Pipeline: Confidence Estimation in Analyzing Optical Coherence Tomography Images with Deep Neural Networks (CEAOCTIDNN)

"Confidence Estimation in Analyzing Optical Coherence Tomography Images with Deep Neural Networks" is the topic which is investigated as well as discussed in the masters thesis by Lennard Korte at the School of Computer Science and Engineering (SCSE) together with the Lee Kong Chian School of Medicine (LKCMedicine) at Nanyang Technological University (NTU) Singapore. This repository contains the entirety of the academic work as well as the foundational code. Below, we present a comprehensive documentation and guide detailing how to use it.

Table of Contents

- Quick Start Guide for Linux
- System Requirements
- Features
    - Logging
    - Checkpoints
- Folder Structure
- Usage
    - Prerequisites / Installation
        - For running locally
        - For running in Docker
    - Training
        - Run locally
        - Run in Docker
    - Testing
    - Data Sampling
    - Configuration
        - Arguments
        - Configuration File
- Customization and Modification
    - Custom Dataset
    - Custom CLI options
    - Update Requirements File
- License

Quickstart Guide for Linux

Note: Training data for OCT are not provided in this repository due to data protection reasons.

1. Clone the Repository to your home directory
2. Add training data under: `./data/datasets/`
3. Install Docker and NVIDIA Container Toolkit
4. execute training and testing

```
$ bash exe/run_train_and_eval_docker.sh
```

System Requirements

```
- Ubuntu 20.04 or later
- NVIDIA Graphics Card
- minimum 24GB RAM
- minimum 100 GB SSD Storage
```

Features

This pipeline facilitates the training and testing of deep learning models with a streamlined approach. It supports extensive customization options for the training process, including adaptable parameters through a configuration file or directly within the code. Features such as checkpoints and logs enable detailed monitoring and analysis. Users can enhance the pipeline's functionality by integrating different model architectures or image transformation techniques. Furthermore, it provides the capability to generate samples of processed images for visual inspection. For comprehensive details, refer to the referenced academic publications or the usage documentation provided below.

Logging

For each iteration of Cross-Validation (CV), a new folder is created with the specified name. Training progress and outputs are logged in `training.log` within the designated directory. Test outcomes are recorded in `test_results.log` in the specified working directory. Enabling additional logging and visualization requires supplying a Weights & Biases (W&B) API key and activating W&B in the configuration settings.

Checkpoints

The model's state, epoch, optimizer, scaler and wand_id are stored at the end of each epoch in `last\_checkpoint.pt`, while checkpoints achieving the highest validation accuracy are retained in `best\_checkpoint.pt`. These are located within `./data/train_and_test/projectspecific/group/name/`.

Note: checkpoints contain:

```
{
'Epoch': epoch,
'Model': self.model.state_dict(),
'Optimizer': self.optimizer.state_dict(),
'Scaler': self.scaler.state_dict(),
'Wandb_ID': self.wandb_id
}
```

Folder Structure

```
CEAOCTIDNN/
│
├── data/                  - holds large data files like training data and DA
│   ├── datasets/          - raw datasets
│   ├── torch_pt_models/   - pretrained model training caching
│   └── train_and_test/    - train and test logs and results Examples
│
├── runs/                 - holds all tests and corresponding configurations
│
├── src/                   - holds all python source code and standard configuration
│   ├── architectures/             - deep learning architectures
│   ├── checkpoint.py              - defines training checkpoints with models, epochs, e
│   ├── config_standard.json       - specifies the standard configuration of applicatior
│   ├── config.py                  - evaluates configurations, arguments and provides Cc
│   ├── data_loaders.py            - dataloader objects for training and testing
│    ├── dataset_preparation.py    - read data structures and assign labels
│    ├── dataset.py                - represent dataset object for dataloader
│    ├── eval.py                   - analysing metrics to assess performance
│    ├── guide_wire.py             - values for ivoct guidewire immitation
│    ├── image_transforms.py       - pre-processing and data augmentation techniques
│    ├── logger.py                 - for logging result files and printing tasks
│    ├── main.py                   - entrance point for application
│    └── utils.py                  - Helper functions and utilities
│
├── .dockerignore         - specifies files Docker should copy to container
│
├── .gitignore            - specifies files or directories Git should ignore
│
├── Dockerfile            - contains all instructions to assemble an image
│
├── exec_docker.sh        - runs pipeline in docker container
│
├── LICENCE               - contains legal notice
│
├── requirements.txt      - specifies modules required for container
│
├── tmus_session.sh      - start tmux session for seamless
│
└── ...
```

Usage

Note: In the ensuing guide, all specified commands are designed to be run within the Bash command line interface.

