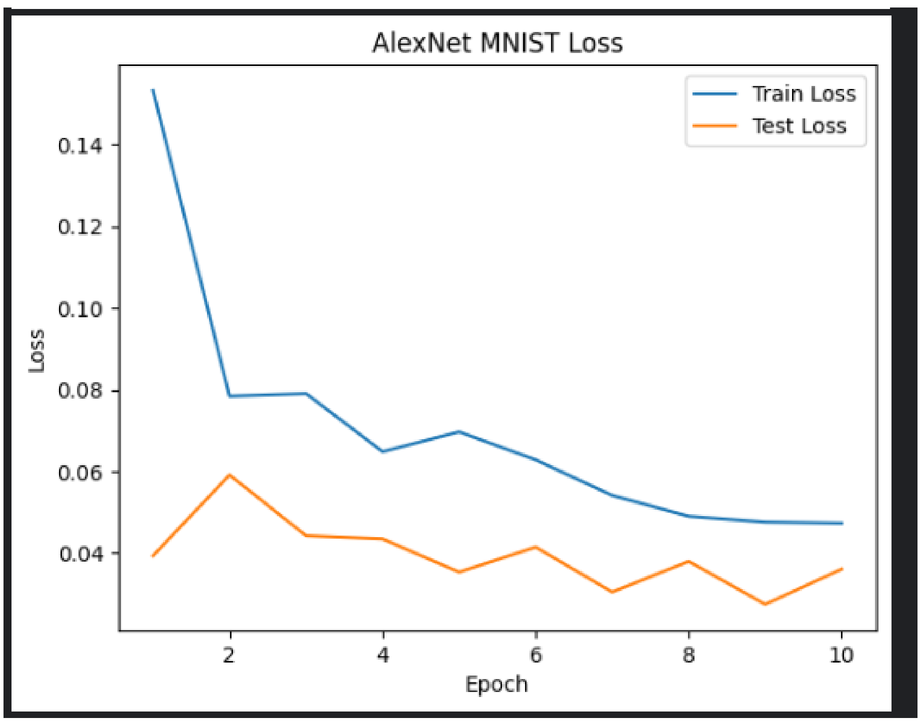


Figure 2.18: AlexNet on MNIST Data set Loss carve

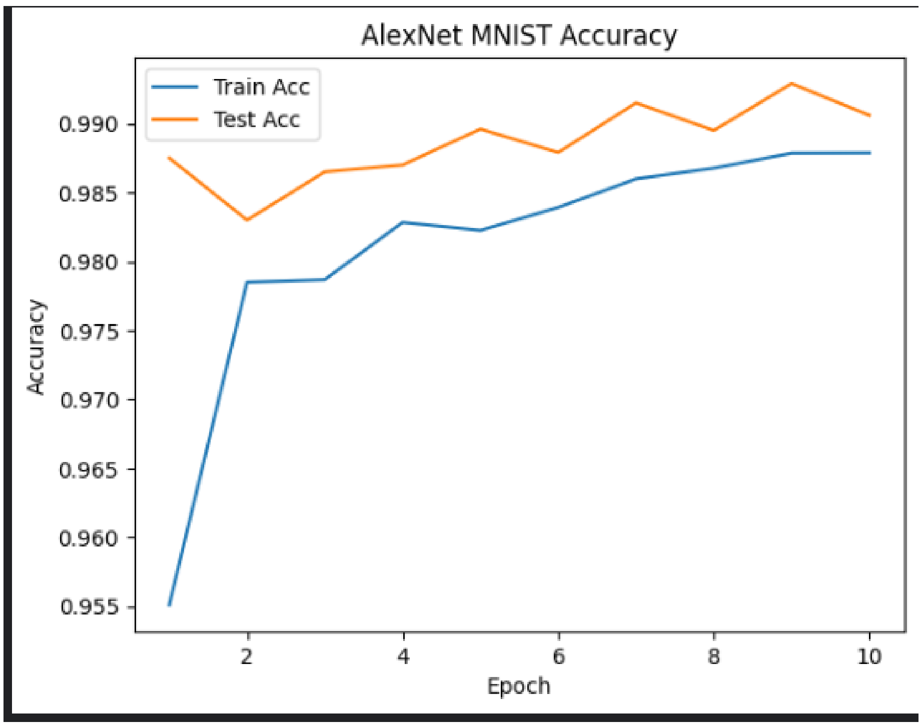


Figure 2.19: AlexNet on MNIST Data set Accuracy carve

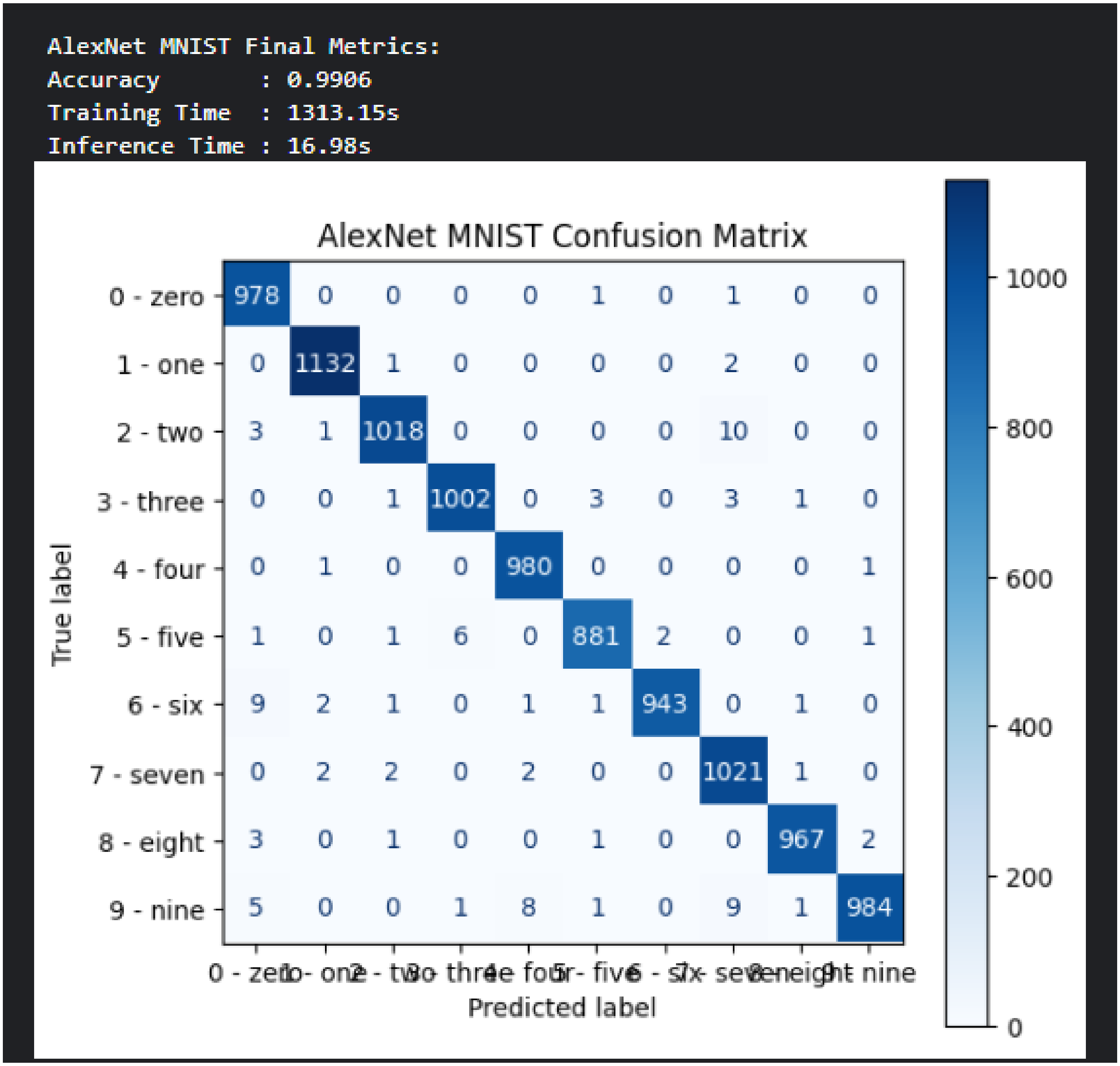


Figure 2.20: Evaluation for AlexNet in MNIST data set

```
Running on device: cuda
PatchLSTM CIFAR-10 Epoch 1/10  Train: loss=2.1427, acc=0.3179  Test:  loss=2.0806, acc=0.3771
