Assignment 1

Computer Vision — Assignment: CNNs from Scratch & Fine-Tuning a pretrained model

Introduction

This assignment gets you hands-on with two complementary skills:

1. **Building a tiny deep-learning stack** (no autograd!) to understand how the PyTorch framework works under the hood, and
2. **Fine-tuning a pretrained CNN efficiently** to deliver strong results with limited compute.
   You will complete two tasks and submit code plus a concise report.

NOTE:

Please install `torchvision` and any other libraries you deem necessary as you tackle the 2 tasks.

Learning Objectives

- Implement forward & backward passes for core CNN primitives (Conv2D, MaxPool2D, ReLU, Linear, Cross-Entropy).
- Read, split, and pre-process image datasets correctly.
- Fine-tune a pretrained ResNet-18 on EMNIST, balancing **accuracy** and **compute efficiency**.
- Produce clear experiments, metrics, and ablations that are **reproducible**.

Task 1 — Build a CNN From Scratch (no autograd)

Goal. Implement the core layers and train a small CNN on MNIST.

What to implement

Write a minimal DL stack (pure PyTorch tensors; no autograd on params):

- Layers: `Linear`, `Conv2D`, `MaxPool2D`, `ReLU`, `Flatten`
- Loss: `CrossEntropy` (logits input)
- The `Sequential` class which chains multiple layers together
- The SGD training loop

I am providing the resources on how to build an Artificial Neural Network (ANN) from scratch:

1. PyTorch basics: https://course.fast.ai/Lessons/lesson11.html (just focus on relevant bits.) or any other PyTorch Tutorial on the internet
2. ANN basis:
   1. https://course.fast.ai/Lessons/lesson13.html
   2. https://course.fast.ai/Lessons/lesson14.html (Note the way the code is finally refactored in this video. This forms the basis for the implementation style followed in Task 1)
   These resources may or may not serve you 100% as you go through the assignment. You are free to refer to other resources as well. However, I urge you to do this assignment sincerely as it serves as a good starting point for the latter assignments as well as anything you decide to build using Huggingface and PyTorch later on.

Your major efforts in Task 1 would be to implement the Convolution, Max Pool, and CrossEntropy classes, as well as writing the final training loop. Please note that this is a non-trivial task, and I have provided detailed comments on how you can go about implementing it. You are also encouraged to check out the code implementations of the same in the PyTorch Library / Huggingface. You may expect questions from this section to come in the quiz as well.

Task 1 Tests

A provided test script checks:

- Numeric gradient sanity for layers & loss
- Forward shape checks
- Tiny end-to-end training sanity (loss drops)

> You must **pass all tests** and demonstrate a brief MNIST training run.

Suggested target (not hard-graded): ≥90% test accuracy with a small 2-conv CNN.

Task 2 — Efficient Fine-Tuning: ResNet-18 on EMNIST

Goal. Train a strong EMNIST classifier while being **resource-efficient**.

Dataset & preprocessing

- Use `torchvision.datasets.EMNIST` (choose split: `byclass` (62)).
- Create a train/val split from the training set (e.g., 90/10) and keep the original test set for final results.
- Preprocessing notes:
  - EMNIST orientation: standard correction is **rotate −90° then horizontal flip**.
  - Convert grayscale → 3-channel (ResNet expects 3).
  - Resize to 224×224; normalize with ImageNet mean/std.

Model & training

- Start from **ResNet-18** pretrained on ImageNet.
- Replace the final FC layer to match your `num_classes`.
- Explore efficiency techniques (pick at least two):
  - **Linear probe** first (freeze backbone), then selective unfreeze (e.g., last block), then full fine-tune.
  - **Mixed precision (AMP)** if CUDA is available.
  - Smaller batches/epochs + early stopping or cosine schedule with warmup.
  - Light data augmentation (e.g., small rotations, modest resized crops).

What to report

- Final **test metrics**: accuracy, precision, recall, F1 (macro); include a confusion matrix (reduced view is fine).
- **Efficiency accounting** (see rubric):

- GPU/CPU type, **epochs**, effective train samples/epoch, trainable parameter count, and wall-clock training time.
- Brief ablation: compare at least two regimes (e.g., linear probe vs. partial unfreeze) and discuss the **accuracy/compute** trade-off.