Prerequisites / Installation

Check if NVIDIA graphics card is available with:

```
$ ls -la /dev | grep nvidia
```

For running locally

1. Install Python

   ```
   $ sudo apt-get update
   ```

   ```
   $ sudo apt-get install python3.9
   ```

2. Install Pip

   ```
   $ curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
   ```

   ```
   $ python3 get-pip.py
   ```

3. Install further requisites

   ```
   $ pip install -r requirements.txt
   ```

For running in Docker

Note: The NVIDIA Container Toolkits are exclusively available for Linux, rendering both containerization and this guide applicable solely to Linux environments.

1. Install Docker

```
$ curl -fsSL https://get.docker.com -o get-docker.sh

$ sudo sh get-docker.sh
```

2. Install NVIDIA Container Toolkit

```
$ distribution=$(. /etc/os-release;echo $ID$VERSION_ID) \
    && curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo apt-key adc
    && curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-docl

$ sudo apt-get update && sudo apt-get install -y nvidia-docker2

$ sudo systemctl restart docker
```

3. Availability of GPU's in Docker can be testes with the following command (current version-tag here):

```
$ sudo docker run --rm --gpus all nvidia/cuda:11.0-base nvidia-smi
```

Training

Note: In the event of an interruption, training will resume from the latest checkpoint using the previous configuration. Should there be any changes in the configuration, a warning will be displayed prior to the continuation of training. Additionally, the W&B API key is required only if W&B integration is activated. For further assistance, the program manual can be accessed by executing the program with the help flag, for example: `$ bash ./exec_docker.sh --help`

Run locally

```
$ python3 src/main.py -wb <WANDB_API_KEY>
```

Run in Docker

```
$ bash ./exec_docker.sh -wb <WANDB_API_KEY>
```

Testing

Upon completion of training, the application automatically evaluates the performance of both the best-validated model and the most recently trained model through systematic analysis.

Data Sampling

The pipeline is designed to autonomously generate a predefined number of samples in '*.png' format. By including the `--show_samples` flag users can prompt the system to output these samples. These are uniformly distributed across the training dataset and saved in the directory `./data/train_and_test/run_name/samples`. This directory is further organized into two subfolders: `augmented` and `original`, which showcase the effect of preprocessing and data augmentation techniques by brightening the images.

Configuration

Arguments

The configuration of the application can also be changed via command line arguments specified directly, when starting the program, e.g.:

Activate Wandb statistics upload

```
$ bash ./exec_docker.sh -wb <WANDB_API_KEY>
```

Deactivate Training and Testing

```
$ bash ./exec_docker.sh -ntt
```

Activate sample generation

```
$ bash exec_docker.sh -smp
```

Choose Devices for multi GPU training

```
$ bash exec_docker.sh -gpu 1,2
```

Choose configuration file

```
$ bash exec_docker.sh -cfg ./config.json
```

Overwrite configuration file

```
$ bash exec_docker.sh -ycf ./config.json
```

Configuration File

Configurations are copied and stored in the `name/` directory after starting the application for protocolling purposes. Different configurations may be provided via config file in `.json`-format under the path (`any_dir/config.json`) given by argument. When using docker the directory for the configuration file must be `./config.json`. Only configurations that have to be changed need to be specified. The standard configuration looks like this:

At the initiation of the application, configurations are duplicated and logged within the according run directory to facilitate documentation. Users can supply alternate configurations through a .json file, designated by the path argument (any_dir/config.json) as described above. In Docker environments, the configuration file must be located at ./config.json. It is only necessary to specify configurations diverging from defaults. The default configuration is outlined below:

```
{
    "name": "run_0",                        // Identifier for the experiment run
    "group": "group_0",                     // Group name for organizing runs
    "overwrite_run": false,                 // Flag to overwrite existing runs
    "overwrite_configurations": false,      // Flag to overwrite existing configuratior
    "gpu": 0,                                  // GPU index to use

    "dataset_no": 1,                           // Dataset number for selection
    "sample_no": 20,                           // Number of samples to use

    "use_cuda": true,                          // Utilize CUDA for GPU acceleration
    "deterministic_training": true,            // Ensure training determinism
    "deterministic_batching": true,            // Ensure deterministic batching

    "architecture": "resnet50",                // Model architecture, e.g., ResNet50
    "arch_version": 2.0,                       // Version of the architecture
    "pretrained": true,                        // Use a pretrained model
    "num_classes": 2,                          // Number of output classes
    "optimizer": "Adam",                       // Optimization algorithm
    "weight_decay": 0.01,                      // L2 penalty (regularization term)
    "momentum": 0.9,                           // Momentum factor for optimization
    "dropout": 0.3,                            // Dropout rate for regularization

    "set_percentage_cv": 80,                   // Training set size percentage for cross-v
    "set_percentage_val": 20,                  // Validation set size percentage

    "preload": false,                          // Preload dataset into memory
    "batch_size": 8,                           // Number of samples per batch

    "enable_wandb": false,                     // Enable logging to Weights & Biases
    "wb_project": "new_project",               // Weights & Biases project name

    "calc_train_error": true,                  // Calculate and log training error
    "calc_and_peak_test_error": false,         // Calculate and log test error

    "trainandtest": true,                      // Perform both training and testing
    "num_cv": 5,                               // Number of cross-validation folds
    "only_x_cv": 5,                            // Only perform a specific cross-validatior
    "epochs": 80,                              // Maximum number of training epochs
    "learning_rate": 1e-6,                     // Initial learning rate
    "scheduler_step_size": 1,                  // Step size for learning rate scheduler
    "scheduler_gamma": 0.97,                   // Decay rate for learning rate scheduler

    "MCdropout_test": false,                   // Perform MC Dropout test
    "mc_iterations": 100,                      // Number of MC Dropout iterations

    "early_stop_patience": 10,                 // Patience for early stopping
    "early_stop_accuracy": 7,                  // Accuracy for early stopping criterion
    "show_samples": true,                      // Display samples during training/testing
    "binary_class": true,                      // Binary classification task

    "c2_or_c3": "ivoct_both_c2/",              // Data specification (C2 or C3)
    "cart_or_pol": "orig",                     // Use original Cartesian format

    "auto_encoder": false,                     // Utilize an autoencoder
    "autenc_depth": 5,                         // Depth of the autoencoder
    "encoder_group": "group_0",                // Encoder group name
    "encoder_name": "run_0",                   // Encoder run name
    "compare_classifier_predictions": true,    // Compare classifier predictions
    "mirror": false,                           // Apply mirroring augmentation
    "freeze_enc": true,                        // Freeze encoder weights
    "load_enc": true                           // Load pre-trained encoder
}
```

Customization and Modification

Custom CLI options

In scenarios where frequent or rapid modifications to configurations are necessary, leveraging command line options proves advantageous. Registering custom options allows for seamless alterations using CLI flags, enhancing usability and flexibility.

```
args = argparse.ArgumentParser(description='CEAOCTIDNN')
args.add_argument('-cfg', '--config', default=None, type=str, help='config file path
args.add_argument('-gpu', '--gpus', default='0', type=str, help='indices of GPUs to
# Add more custom args here
```

The command-line arguments are utilized to modify configuration settings. For instance, executing the command `bash exec_docker.sh -cfg config.json --bs 256` initiates training according to the parameters specified in `config.json`, with the exception of `batch size`, which is overridden to 256 by the command-line option.

Update Requirements File

In case any changes were made to the code affecting the imports of the application, the requirements file can always be updated (or replaced in case there is one already) by generating a new requirements file with:

```
$ pip3 install pipreqs
```

```
$ pipreqs --savepath ./requirements.txt ./src
```

Note: For the command to function correctly, ensure that all required modules, which are to be imported, are installed to avoid unexpected behavior.

License