PatchLSTM CIFAR-10 Epoch 2/10  Train: loss=2.0724, acc=0.3849  Test:  loss=2.0542, acc=0.4072
PatchLSTM CIFAR-10 Epoch 3/10  Train: loss=2.0418, acc=0.4174  Test:  loss=2.0364, acc=0.4230
PatchLSTM CIFAR-10 Epoch 4/10  Train: loss=2.0196, acc=0.4402  Test:  loss=2.0151, acc=0.4464
PatchLSTM CIFAR-10 Epoch 5/10  Train: loss=2.0010, acc=0.4607  Test:  loss=2.0047, acc=0.4551
PatchLSTM CIFAR-10 Epoch 6/10  Train: loss=1.9861, acc=0.4746  Test:  loss=1.9976, acc=0.4611
PatchLSTM CIFAR-10 Epoch 7/10  Train: loss=1.9715, acc=0.4915  Test:  loss=1.9863, acc=0.4719
PatchLSTM CIFAR-10 Epoch 8/10  Train: loss=1.9577, acc=0.5044  Test:  loss=1.9848, acc=0.4750
PatchLSTM CIFAR-10 Epoch 9/10  Train: loss=1.9461, acc=0.5180  Test:  loss=1.9791, acc=0.4808
PatchLSTM CIFAR-10 Epoch 10/10  Train: loss=1.9350, acc=0.5296  Test:  loss=1.9777, acc=0.4820
```