Efficiency Score (lightweight; supports the rubric)

We will consider both performance and compute. Include the following so we can compare fairly:

- `Accuracy`: Test set top-1.
- `Trainable Params (M)`: number of parameters with `requires_grad=True`.
- `Epochs × Samples_per_Epoch`: total update budget (state if you used early stop).
- `Hardware & Time`: device model and total train time.

> There is **no single formula**—we grade holistically (see rubric)—but clear, honest accounting is required.

Deliverables & Submission

Code (you may submit a zipped folder containing the below-mentioned files)

- **Task 1**
  - `cnn_from_scratch.py` — your implementation and a tiny MNIST training run
  - SBATCH files - .SBATCH script, error file, and output file used to run `test.py` and `cnn_from_scratch.py`
- **Task 2**
  - `resnet18.py` — training script for EMNIST fine-tuning (arguments or config dict OK)
  - SBATCH files - .SBATCH script, error file, and output file used to run `resnet.py`
    Repository requirements
    Please make sure that the zipped folder also contains the following:
- A very simple `README.md` / `pdf` with exact run commands that you used.
  Note
- Make runs deterministic where possible (set random seeds consistently)

Report (PDF, ≤4 pages + figures)

1. **Task 1 internals**
   - How your **Convolution** is implemented (im2col/unfold view, weight/bias grads, fold back)
   - How your **MaxPool** forward/backward works (argmax scatter)
   - How your **Training Loop** works
   - **Screenshots**: tests passing; a short MNIST training plot demonstrating progress
2. **Task 2 methodology**
   - Data splits & preprocessing (including EMNIST orientation fix)
   - Training regimes tried and the rationale (what layers did you freeze, when did you unfreeze, AMP, aug, schedulers)
   - **Optimizations** that helped most (with a sentence on “why”)
   - **Results**: train/val curves; final **test**: accuracy, precision, recall, F1; confusion matrix
   - **Efficiency accounting** table (hardware, epochs, samples/epoch, trainable params, total time)
   - A **brief takeaway**: the best accuracy/compute trade-off you achieved

NOTE:

- Please keep the textual content together at the beginning of the document. The images should be at the end of the document. You may refer to the images by indicating them when required as Fig. 1, Fig. 2, etc.
- The textual content should be under 2 pages.
- When you are asked to explain how something works, explain with clear step-by-step logic OR a mathematical explanation, OR a dry run of the code simulating the same behavior (in case you wish to show a dry run, you may use a screenshot of your workings for the same).
- Please also include screenshots at the end of your report indicating that you have run the code on Google Cloudburst. You can do this by running an SBATCH file and showcasing the timestamps of the output and error logs generated by the SLURM job
  (use command `ls -l` in directory containing the output and error logs)
- In case you are struggling to run the code on singularity, you may contact me via email, or during the TA's office hours. In the meantime, you may start working on Google Colab pro (students get 1 year access to Colab Pro) and then move your code into Google CloudBurst.
- In case you wish to code in VSCode locally, I would recommend creating a **PRIVATE** github repository and cloning the repository into Google CloudBurst as well as your local machine. This way you can edit on your local machine, push the commits to GitHub, and see the reflected changes in Google CloudBurst when you do a `git pull`

Grading Rubric (100 pts)

- **Task 1 correctness & tests (35 pts)**
  - Pass layer and E2E tests (25)
  - MNIST training shows learning (loss/acc) with sensible code structure (10)
- **Task 2 performance & methodology (35 pts)**
  - Clean data pipeline, proper EMNIST handling, correct model surgery (10)
  - Solid validation protocol; final test metrics reported (10)
  - Clear ablation(s) with conclusions (15)
- **Efficiency & reproducibility (20 pts)**
  - Transparent compute accounting; at least two efficiency techniques attempted (12)
  - Seeds/env specified; runs are reproducible from README (8)
- **Report quality (10 pts)**
  - Concise, well-organized, with the requested visuals & explanations

---

Constraints & Policies

- **Libraries**: PyTorch/torchvision, NumPy, scikit-learn (metrics), matplotlib. For Task 1, do **NOT** use autograd for your parameters; you must compute grads manually.
- **Academic integrity**: Write your own code. Cite any external snippets/ideas. Keep your repo private until grading is complete.

- **Environment**: CPU or GPU acceptable; note device type in your report.

How to Run (example)

```
# Task 1: run tests then a short MNIST train
(please make sure to import the libraries correctly from your cnn_from_scratch program in tests.py)
python tests.py
python cnn_from_scratch.py

# Task 2: one sensible default run (feel free to add args/flags)
python resnet18.py
```