ATNN - Homework 3 - Report

Marin Andrei-Vasile

November 23, 2025

1 Pipeline Features

The training pipeline is designed to be flexible and easily configurable for a variety of experimental settings. It is fully device-agnostic, enabling execution on any available compute accelerator, including CPU and GPU, without requiring changes to the training code. Device selection is automatically handled at runtime based on user configuration and hardware availability.

1.1 Datasets and Augmentation

The pipeline supports four datasets: **MNIST**, **CIFAR-10**, **CIFAR-100**, and **Oxford-IIIT Pet**. Each dataset is integrated with a consistent transformation interface, allowing the user to enable or disable a variety of data augmentations. The following image transformations are available:

- *RandomHorizontalFlip*
- *RandomCrop*
- *ColorJitter*
- *RandomErasing*

These augmentations are applied conditionally based on the selected configuration. The datasets are also normalized (when needed) with the precomputed values of the mean and standard deviation of each channel.

1.2 Model Support

The pipeline provides support for several, namely `resnet18`, `resnet50`, `resnest14d`, `resnest26d`, and a simple `MLP`. All convolutional architectures are imported from the `timm` library and may optionally utilize pretrained ImageNet-1k weights. This enables both from-scratch training and transfer-learning scenarios.

1.3 Optimizers and Hyperparameters

A variety of optimizers are available, each with configurable hyperparameters. All optimizers support user-defined learning rate and weight decay. The following optimization algorithms can be selected:

- **SGD** with optional momentum and Nesterov acceleration
- **Adam**
- **AdamW**
- **Muon**
- **SAM**, combined with a user-chosen base optimizer (either SGD, Adam or AdamW)

SAM is integrated using a two-step optimization procedure and inherits the hyperparameters of its underlying optimizer.

1.4 Learning Rate Scheduling

Two learning rate scheduling strategies are included:

- **StepLR**, configured by a decay step interval and multiplicative factor (γ)
- **ReduceLROnPlateau**, which monitors validation loss and reduces the learning rate when no improvement is observed for a specified patience value

1.5 Batch Size Scheduling

The pipeline incorporates a simple linear batch size scheduler. The batch size is increased gradually from an initial value to a user-specified maximum between a configurable start and end epoch. The scheduler updates the batch size only at a specified epoch interval, reducing the need for frequent DataLoader reinitialization and thus minimizing overhead.

1.6 Experiment Tracking and Early Stopping

Integration with Weights & Biases (WandB) enables real-time logging of performance metrics, including:

- Training accuracy and loss
- Validation accuracy and loss
- Training and validation VRAM usage (MB)

The pipeline also includes an early stopping module that terminates training when either validation loss or validation accuracy doesn't improve for a specified patience period.

1.7 Test-Time Augmentation

During inference, the pipeline applies a structured test-time augmentation (TTA) procedure to improve prediction robustness. Each image is augmented using **eight spatial translations** together with an additional version of the original (unshifted) input. For each of these nine variants, a horizontally flipped counterpart is also generated. This results in a total of $18 = (8 + 1) \cdot (1 + 1)$ separate forward passes through the neural network. The individual outputs are then aggregated to produce a single final prediction, reducing sensitivity to image alignment and enhancing classification stability.

2 Training Time Efficiency

The pipeline incorporates several mechanisms aimed at reducing computational overhead and improving overall training throughput. First, cuDNN benchmarking is enabled, allowing the backend to automatically select the most efficient convolution algorithms for the given input shapes. Once the optimal kernels have been identified, subsequent iterations benefit from significantly faster execution.

To further accelerate training, the DataLoaders can be configured with multiple worker processes, which persist across epochs. This eliminates the cost of respawning workers at the beginning of every epoch and improves data-loading parallelism. Additionally, pinned memory is enabled, allowing host-to-device data transfers to be performed asynchronously. Combined with the use of `non_blocking=True` when moving tensors to the desired device, this minimizes input pipeline latency by overlapping data transfer with computation.

A caching mechanism is applied to some of the data transformations. For the training set, image resizing and conversion to `float32` tensors in the $[0, 1]$ range are cached to avoid repeated preprocessing. For the validation set, the same caching strategy is applied, additionally including normalization and the padding and flipping transformations required for test-time augmentation. This substantially reduces transformation overhead during repeated evaluations.

The batch size scheduler is also designed with efficiency in mind. The batch size increases only at a reduced frequency between the configured start and end epochs, preventing frequent DataLoader reinitialization. As a result, worker processes are preserved for longer intervals, and cuDNN benchmarking is not repeatedly re-triggered, both of which contribute to more stable and faster epoch transitions.

