Graded Assignment 2

Test-time Adaptation and Multiple Instance Learning

Prepared by Eeshaan Jain

Overview

Introduction. In this two-part assignment, you will **implement and investigate** (a) test-time adaptation techniques for improving generalization to new and diverse data during inference, and (b) downstream evaluation methods on histopathological datasets. The assignment builds directly on concepts covered in the lectures and exercise sessions.

Due date: **Wednesday 3rd December, 2025, 23:59 CET**
Compute platform: `http://gnoto.epfl.ch`. Your code should be located on Gnoto, as we will scrap it from there.
Note:

- Your submission directory should only contain the base code files given along with the aforementioned file requirements. In case you clone or download other sources which are not used, they need to be removed prior to submission.
- Whenever you dump weights, you should create a folder called `ckpts/` and save the weights inside it.

1 Part I: Test-Time Adaptation

Model accuracy typically degrades when the training (source) data differ from the testing (target) data—a phenomenon known as dataset/domain shift. Models are often sensitive to shifts that were not observed during training, whether due to natural variations or synthetic corruptions. Nonetheless, deploying models in real-world settings requires robustness to such distribution changes, making test-time adaptation essential.

In this part, you will work with corrupted versions of the CIFAR-10 dataset. The aim is to implement different TTA algorithms on the CIFAR-10 dataset, and choose the best one that will be evaluated on our benchmark. Your implementation should aim to improve over the `Unadapted` baseline (see below) significantly.

Section 1. Dataset Setup.

For this part of the assignment, you are provided with two datasets:

1. **Exploratory dataset:** 19 corruption types at three severity levels, totaling 570k unlabeled images. You may use this dataset in an **unsupervised** manner for developing and analyzing your adaptation strategies.
2. **Public test bench:** 6 corruption types, totaling 190k images. This dataset is for evaluating your method and comparing performance against the provided baselines.

Important!

- During grading, your model will be evaluated on a **different hidden test bench**.

Section 2. Model Implementation.

You are provided with a codebase for implementing your Test-Time Adaptation (TTA) method. Follow these guidelines carefully:

1. **Main Implementation File**
   - Implement your TTA method in `tta/submission.py` inside a class named `Submission`
   - **Do not change** the class name or its interface – the automatic evaluation system depends on this
   - Initialize all method-specific parameters and components in the `__init__` constructor
2. **Configuration Files**
   - `configs/base_config.yaml`: Contains general settings (dataset class, source model path, evaluation parameters)
   - `configs/unadapted.yaml` and `configs/norm.yaml`: Two baseline implementations provided for reference
   - `configs/submission.yaml`: Add any method-specific arguments here (these will override base configuration settings)
3. **Dataset**
   - You may create your own dataset class for development purposes.
   - `/shared/CS461/cs461_assignment2_data/part1/data/` is the dataset directory. It contains the exploratory dataset, public test bench, along with the training set of CIFAR10. Hence, if you wish to load the CIFAR10 training set, you can set the root directory as `CIFAR10(root=cfg.dataset.args.dataset_path)`.

- **Important:** The grading will use the implemented `TestTimeAdaptationDataset`.

4. **Source Model**
   - Pre-trained ResNet50 on CIFAR-10 (available by default).
     Checkpoint path: `/shared/CS461/cs461_assignment2_data/part1/ckpts/pretrained_cifar10_resnet50.pt`
   - **If you choose to retrain:** You may retrain your own ResNet50 only using the CIFAR-10 training data along with the exploratory dataset. In this case, you **must** submit the new model parameters. Update the `model.ckpt_path` in `submission.yaml` to point to your checkpoint.
   - **Required:** Use ResNet50 architecture only – other architectures will result in grade deductions. Avoid any train-test leakage, violations will result in grade deductions.
5. **Provided Baselines:** Two baseline implementations are provided for reference:
   - **Unadapted:** No adaptation at test-time,
   - **TestTimeNorm:** Uses batch statistics during test-time.
6. **Testing Your Implementation:** You can test your implementation using `python -m eval m1 m2 ...` where m1, m2 etc. are names of the configuration files (`m1.yaml` etc.). To test the baselines, you can run `python -m eval`. You have already been provided with the results of the baselines, but you can recompute them by setting `force_recompute=True` inside the `eval.py` file. Results of runs are auto-cached, so you can set it to False to avoid recomputation.