Figure 2.21: Training 10 epochs for PatchLSTM on CIFAR-10

2.5.3 Patch-LSTM Performance

Training 10 epochs for PatchLSTM on CIFAR-10 as shown on Figure 2.21

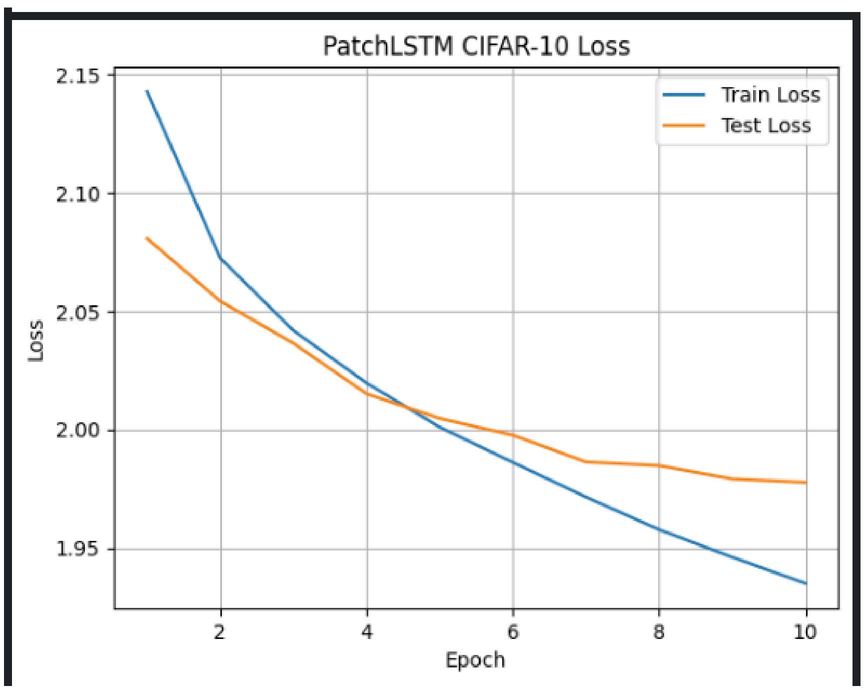


Figure 2.22: PatchLSTM on CIFAR-10 loss carve

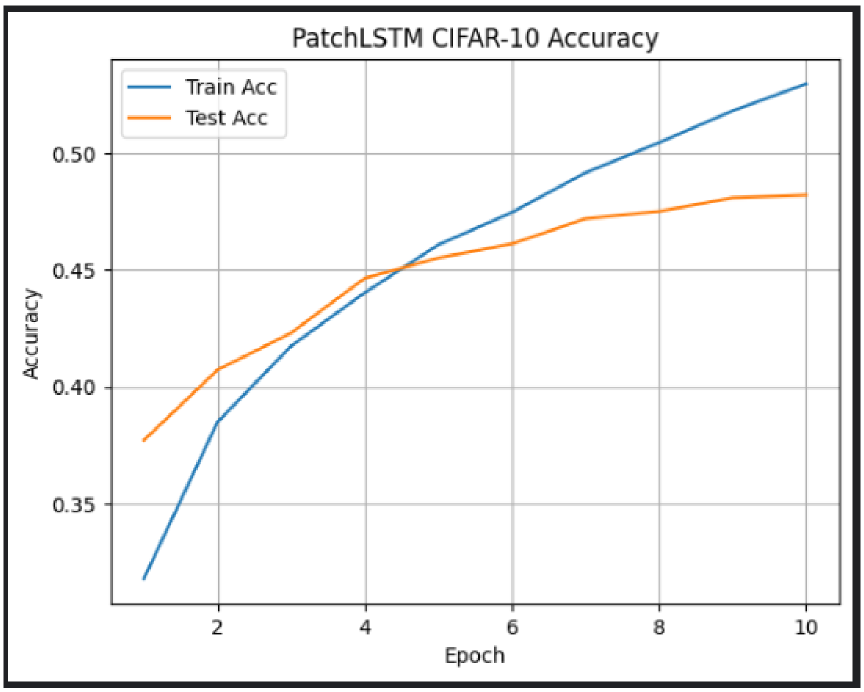


Figure 2.23: PatchLSTM on CIFAR-10 Accuracy carve

```
PatchLSTM CIFAR-10 Final Metrics
------------------------------
Accuracy       : 0.4820
Training Time  : 140.72 s
Inference Time : 2.28 s
```

