CS 6375: Machine Learning
Project 3: Deep Learning for MNIST and CIFAR-10

In this project, you will implement and evaluate Multilayer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs) on two well-known image datasets: **MNIST** and **CIFAR-10**. You will utilize **PyTorch** for implementation. *For help with PyTorch, refer to the official documentation at:* `https://pytorch.org/docs`.

Datasets and Preprocessing

- Use PyTorch (`torchvision`) to load MNIST and CIFAR-10 datasets.
- Normalize pixel values to [0,1] or use standard normalization.
- Clearly document your preprocessing steps.

Part 1: Multilayer Perceptrons (MLPs)

- Implement MLPs that accept flattened images from MNIST and CIFAR-10.
- Evaluate **at least three distinct architectures**:
  1. Shallow (1 hidden layer, e.g., 128 units)
  2. Medium-depth (3 hidden layers, e.g., [512, 256, 128])
  3. Deep (at least 5 hidden layers, your choice)
- For each architecture and each dataset, perform:
  - **Validation-based hyperparameter tuning** using a held-out validation set (e.g., use 45,000 training samples and 5,000 validation samples for CIFAR-10; 50,000 training samples and 10,000 validation samples for MNIST). Use `torch.utils.data.random_split` or a sampler to construct validation sets.
  - Tune the following key hyperparameters:
    - Learning rate (e.g. $\{0.01, 0.001, 0.0001\}$)
    - Batch size (e.g. $\{32, 64, 128\}$)
    - Optimizer (SGD vs. Adam)
    - Dropout rate (e.g. {0.2,0.5})

Table 1: MNIST Results

Architecture	**Learning rate**	**Batch size**	**Optimizer**	**Dropout**	**Validation Acc (±std)**	**Runtime (min)**
MLP (shallow, 1 hidden)	...	...	...	...	...	...
MLP (medium, 3 hidden)	...	...	...	...	...	...
MLP (deep, ≥5 hidden)	...	...	...	...	...	...
Test accuracy on final MLP model: ...						
CNN (baseline, 2 conv)	...	...	...	...	...	...
CNN (enhanced, BN+dropout)	...	...	...	...	...	...
CNN (deeper, ≥3 conv)	...	...	...	...	...	...
Test accuracy on final CNN model: ...						

Table 2: CIFAR-10 Results

Architecture	**Learning rate**	**Batch size**	**Optimizer**	**Dropout**	**Validation Acc (±std)**	**Runtime (min)**
MLP (shallow, 1 hidden)	...	...	...	...	...	...
MLP (medium, 3 hidden)	...	...	...	...	...	...
MLP (deep, ≥5 hidden)	...	...	...	...	...	...
Test accuracy on final MLP model: ...						
CNN (baseline, 2 conv)	...	...	...	...	...	...
CNN (enhanced, BN+dropout)	...	...	...	...	...	...
CNN (deeper, ≥3 conv)	...	...	...	...	...	...
Test accuracy on final CNN model: ...						

  - Instead of exhaustively testing all 36 combinations, explore at least **10–12 meaningful configurations** for each architecture (random search or small grid is acceptable). Select the best configuration based on validation accuracy, then retrain the final model on the combined training and validation data, and report test accuracy on the designated test set.
  - For each dataset and architecture, present results in the format as shown in Tables 1 and 2 (Focus on the rows for MLPs). Clearly justify your final chosen model.

Part 2: Convolutional Neural Networks (CNNs)

- Implement CNNs for image classification on MNIST and CIFAR-10.
- Evaluate **at least three distinct CNN architectures**:
  1. Baseline CNN (2 convolutional layers + pooling, fully connected layer)
  2. Enhanced CNN (add batch normalization and dropout)
  3. Deeper CNN (at least 3 convolutional layers with pooling, normalization, dropout, etc.)
- For each CNN architecture and each dataset, perform validation-based hyperparameter tuning similar to the above (Learning rate, Batch size, Optimizer, Dropout rate) and present results in the format as shown in Tables 1 and 2 (Focus on rows for CNNs).

- Discuss the relative performance improvement CNNs provide over MLPs, and analyze how different CNN architectures perform on each dataset.

Experimental Guidance (Important)

- For all hidden layers in both MLPs and CNNs, use **ReLU** activations. The final output layer should be followed by a softmax (via `CrossEntropyLoss` in PyTorch).
- **Validation splits must be clearly documented.** Clearly state how you partitioned training vs. validation data (e.g., 45k/5k for CIFAR-10, 50k/10k for MNIST).
- **Carefully manage runtime.** On CPU, expect approximate training times per model:
  - MNIST: MLP ∼5–10 minutes, CNN ∼10–20 minutes
  - CIFAR-10: MLP ∼15–25 minutes, CNN ∼30–60 minutes
- **GPU (Colab recommended)** significantly reduces runtime (MNIST: seconds, CIFAR-10: 1–3 minutes per training run).
- **Early stopping:** You may stop training when validation accuracy plateaus or begins to decrease. Report the stopping epoch if you use early stopping.

Strong Recommendation: Use Google Colab (Free GPU)

To efficiently complete this deep learning project, we **strongly recommend** using **Google Colab**, which provides **free GPU access**. Colab significantly speeds up training (10–20x faster than CPU), making your experimentation and hyperparameter tuning quicker and easier.

- Access Colab: `https://colab.research.google.com/`
- GPU Setup: In Colab, click on: `Runtime` → `Change runtime type` → `Hardware accelerator: GPU (T4 GPU)`

Grading Criteria

- Correct implementation and clear use of validation splits: 40 points
- Comprehensive hyperparameter exploration and reporting: 30 points
- Clear, detailed analysis of results (tables, comparisons, and **justification of final model**): 20 points
- Code clarity and reproducibility: 10 points

What to turn in

Submit a single zip file containing:

- A PDF report (max 6 pages; **tables may extend beyond the page limit**) clearly describing:

- Detailed architectures tested
- Hyperparameter tables with validation results (accuracy $\pm$ std)
- Test-set results for your final chosen models
- Discussion explaining why certain hyperparameters/architectures performed better, supported by results from your tables
- Describe one key challenge you faced and how you resolved it.

Your code must compile and reproduce your results exactly, or no credit will be given. To ensure reproducibility, try setting random seeds in your code and mention it in your report.