3 Experiments

A series of experiments were conducted to evaluate the performance of the training pipeline on the CIFAR-100 dataset. Among these, one configuration surpassed the validation accuracy threshold of 70%. This experiment achieved a final validation accuracy of **72.16%** at epoch 74. The full configuration used for this run is summarized below.

Dataset Configuration

- **Workers:** 3 parallel DataLoader workers
- **Augmentations:**
  - Random horizontal flip: enabled
  - Random crop: enabled
  - Color jitter: disabled
  - Random erasing: enabled

Model Configuration

- Architecture: `resnest26d`
- Pretrained weights: disabled

Optimizer Configuration

- Optimizer: SGD
- Learning rate: 0.0003
- Weight decay: 0.005
- Momentum: 0.99
- Nesterov acceleration: enabled

Scheduler Configuration

- Scheduler: StepLR
- Step size: 20 epochs
- Multiplicative factor (γ): 0.3

Training Configuration

- Total epochs: 80
- Initial training batch size: 64
- Validation batch size: 500
- Device selection: automatic (accelerator if available, in this case GPU)
- Early stopping: monitors validation accuracy with a patience of 10 epochs

Batch Size Scheduling

A linear batch size scheduling policy was applied:

- Policy: linear increase
- Maximum batch size: 128
- Schedule active from epoch 30 to 80
- Batch size updated every 10 epochs

This strategy is intended to improve computational efficiency in later stages of training while reducing noisy gradient estimates in the early phase.

Results

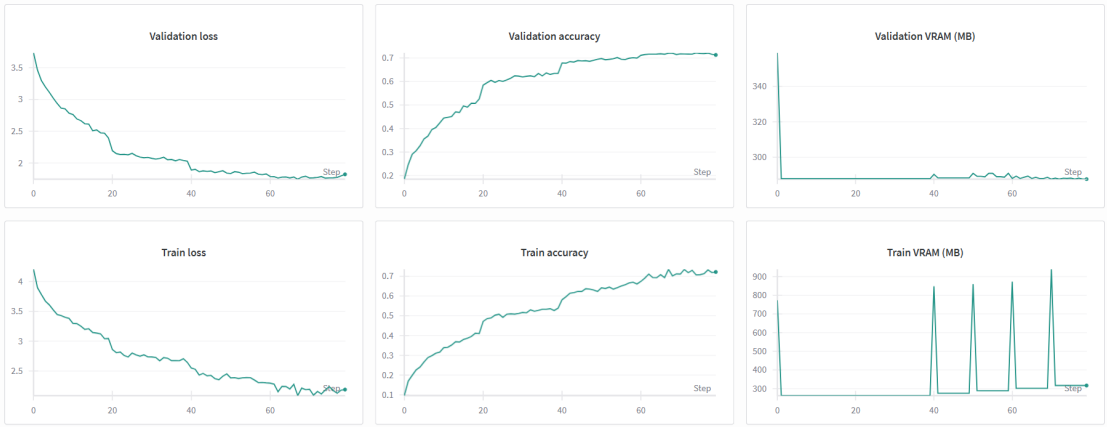

Figure 1: Weights & Biases metric analytics

4 Self Evaluation

Features. The pipeline implements all the requested functional components, including dataset support, augmentation options, model configurations, optimizer and scheduler variants, early stopping, batch size scheduling, and Weights & Biases integration. For this reason, the full **8 points** should be awarded for the features section.

Experiments. Although the experiments were not conducted using a systematic hyperparameter sweep, one of the runs surpassed the required performance threshold of 70% validation accuracy on CIFAR-100, achieving a maximum of 72.16%. Therefore, this section merits **1 point out of 8**.

Efficiency. The pipeline incorporates several mechanisms aimed at improving runtime efficiency, including caching of basic transformations, pinned memory and non-blocking transfers, persistent DataLoader workers, reduced batch size update frequency, and the use of cuDNN benchmarking. While these features meaningfully enhance performance, quantitative metrics were not reported. As a result, the efficiency section should receive **2 out of 3** points.

Total Score. Aggregating the section-wise evaluations, this homework should receive a total of **11 points** out of 25.

5 How to Run

Configuration

The pipeline is fully configured through a YAML file located at:

```
configs/current_config.yaml
```

All settings for the dataset, model architecture, optimizer, scheduler, training procedure, and augmentations are controlled through this file.

Weights & Biases (Optional)

If experiment tracking via Weights & Biases (WandB) is enabled in the configuration file, the user must authenticate prior to running the pipeline. This can be done by executing:

```
wandb login
```

The pipeline will automatically initialize WandB and log metrics, configurations, and system information.

Running the Training Script

To start training, simply execute:

```
python train.py
```

No additional arguments are required, as the script automatically loads all necessary parameters from the configuration file.