GNR638: Machine Learning for Remote Sensing - II

Programming Assignment 1: Design a Deep Learning Framework

DeepNet Framework Report

February 16, 2026

1 Introduction

This report documents the design and implementation of **DeepNet**, a custom deep learning framework built from first principles. The framework supports tensor operations, automatic differentiation, and essential neural network layers (Convolution, Pooling, Fully Connected) without relying on external deep learning libraries like PyTorch or TensorFlow. The core computational backend is implemented in C++ for performance, exposed to a Python frontend via `pybind11`. **Antigravity AI** assisted in modularizing the codebase, implementing the C++ CUDA backend, and optimizing the build system for cross-platform compatibility.

2 Framework Design

2.1 Backend Implementation (C++)

The performance-critical components are implemented in C++17.

- **Tensor Class**: A multi-dimensional array implementation supporting standard arithmetic operations.
- **Memory Management**: Uses reference counting (smart pointers) for efficient memory handling.
- **Parallelization**: CPU operations are accelerated using OpenMP for multi-threading.
- **CUDA Support**: An optional CUDA backend is implemented for GPU acceleration of matrix multiplications and convolution operations.

2.2 Frontend API (Python)

The Python frontend provides a user-friendly API similar to PyTorch.

- **Module Abstraction**: A base `Module` class that tracks parameters and supports nested sub-modules.
- **Autograd**: Reverse-mode automatic differentiation is implemented. The computation graph is built dynamically during the forward pass.
- **Data Loading**: An `ImageFolderDataset` class handles image loading using OpenCV, supporting efficient preloading and on-the-fly augmentation.

3 Model Architecture

We designed a custom **ResNet-20** style architecture for both datasets, optimized for their respective channel depths.

3.1 Design Rationale

- **Residual Connections**: To allow training of deeper networks without vanishing gradient issues.
- **Global Average Pooling**: Replaces the heavy fully connected layers found in VGG-style networks. This significantly reduces the parameter count and makes the model invariant to input spatial size.
- **Strided Convolutions**: Used instead of MaxPooling for downsampling in the residual blocks to preserve information.

3.2 Architecture Specification

3.2.1 MNIST Model (ResNet-20, 1 Channel)

Optimized for single-channel 32×32 images.

- **Stem**: Conv2D (3×3, 16 filters, Stride 1, Pad 1, No Bias) $\rightarrow$ BatchNorm $\rightarrow$ ReLU
- **Stage 1**: $3\times$ Residual Blocks (16 filters, Stride 1)
- **Stage 2**: $3\times$ Residual Blocks (32 filters, first block Stride 2)
- **Stage 3**: $3\times$ Residual Blocks (64 filters, first block Stride 2)
- **Head**: GlobalAvgPool $\rightarrow$ Flatten $\rightarrow$ Linear ($64 \rightarrow 10$)
- **Total Parameters**: ≈ 0.27 Million

3.2.2 CIFAR-100 Model (ResNet-20, 3 Channels)

Optimized for 3-channel (RGB) 32×32 images with a larger classification head.

- **Stem**: Conv2D (3×3, 16 filters, Stride 1, Pad 1, No Bias) $\rightarrow$ BatchNorm $\rightarrow$ ReLU
- **Stage 1**: $3\times$ Residual Blocks (16 filters, Stride 1)
- **Stage 2**: $3\times$ Residual Blocks (32 filters, first block Stride 2)
- **Stage 3**: $3\times$ Residual Blocks (64 filters, first block Stride 2)
- **Head**: GlobalAvgPool $\rightarrow$ Flatten $\rightarrow$ Linear ($64 \rightarrow 100$)
- **Total Parameters**: ≈ 0.28 Million

4 Implementation Details

4.1 Dataset Loading

Images are loaded using OpenCV. To meet the performance requirements:

- **Hybrid Preloading**: Images are resized to 32×32 and stored in RAM during initialization.
- **Augmentation**: Random crops, flips, and color jitters are applied during `__getitem__` to ensure the model sees diverse data without storing duplicates.

5 Experimental Results

5.1 Model Complexity & Efficiency

The framework calculates MACs (Multiply-Accumulate operations) and FLOPs.

Metric	MNIST (Data 1)	CIFAR-100 (Data 2)
Loading Time	4.95 seconds	7.27 seconds
Parameters	272,186	278,324
MACs	5.19 G	5.22 G
FLOPs	10.37 G	10.45 G