Section 3. Submission Instructions.

Your submission directory must contain

- `tta/submission.py`: your TTA implementation,
- `configs/submission.yaml`: your TTA configuration,
- Any other files which you implement and the model weights (if any)
- `report.md`: short report describing your approach (max. 500 words). You should also add a section called "**Reproduction of results**" where you need to mention the scripts to run to train your implementation if it requires training (not included in the word limit).

inside the directory `/home/cs461_assignment2_submission/part1`. We will scrape them after the deadline.

---

2 Part II: Downstream evaluations on histopathological datasets

Whole Slide Images (WSIs) are extremely large, making it computationally infeasible to process them directly using deep neural networks. As a result, WSIs are commonly divided into

smaller patches that can be processed individually. In this part of the assignment, you will work with such pre-processed WSIs for downstream cancer subtype classification. The aim of the implementation should be to improve over the linear baseline (see below).

Section 1. Dataset Setup.

You are provided with an anonymized dataset of H&E-stained WSIs. Each patient's WSI varies in size and has been pre-processed into patches. Every patch has already been embedded using a pretrained H&E foundation model. Thus, for each patient, you have been given the set of embeddings $z_P \in \mathbb{R}^{N_p \times d}$, where n_P (the number of patches) varies per patient along with the corresponding downstream label y_P. Note that the class distribution is imbalanced. You may use the **entire provided dataset** for training your downstream model. The dataset is present at `/shared/CS461/cs461_assignment2_data/part2/data/`

Important!

- During grading, your model will be evaluated on a hidden test bench.

Section 2. Model Implementation.

Implement your classification method for histopathology patch analysis. Follow these guidelines carefully:

1. **Main Implementation File:**
   - Implement your method in `models/submission.py` inside a class named `Submission`
   - **Do not change** the class name – the automatic evaluation system depends on this
   - Implement a `load_weights` method in your Submission class to load trained model parameters
2. **Configuration Files**
   - Create `configs/submission.yaml` with your method settings
   - **Must define**: `model.best_weight_path` pointing to your saved model weights
   - See the baseline configuration for the expected structure
3. **Dataset**
   - You may create a custom dataset class for training
   - **Important:** Evaluation will use the provided `ImageDataset` class only
4. **Data Collation**
   - You must write your own `collate_fn` function
   - **Expected format:** If your collate function returns n items, items 1 through $n-1$ are passed as inputs to the model. Item n is treated as the label. Example: `return images, metadata, labels`.

- You need to update the `dataset.collate_fn` key in your `submission.yaml` to point to the collate function implementation.

5. **Model Architecture Guidelines**
   - **Allowed:** (1) Any model architecture you design from scratch, and train on the provided dataset, (2) **PyTorch** is strongly recommended
   - **Not Allowed:** (1) any pre-trained models, (2) external datasets
   - If using non-PyTorch libraries, wrap your model with a `forward` method, and set the correct device type in the `evaluation` key inside your `submission.yaml`. Make sure your implementation then works with the `eval.py` file.
6. **Model Weights**
   - **Important:** Save your trained model weights inside the directory itself
   - Update the `model.best_weight_path` inside `configs/submission.yaml` to point to your weights file
7. **Provided Baseline**
   - A linear probing based method has been provided as a baseline, which averages patch embeddings per patient and uses a linear classifier, trained with 5-fold cross-validation.
   - Both the training and model code are available inside `models/linear_baseline.py`.
   - Use this as a reference for structure and expected interfaces
8. **Testing your Implementation:** `python -m eval m1 m2 ...` where m1, m2 etc. are names of the configuration files (`m1.yaml` etc.). To evaluate the baseline on the training set, you can run `python -m eval`. You have already been provided with the results of the baseline, but you can recompute them by setting `force_recompute=True` inside the `eval.py` file. Results of runs are auto-cached, so you can set it to False to avoid recomputation. You will be evaluated on the Macro F1-score.

Section 3. Submission Instructions.

Your submission folder must contain

- `models/submission.py`: your downstream classifier implementation,
- `configs/submission.yaml`: your model configuration,
- The model weights
- `report.md`: short report describing your approach (max. 500 words).

at `/home/cs461_assignment2_submission/part2`. We will scrape them after the deadline.

Good luck! And please, double-check your submission!