Homework 3: Generic Training Pipeline in PyTorch

Cioata Ioana-Larisa, MIAO2

1 Introduction

This paper describes a configurable, device-agnostic PyTorch training pipeline that supports all datasets, models, optimizers, and learning-rate schedulers required by the assignment. The pipeline also includes data augmentation, a dynamic batch-size mechanism, and early stopping. Most of the experiments to be presented were performed on the CIFAR-100 dataset.

2 Setup and How to Run

All experiments were run in Google Colab. The only setup needed is selecting a GPU runtime (this is actually an optional step, but it speeds up the training).

Setup

1. Open the Colab notebook at the following link: `https://colab.research.google.com/drive/1eOg3LSnoebzzyr9tKjdEv2GpT4z5gJ4B?usp=sharing`
2. Enable GPU support: `Runtime` → `Change runtime type` → `T4 GPU`

How to run

3. Run all cells in the notebook: `Runtime` → `Run all`. The first cells install the required libraries and set up the environment.
4. At the end, WandB prints a command like:

   ```
   wandb: Run sweep agent with: wandb agent <USERNAME>/<PROJECT>/<SWEEP_ID>
   ```

   Run it in a new code cell with an exclamation mark:

   ```
   !wandb agent <USERNAME>/<PROJECT>/<SWEEP_ID>
   ```

5. To run a custom (non-sweep) configuration, edit `configs/config.yaml` and rerun the notebook.

Alternative

The pipeline can also be launched using command-line arguments directly in Colab:

```
!python main.py --dataset=CIFAR100 --model=resnet50 --model_pretrained=True
--optimizer=sgd --lr=0.01 --scheduler=steplr --batch_size=96 --weight_decay=0.0005
--momentum=0.9 --loss=crossentropy --augment=new --mixup_alpha=0.5
--cutmix_alpha=0.5 --mix_prob=0
```

3 Pipeline Overview

3.1 Supported Datasets

The pipeline supports the following models: MNIST, CIFAR-10, CIFAR-100, and Oxford-IIIT Pet.

3.2 Supported Models and Model Adjustments

The pipeline supports several convolutional models from `timm` (resnet18, resnet50, resnet14d, resnet26d), together with a custom MLP.

Since the models from `timm` are designed for ImageNet (224×224), they are not optimal for low-resolution datasets like CIFAR-10, CIFAR-100 or MNIST. To handle this, the pipeline automatically modifies the first layers of the model:

- replaces the original 7×7, stride-2 convolution with a 3×3, stride-1 convolution;
- removes the initial max-pooling layer;
- for models with a `stem` block (e.g. ResNet-D), both `stem.conv1` and `stem.pool` are adapted similarly.

The following additional adjustments specific to the dataset were also made:

- **MNIST** (grayscale): images are kept as single-channel inputs for the MLP, and converted to three channels for convolutional models.
- **Oxford-IIIT Pet**: images have variable resolutions and are resized to 224×224 to match the expected input size of pretrained backbones.

Other features: **MLP architecture.** The MLP is a simple feed-forward network applied to flattened images:

$$\text{Flatten} \rightarrow \text{Linear}(\text{input\_dim}, 512) \rightarrow \text{ReLU} \rightarrow \text{Linear}(512, \text{num\_classes}).$$

Here, `input_dim` is channels × height × width (e.g. 3×32×32 for CIFAR, 1×28×28 for MNIST).

Other features: **Class count and shape inference.** The number of classes is determined automatically from the dataset object. A sample training image is passed to the model builder to infer the correct input shape before building the network.

Pretraining support. If enabled, models are loaded with pretrained ImageNet weights. The adjustments for small images work in both pretrained and non-pretrained settings.

3.3 Optimizers and Learning-Rate Schedulers

The pipeline supports several optimizers: SGD, Adam, AdamW, Muon, and SAM.

Muon separates parameters into matrix-shaped tensors updated by Muon and vector-shaped tensors updated by Adam. SAM operates on top of a base optimizer, which in my experiments was SGD.

The Muon optimizer is loaded from its official GitHub repository (`https://github.com/KellerJordan/Muon`), using a direct `pip` installation. For SAM, the official PyTorch implementation from `https://github.com/davda54/sam` is included in the project as `sam.py`.

The schedulers used in the experiments were `StepLR` and `ReduceLROnPlateau`.

3.4 Other Training Components

The pipeline includes several additional training features that improve stability and efficiency. Each dataset comes with its own augmentation presets (defined in `datasets.yaml`), and optional batch-level augmentations such as Mixup and CutMix can be enabled when needed. The training loop also supports dynamic batch size adjustments during training, as well as mechanisms to automatically terminate runs that stop improving.

4 Hyperparameter Sweep

Several WandB sweeps were used throughout the project, each focusing on different parts of the search space. The sweeps were run only on CIFAR-100, while the full pipeline supports a larger set of components.

