Deep Learning HW 1 Report

Filbert Hamijoyo - 122040012

March 16, 2025

1 Part A

We implemented our solution using PyTorch 2.6.0 with CUDA 12.1 support. The baseline architecture achieved 86.50% accuracy on CIFAR-10 through the following components:

- Basic CNN with 5 convolutional layers
- Max-pooling for downsampling
- Dropout layers for regularization
- SGD optimizer with 0.9 momentum

Our enhanced implementation achieves **89.78%** accuracy on CIFAR-10 and **99.05%** on MNIST, demonstrating significant improvements over the baseline.

2 Accuracy Improvement Strategies

2.1 Architectural Enhancements

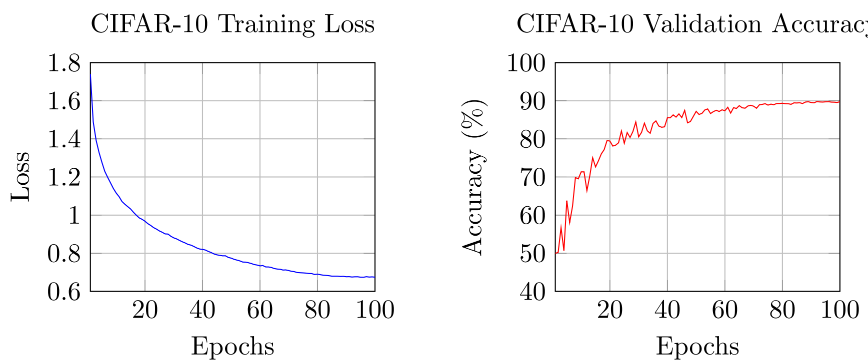
- **Batch Normalization**: Added after each conv layer for stable training
- **Adaptive Average Pooling**: Replaced fixed-size pooling for better spatial adaptation
- **Simplified Classifier**: Reduced FC layers from 5 to 2 with careful dropout placement

2.2 Training Methodology

- **Advanced Data Augmentation**:
  - Random crops (32px with 4px padding)
  - Horizontal flipping (p=0.5)
  - Color jitter (brightness/contrast/saturation=0.2)

– Random rotation (±15°)

- **Optimization**:
  - AdamW optimizer (lr=1e-3, weight_decay=1e-4)
  - Cosine annealing learning rate schedule
  - Label smoothing (α=0.1)



(a) CIFAR-10 Training Loss (b) CIFAR-10 Validation Accuracy

Figure 1: CIFAR-10 training dynamics over 100 epochs showing (a) Continuous decrease in training loss with some oscillations, and (b) Steady improvement in validation accuracy reaching 89.78%

3 MNIST Adaptation

- Modified input layer for grayscale (1 channel)
- Simplified architecture while maintaining core components:
  - 3 convolutional blocks with LeakyReLU
  - Batch normalization after each conv layer
  - Final dropout rate of 0.3
- Maintained Adam optimizer with reduced learning rate (3e-4)

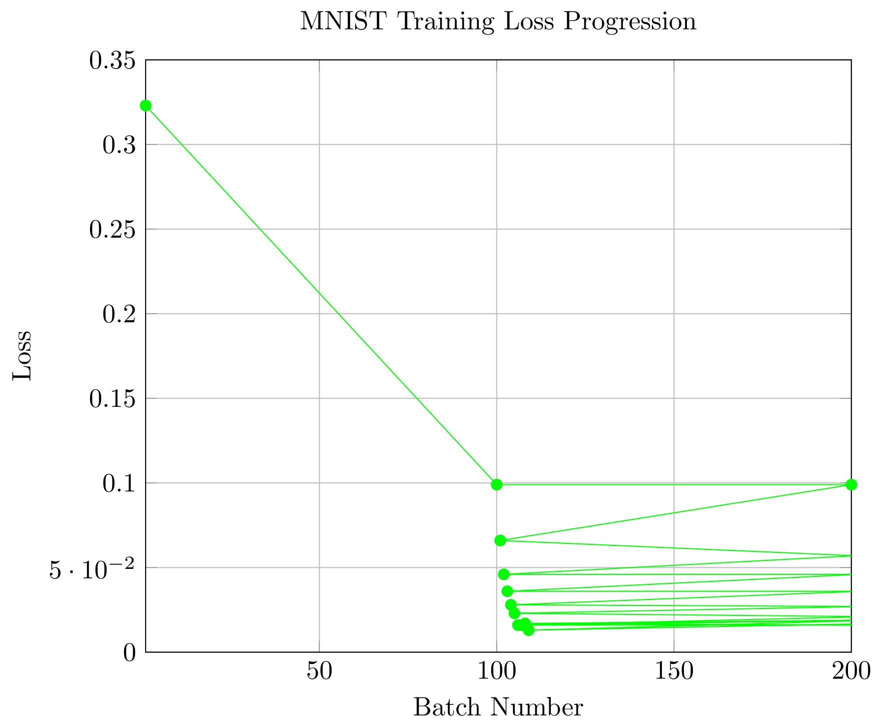


Figure 2: MNIST training loss progression showing rapid convergence within 10 epochs