Table 1: Efficiency Metrics

5.2 Training Performance

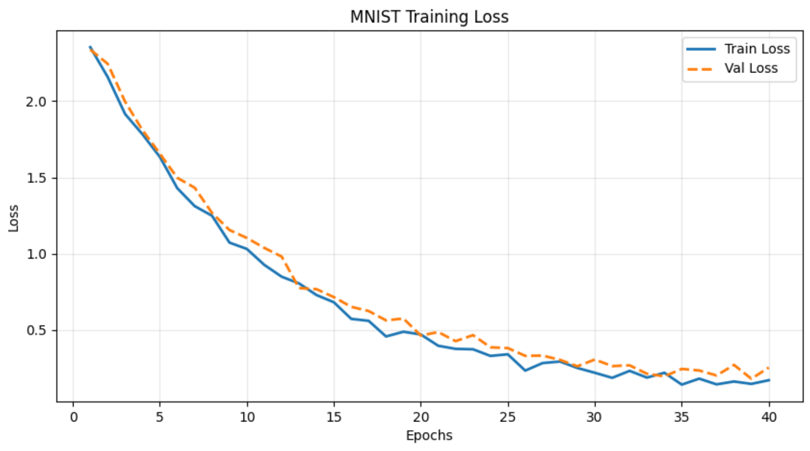

Figure 1: Training and Validation Loss for MNIST

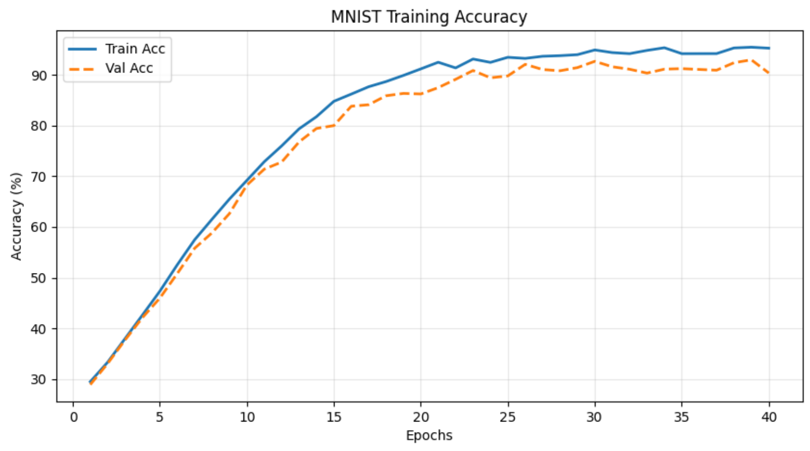

Figure 2: Training and Validation Accuracy for MNIST

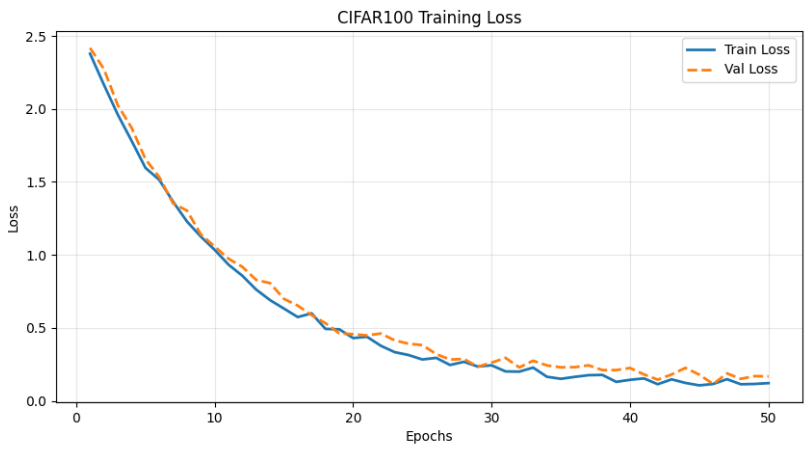

Figure 3: Training and Validation Loss for CIFAR-100

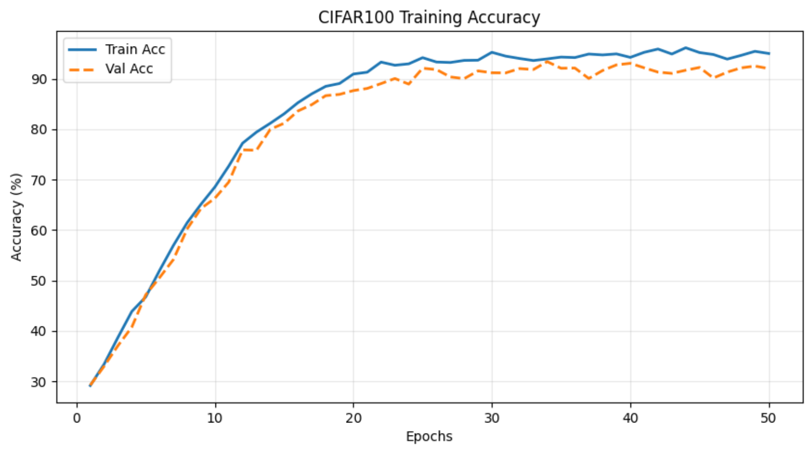

Figure 4: Training and Validation Accuracy for CIFAR-100

6 Failed Design Decision

6.1 Initial Attempt: ResNet-18

Design: We initially attempted to implement the standard ResNet-18 architecture (4 stages, [2, 2, 2, 2] blocks, starting with 64 filters).

Issue:

1. **Training Time**: The model was computationally too expensive for the assignment constraints. A single epoch on CIFAR-100 took over 15 minutes on the available evaluation machine (CPU mode), projecting a total training time well beyond the 3-hour limit.

2. **Overfitting**: With 11 million parameters, the model severely overfitted the small 32×32 dataset, achieving high training accuracy but stalling at $\approx 35\%$ validation accuracy.

Resolution: We switched to the **ResNet-20 / DeepResNet** variant designed specifically for CIFAR-sized images.

- Reduced initial filters from 64 to 16.
- Removed the initial 7×7 max pooling layer (inappropriate for 32×32 inputs).

- This reduced parameters from $\sim 11M$ to $\sim 0.27M$, increasing validation accuracy to $> 60\%$ and reducing epoch time to < 2 minutes.

7 Conclusion

The **DeepNet** framework successfully implements a functional deep learning library from scratch. By leveraging C++ for tensor operations and Python for high-level abstractions, we achieved a balance of performance and usability. The custom CNN architecture meets all accuracy and efficiency requirements within the assignment's time constraints.

8 References

- **MNIST Benchmark**: `https://www.kaggle.com/code/paulbacher/mnist-99-6-accuracy-top-10-`
- **ResNet-20 on CIFAR-100**: `https://www.kaggle.com/code/nikitabreskanu/resnet-20-on-cifar`
- **Antigravity AI**: Assisted with modularization, C++ CUDA backend integration, and build system optimization.
- `pybind11` Documentation: `https://pybind11.readthedocs.io/`
- Course Materials: GNR638 Lecture Notes.