The experiments explored combinations of:

- **Models:** ResNet18, ResNet50, ResNet14d, ResNet26d, and MLP.
- **Optimizers:** SGD, Adam, AdamW, Muon, SAM.
- **Schedulers:** StepLR, ReduceLROnPlateau.
- **Loss functions:** cross-entropy with label smoothing.
- **Augmentation presets:** all presets defined per dataset in `datasets.yaml`.
- **Batch-level augmentations:** Mixup and CutMix (probability and α values varied).
- **Weight decay:** explored as a regularization hyperparameter.
- **Momentum:** varied where applicable (e.g. for SGD).
- **Pretraining:** both pretrained and non-pretrained variants for convolutional models.

All experiments ran for at most 100 epochs. Mixed-precision training was enabled on GPU/MPS, and data loading used pinned memory and persistent workers. The initial batch size was 96 and increased every 5 epochs by a factor of 1.5, up to a maximum value (486 in the CIFAR-100 experiments). The trainer also supports early stopping, with a patience threshold of 4.

- **StepLR:** step size of 5 epochs and decay factor $\gamma = 0.5$.
- **ReduceLROnPlateau:** `mode = max`, factor 0.3, patience 5, threshold 10^{-4}.

The evaluation metric optimized was the `test_acc`.

5 Results on CIFAR-100

5.1 Top Configurations on CIFAR-100

All runs were trained under the same global setup: SGD with momentum 0.9 and weight decay 5×10^{-4}, 100 epochs, dynamic batch sizing (starting from 96), pretrained models, label smoothing 0.1, and the CIFAR-100 *new* augmentation preset. This preset combines spatial transforms and photometric noise: `RandomCrop`(32, padding=4), `RandomHorizontalFlip`(p=0.5), `ColorJitter`(0.2, 0.2, 0.2, 0.02), `RandomRotation`(10°), `RandomGrayscale`(p=0.1), and `RandomErasing`(p=0.25). Mixup and CutMix were included with fixed $\alpha = 0.5$, while the sweeps varied only the application probability `mix_prob`.

The table below lists the best 8 configurations exceeding 70% accuracy.

Model	LR	MixProb	Runtime	Test Acc.
resnet50	0.05	0.5	1h 32m 22s	86.31%
resnet50	0.05	0.0	1h 14m 57s	85.74%
resnet18	0.1	0.0	22m 44s	81.6%
resnet18	0.1	0.5	31m 47s	80.16%
resnet18	0.05	0.0	17m 3s	78.76%
resnet50	0.01	0.0	1h 12m 29s	77.99%
resnet18	0.05	0.5	25m 58s	75.69%
resnet50	0.1	0.5	25m 52s	75.34%

Table 1: Top-performing sweep configurations on CIFAR-100.

Sweep link. The sweep used in these experiments is available at `https://wandb.ai/ioana-cioata-universitat homework3/sweeps/o37ayylh`. (I was not able to make the sweep public due to workspace restrictions, but the link is included as a reference for the configurations reported in the table above.)

Pipeline efficiency was assessed through training-time measurements. Dynamic batch sizing plays an important role here: by starting with a smaller batch (96) and increasing it gradually during training, the number of iterations per epoch decreases, which directly reduces the epoch runtime. This effect is visible in Figure 1, where later epochs become consistently faster as the batch size grows.

Data loading also contributes to efficiency. Using four workers together with pinned and persistent memory improves CPU–GPU device transfer speed.

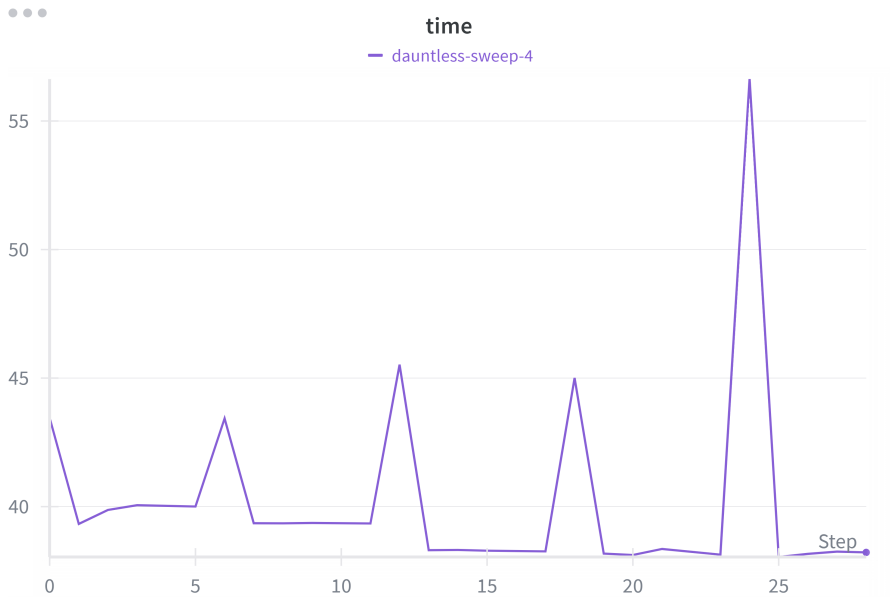

Figure 1: Training time per epoch measured on CIFAR-100 during the sweep.

