Experiment Usage Guide

Ravindra Kumar Saini
Missouri State University

November 7, 2025

This document provides a complete usage guide for the experimental image classification framework. It helps users train, evaluate, and analyze both deep learning and topology-enhanced models with this framework.

1. Project Overview

The framework provides the following capabilities:

- Train deep learning models: `cnn`, `resnet18`, and `vgg11`.
- Apply topological preprocessing via the **Persistence Landscape Layer (PLLay)**.
- Conduct robustness testing using Gaussian noise perturbations.
- Compare results with classical machine learning baselines:
  - HOG or PCA for feature extraction.
  - SVM or Random Forest for classification.
- Visualize and analyze topological feature spaces.

Core scripts in the framework:

- `train.py` – Train models.
- `evaluate.py` – Evaluate saved checkpoints and generate reports.
- `classical_ml.py` – Run classical ML baselines (SVM, RF).
- `analysis.py` – Aggregate and compare performance results.

2. Installation

Required dependencies:

```
pip install torch torchvision torchaudio matplotlib seaborn pandas tqdm scikit-learn
    scikit-image gudhi
```

Note: The `gudhi` library is required only for true topological computations. If it is not installed, the framework uses a learned approximation of the persistence features.

3. Datasets

Supported datasets are automatically downloaded via `torchvision.datasets`:

- MNIST
- Fashion-MNIST
- CIFAR-10

4. Training Deep Models

Models can be trained from the command line as follows:

```
python train.py --model MODEL_NAME --dataset DATASET_NAME \
  --epochs 20 --batchSize 128 --lr 1e-3 \
  --weightDecay 1e-4 --outDir outputs --tag my_experiment
```

4.1 Key Arguments

Argument	Description	Values / Examples
`--model`	Type of model to train.	`cnn`, `resnet18`, `vgg11`, `cnn_topology`, `resnet18_topology`, `vgg11_topology`
`--dataset`	Dataset to use.	`mnist`, `fashion`, `cifar10`
`--epochs`	Number of training epochs.	e.g., `10`, `20`
`--batchSize`	Mini-batch size for training and evaluation.	e.g., `64`, `128`
`--lr`	Learning rate for the optimizer.	e.g., `1e-3`
`--weightDecay`	Weight decay (L2 regularization).	e.g., `1e-4`
`--useGudhi`	Enable true topological computation via Gudhi.	flag (no value)
`--subsetSize`	Limit training to the first `N` samples (useful with Gudhi).	e.g., `10000`
`--evalNoise`	Noise type applied during evaluation.	`gaussian`
`--noiseSigma`	Standard deviation σ for Gaussian noise.	e.g., `0.3`
`--testRobustness`	Run systematic robustness test over multiple noise levels.	flag (no value)
`--robustnessLevels`	Number of noise levels between 0 and `--noiseSigma`.	e.g., `6`

4.2 Example Commands

Train a simple CNN on MNIST:

```
python train.py --model cnn --dataset mnist --epochs 15 \
  --batchSize 128 --lr 1e-3 --weightDecay 1e-4 \
  --outDir outputs --tag mnist_cnn
```

Train ResNet18 with topology on CIFAR-10:

```
python train.py --model resnet18_topology --dataset cifar10 \
  --epochs 10 --batchSize 64 --lr 1e-3 --useGudhi \
  --subsetSize 10000 --outDir outputs --tag cifar10_resnet_topology
```

Train CNN with noise robustness testing (Fashion-MNIST):

```
python train.py --model cnn --dataset fashion --epochs 10 \
  --evalNoise gaussian --noiseSigma 0.3 --testRobustness \
  --robustnessLevels 6 --outDir outputs --tag fashion_cnn_robust
```

5. Evaluation of Trained Models

You can re-evaluate trained models using:

```
python evaluate.py --model MODEL_NAME --dataset DATASET_NAME \
  --checkpoint PATH_TO_CHECKPOINT --outDir eval_outputs --tag eval_run
```

This script supports:

- Optional Gaussian noise during evaluation.
- Systematic robustness analysis with multiple noise levels.
- Topological feature visualization for topology-enabled models.

6. Classical ML Baselines

Run classical machine learning baselines for comparison:

```
python classical_ml.py --dataset mnist --feature hog \
  --clf svm --max-samples 20000 --outDir classical_outputs
```

Feature extraction options:

- `hog` – Histogram of Oriented Gradients.
- `pca` – Principal Component Analysis.

Classifier options:

- `svm` – Support Vector Machine.
- `rf` – Random Forest.

7. Cross-Run Analysis

Compare results across multiple experiments:

```
from analysis import collectResults, createComparisonPlots

df = collectResults("outputs")
createComparisonPlots(df, "plots")
```

Generated artifacts:

- `metrics.json` / `metrics.pt` – Stored metrics for each run.
- `confusion_matrix.png` – Confusion matrix for test predictions.
- `robustness_plot.png` – Accuracy vs. noise level (if robustness testing is enabled).
- `topology_features.png` – Visualization of topological representations (for topology models).