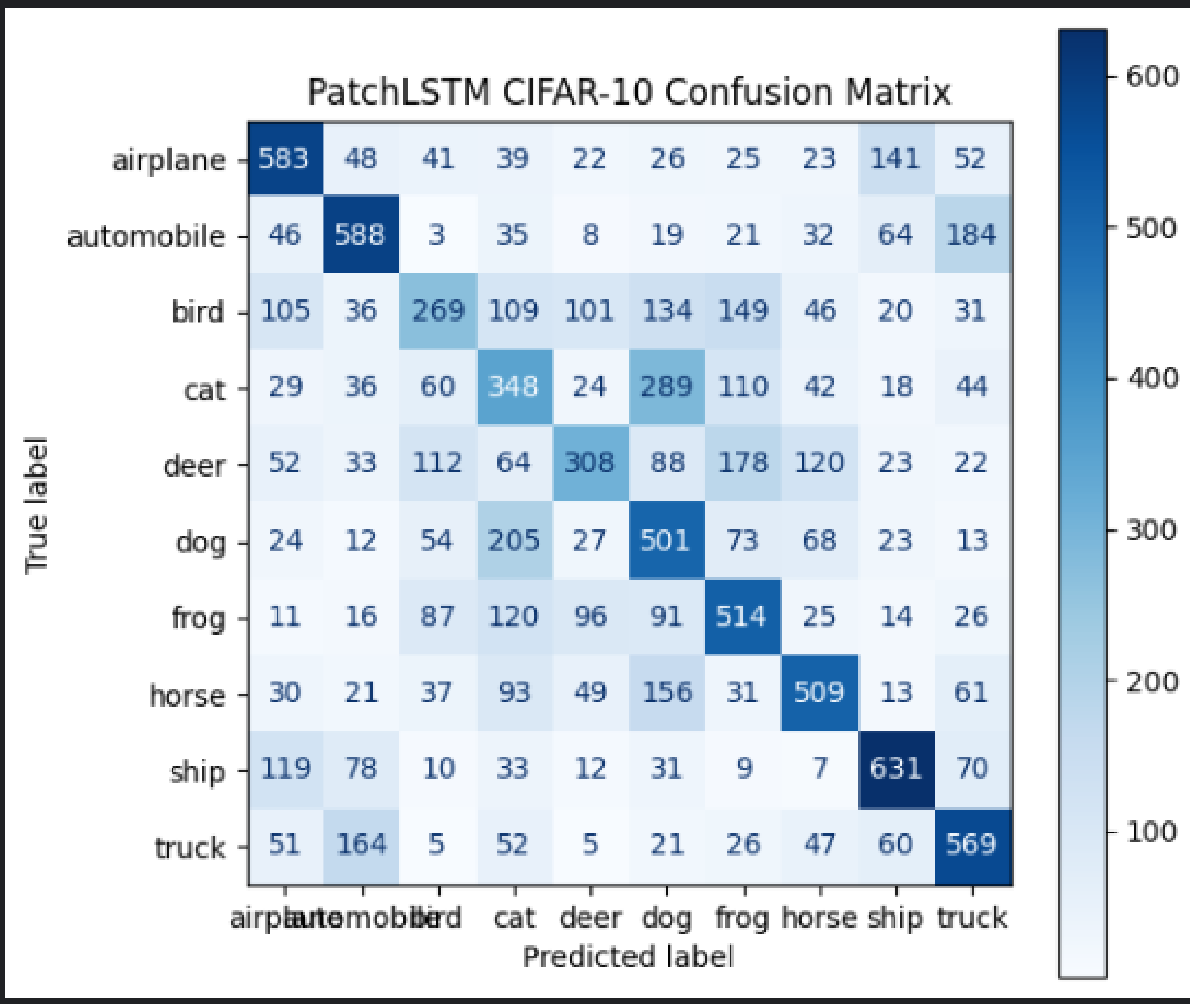


Figure 2.24: Evaluation for PatchLSTM on CIFAR-10 data set

```
PatchLSTM MNIST Epoch 1/10  Train: loss=1.8349, acc=0.6360  Test:  loss=1.6594, acc=0.8110
PatchLSTM MNIST Epoch 2/10  Train: loss=1.5775, acc=0.8909  Test:  loss=1.5545, acc=0.9115
PatchLSTM MNIST Epoch 3/10  Train: loss=1.5343, acc=0.9304  Test:  loss=1.5272, acc=0.9373
PatchLSTM MNIST Epoch 4/10  Train: loss=1.5204, acc=0.9435  Test:  loss=1.5138, acc=0.9490
PatchLSTM MNIST Epoch 5/10  Train: loss=1.5107, acc=0.9527  Test:  loss=1.5223, acc=0.9416
PatchLSTM MNIST Epoch 6/10  Train: loss=1.5025, acc=0.9603  Test:  loss=1.5072, acc=0.9555
PatchLSTM MNIST Epoch 7/10  Train: loss=1.4994, acc=0.9630  Test:  loss=1.4973, acc=0.9650
PatchLSTM MNIST Epoch 8/10  Train: loss=1.4968, acc=0.9656  Test:  loss=1.4980, acc=0.9639
PatchLSTM MNIST Epoch 9/10  Train: loss=1.4947, acc=0.9677  Test:  loss=1.5007, acc=0.9615
PatchLSTM MNIST Epoch 10/10  Train: loss=1.4916, acc=0.9706  Test:  loss=1.4979, acc=0.9649
```

