CSCE 636: Project Description

Due Date
11:59pm, April 22, 2024
Late Submissions Are Not Accepted

1 Overview

In this project, you will design and implement your own deep learning model to perform **10-class image classification** on the given dataset. You will be able to access the training data to train and tune your model, and a public test dataset for the evaluation before your final submission. Your model will be finally evaluated on a private test dataset.

2 Project Instructions

What You Are Expected to Do: You are expected to explore the state-of-the-art of deep learning based image classification, propose and implement your own deep learning models. You can explore everything including

- Novel neural network architectures, novel operations, blocks and modules,
- Data pre-processing, normalizations and augmentations,
- Training strategies, optimizers, parameter initializations, regularizations, etc.

What You Are NOT Expected to Do:

- In this project, you are required to train your model using ONLY the provided training dataset. Specifically, for the sake of fairness, you are **NOT allowed to use extra image data** for transfer learning or pre-training.
- Do not only apply the existing commonly used network architectures such as ResNet, GoogLeNet, VGG, etc. You need to go one step beyond these well-known architecture.

Deep Learning Framework and Libraries: In this project, you are required to implement your model using PyTorch framework. Please choose a version according to the compatibility of your CUDA driver. Recommended stable versions are

- PyTorch 1.6
- PyTorch 2.1

Please specify the version you use in your report. If you use any other versions, please make sure your code works on one of the three versions. Besides, you are free to use any other common open source libraries in your project.

Training and Testing Dataset You are provided with the CIFAR-10 Dataset for this project. The Dataset is available at https://www.cs.toronto.edu/~kriz/cifar.html. Please carefully read the dataset description and implement your code to load and process the images. Note that you should only use the training dataset containing 50k images for training and validation. The public testing dataset should NOT be used for tuning your model.

The images of the private testing dataset will be released one week prior to the due date. **The images are stored in an *.npy* file in the shape *[N, 3072]*, where N is the number of testing images. You can use the same *parse_record()* function to process the data.** The labels of the private testing dataset will not be available. You will need to run your model prediction on the private dataset and submit the prediction results. Please make sure you have saved your model variables in files after training for your convenience.

Code Files Structure For better code readability, please follow the starter code and organize your codes into the following files during implementation.

- *main.py*: Includes the code that loads the dataset and performs the training, testing and prediction.
- *DataLoader.py*: Includes the code that defines functions related to data I/O.
- *ImageUtils.py*: Includes the code that defines functions for any (pre-)processing of the images.
- *Configure.py*: Includes dictionaries that set the model configurations, hyper-parameters, training settings, etc. The dictionaries are imported to *main.py*
- *Model.py*: Includes the code that defines the your model in a class. The class is initialized with the configuration dictionaries and should have at least the methods "*train(X, Y, configs, [X_valid, Y_valid,])*", "*evaluate(X, Y)*", "*predict_prob(X)*". The defined model class is imported to and referenced in *main.py*.
- *Network.py*: Includes the code that defines the network architecture. The defined network will be imported and referenced in *Model.py*.

Detailed descriptions are provided in the starter code. You can add additional files that define your specific modules, blocks, utility functions, etc.

3 Environment Instructions

Recommended environment

- **System:** Ubuntu 18.04 or 20.04
- **Package Manager:** Anaconda or Mamba.
- **Python version:** Python >= 3.8
- **PyTorch:** PyTorch 1.6 or 2.1

Conda (you should check and understand the correlation between Conda and Anaconda) or Mamba are highly recommended to manage your environment. Please refer to Dependency Hell for the reasons.

Environment setup Conda environment creation example:

```
$ conda create -n csce636 python=3.8
$ conda activate csce636
```

Please make sure you understand what is a conda environment, and the difference between *conda* and *pip*. For example, 'matplotlib' can be installed using *pip* in a conda environment, but it is recommended to use *conda* to install *matplotlib* to avoid possible dependency version conflicts.

```
$ conda install matplotlib # Stable version
$ conda install -c conda-forge matplotlib
    # Latest version: maintained by the community
$ pip install matplotlib # Not recommended
```

PyTorch installation Please choose the version according to the compatibility of your CUDA driver. Specifically, please check your driver version using `nvidia-smi` command, check the compatible cudatoolkit version in this documentation and install the corresponding PyTorch/cuda version. (*Note that you don't need to use your system's CUDA (/usr/local/cuda/). You can install a specific version of CUDA library using the following command in your conda environment, which is another reason of using conda.*)

```
$ conda install pytorch==2.1.2 torchvision==0.16.2 torchaudio==2.1.2
    pytorch-cuda=12.1 -c pytorch -c nvidia # CUDA 12.1
$ conda install pytorch==2.1.2 torchvision==0.16.2 torchaudio==2.1.2
    pytorch-cuda=11.8 -c pytorch -c nvidia # CUDA 11.8
$ conda install pytorch==1.6.0 torchvision==0.7.0
    cudatoolkit=10.2 -c pytorch # CUDA 10.2
$ conda install pytorch==1.6.0 torchvision==0.7.0
    cudatoolkit=9.2 -c pytorch # CUDA 9.2
```

Reference: PyTorch Installation Guide

For more details, please refer to the official documentations of CUDA, Conda, PyTorch, and other libraries.

4 Submission Guidance

Your final submission should include the following files.

1. **Prediction on the private test images.** Please store your prediction results as an array into an *.npy* file named "*predictions.npy*". For each image, **store a vector of the probabilities for the 10 classes** instead of the predicted class. The shape of the saved array is *[N, 10]*, where N is the number of testing images.
2. **Code.** Please put all your code files in a "*code*" folder. Also include a README file that describes how to run your code for training and prediction.
3. **Saved models.** Please keep your trained model that can reproduce the results on the public testing set and predictions on the private testing set. Put the related model files in a "*saved_models*" folder. If the file exceeds the uploading size limit, you can put it on Google Drive and include a share link in the "*saved_models*" folder.
4. **Report.** Describe your proposed method, implementation details and summarize your results on the public testing dataset in your report. You can also report anything that you find interesting. The report should be in *.pdf* format and named as "*report.pdf*". All students are required to use LaTeXfor typesetting the report and the NeurIPS LaTeXtemplate is recommended. You can search to find a template for any recent year.

Please compress all the files above in a single *.zip* file with name "*Firstname_Lastname.zip*".

5 Grading Criteria

The grading of your project will be based on the following criteria.

1. (80 points) Test accuracy on the private dataset.
2. (20 points) Novelty of the your methods; clarity of the report; structure and readability of the code.