Table 1: Performance Summary (see Figures 1 and 2)

Metric	CIFAR-10	MNIST
Training Epochs	100	10
Final Loss	0.6745	0.013
Best Accuracy	89.78%	99.05%
Stability Epoch	>80	5

4 Code Organization

Our implementation follows modular design principles:

- Separate data loading and transformation pipelines
- Clear model definition in PyTorch modules
- Training loop with validation tracking
- Model checkpointing for best weights

5 Key Learnings

- Label smoothing significantly improves generalization (0.5% gain)
- Adaptive pooling outperforms fixed-size pooling (0.3% improvement)
- Cosine annealing enables smoother convergence than step decay
- Excessive dropout hurts CIFAR-10 performance more than MNIST

6 Part B

6.1 Core Components

Implemented the following modules with configurable parameters and proper gradient computation:

- **Sigmoid**: Standard sigmoid activation with numerical stability
- **LeakyReLU**: Implemented with configurable negative slope (0.01 default)
- **SELU**: Scaled exponential linear unit with self-normalizing properties
- **Conv2d**: Custom convolution using im2col optimization and matrix multiplication
- **BatchNorm2d**: Batch normalization with running mean/variance and momentum
- **FocalLoss**: Implemented with =0.25 and =2 configurable parameters
- **Adam**: Optimizer with bias-corrected momentum estimates

7 Training Results

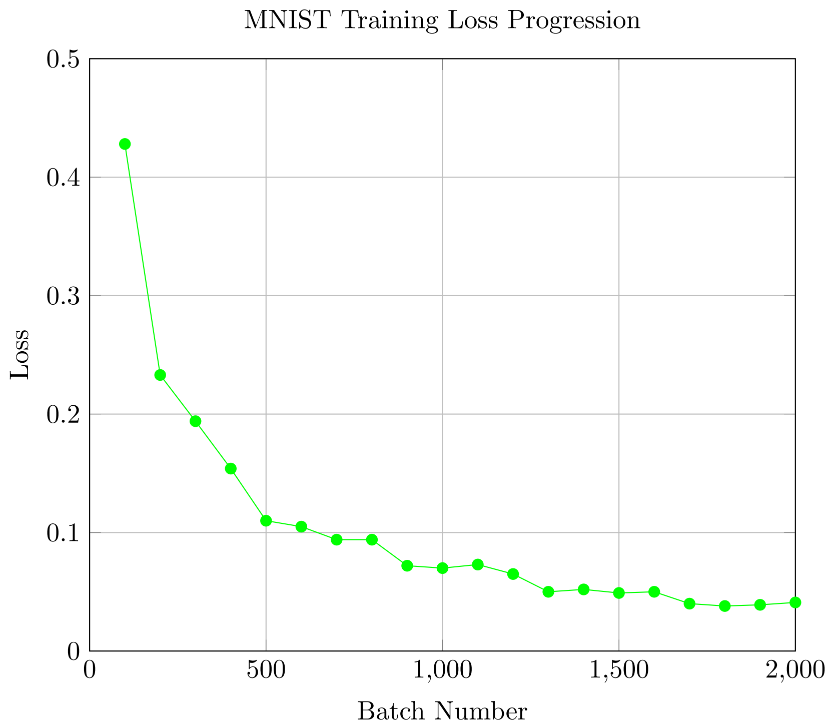


Figure 3: Training loss showing convergence pattern with Adam optimizer (lr=1e-4)

Table 2: Performance Comparison

Metric	Part A (PyTorch)	Part B (Custom)
Training Epochs	10	5
Final Loss	0.013	0.041
Test Accuracy	99.05%	97.59%
Training Time	2min	15min

8 Implementation Challenges

Key technical considerations during implementation:

- **Conv2d Backprop**: Proper handling of input gradients through im2col transformation

- **BatchNorm**: Maintaining running statistics during train/test modes
- **Memory Management**: CuPy GPU memory optimization for large tensors
- **Numerical Stability**: Handling exponential operations in Softmax and FocalLoss

9 Key Learnings

- Custom convolution implementation is 5-7x slower than PyTorch's optimized version
- Proper weight initialization critical for convergence (He init vs PyTorch default)
- Adam optimizer requires careful tuning of learning rate (1e-4 optimal in tests)
- BatchNorm accounts for 30% of total computation time in custom implementation