Figure 2.25: Training 10 epochs for PatchLSTM on MNIST Data set

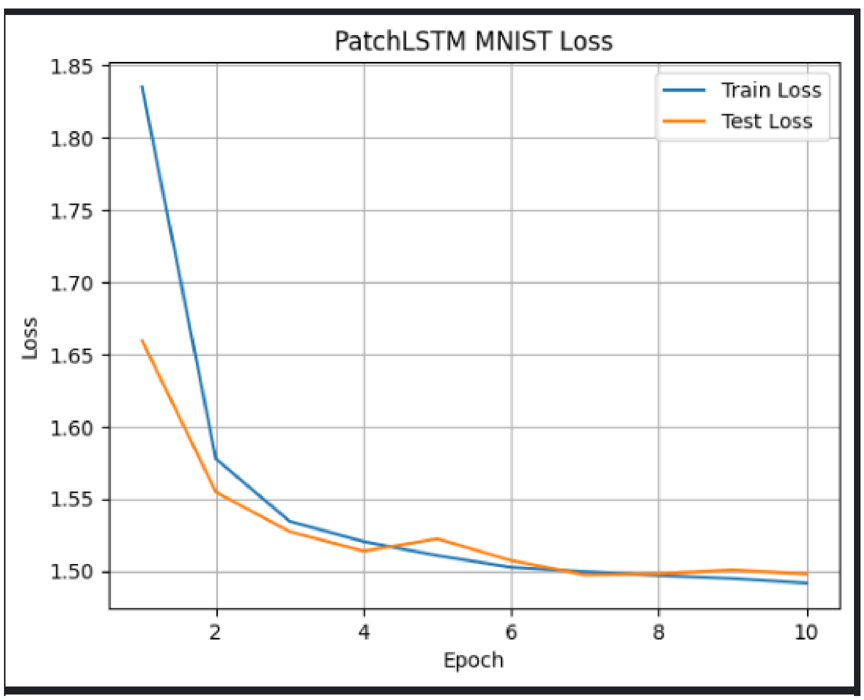


Figure 2.26: PatchLSTM on MNIST data set loss carve

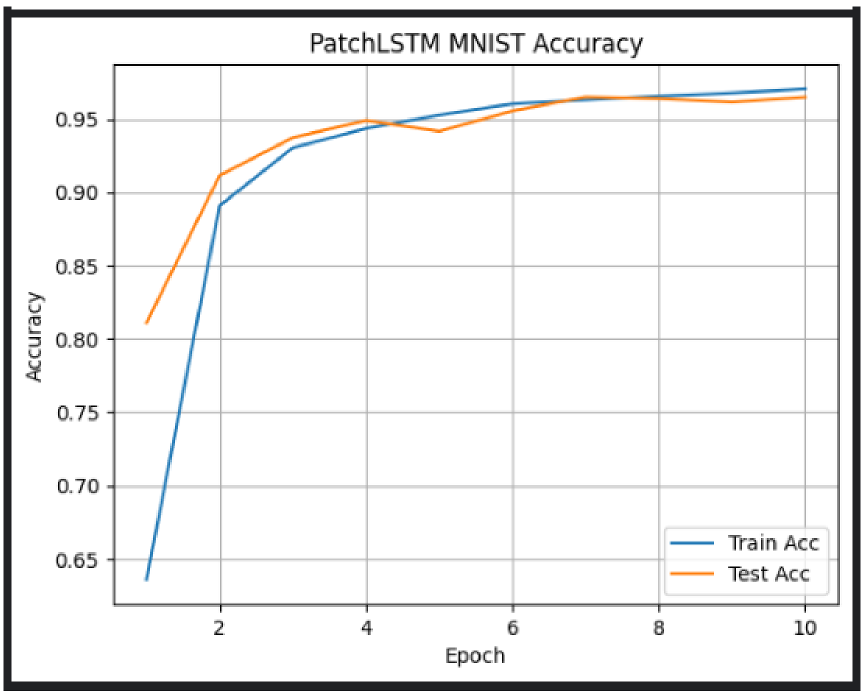


Figure 2.27: PatchLSTM on MNIST data set Accuracy carve

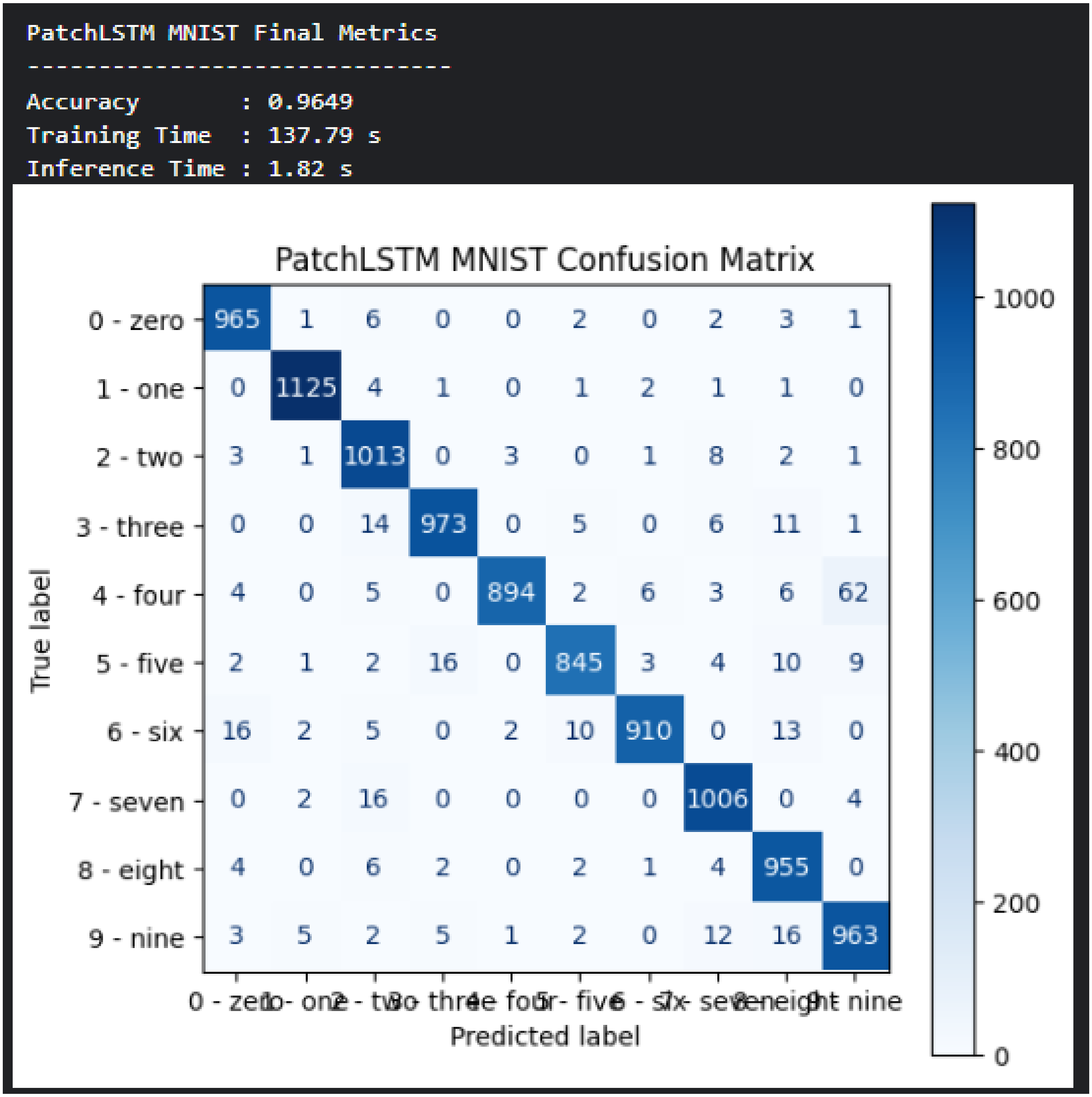


Figure 2.28: Evaluation for PatchLSTM on MNIST data set

3 Conclusion

In this assignment, the performance of three deep learning architectures—SimpleCNN, fine-tuned AlexNet, and Patch-based LSTM—has been examined on the MNIST and CIFAR-10 classification tasks.

AlexNet consistently achieved the highest accuracy by leveraging rich pretrained feature representations, highlighting the power of transfer learning. SimpleCNN offered the fastest training and inference times, demonstrating that a lightweight design can deliver competitive performance with minimal computational cost.

The Patch-LSTM model provided strong results on the simpler MNIST dataset but incurred greater computational overhead due to sequential patch processing.
These findings underscore the trade-offs between model complexity, predictive accuracy, and resource efficiency. While deeper, pretrained networks like AlexNet excel in classification performance, simpler CNNs or sequence-based architectures may be more suitable for applications with strict latency or hardware constraints.

Ultimately, the choice of architecture should balance the desired accuracy against available computational resources and real-time requirements.