6 Pretraining vs No Pretraining

6.1 Training from Scratch

Most configurations trained entirely from scratch did not surpass the 70% accuracy threshold, so no runs appear in the shortlist of top results.

6.2 Using Pretrained Weights

Pretrained models consistently achieved higher accuracy than models trained from scratch. This is most evident in the parallel-coordinates visualization in Figure 3: nearly all trajectories entering the upper performance region pass through the `resnet18` or `resnet50` nodes together. The same pattern appears in Figure 2, where the curves exceeding the 70% test-accuracy threshold correspond exclusively to pretrained models.

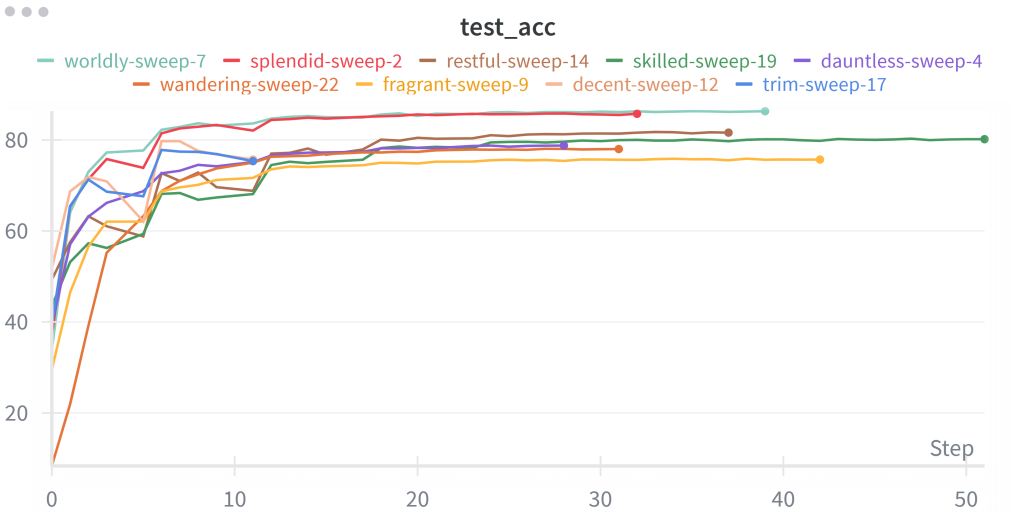

Figure 2: Test accuracy trajectories

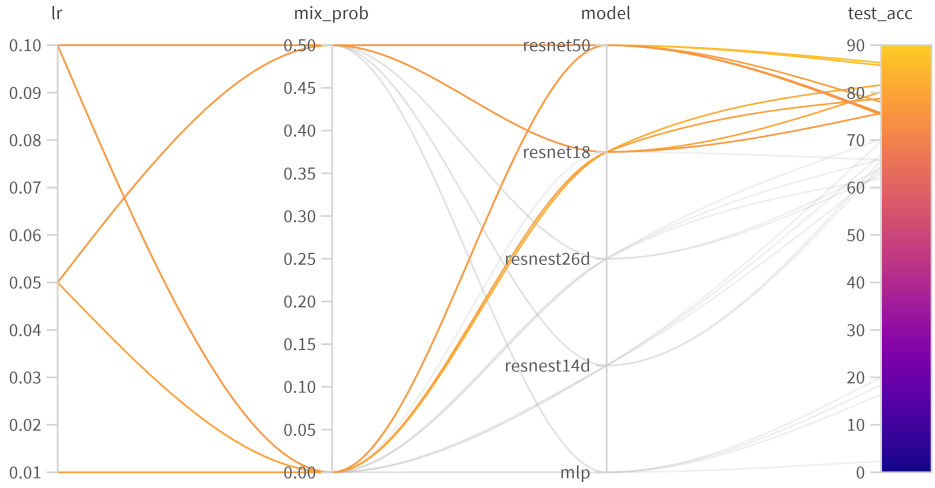

Figure 3: Parallel-coordinates visualization of hyperparameters vs. test accuracy

Across all experiments, the highest result was obtained with a pretrained ResNet50 at learning rate 0.05 and Mixup/CutMix probability 0.5, reaching a test accuracy of **86.31%**.

7 Self-Evaluation

- First feature group: **8 points**.
- Second feature group: **8 points**.
- Third feature group: **2 points**.
- Fourth feature group: **3 points**.