# Pipeline: Impact of different Data Augmentation Techniques for Deep Learning with Optical Coherence Tomography (IDDATDLOCT)

"Impact of different Data Augmentation Techniques for Deep Learning with Optical Coherence Tomography"
is the topic which is investigated as well as discussed in the bachelor thesis by Lennard Korte at the Institute of Medical Technology and Intelligent Systems (MTEC) at Hamburg University of Technology (TUHH).
This repository includes the complete academic work and the code basis whose usage is explained below.

**Table of Contents**

## Quickstart Guide for Linux

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

1. Clone the Repository to your home directory

2. Add training data under: `/<home_directory>/IDDATDLOCT/data/`
3. Install Docker and NVIDIA Container Toolkit
4. execute training and testing

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

### Quickstart Guide for MTEC Irmo Server

**Note**: A copy of this repository including training data
is stored on the irmo server.

1. Connect via VPN: (Website Rechenzentrum TUHH)
2. Run Training and Testing on Irmo Server
   1. Login to irmo server via remote SSH repository Access:

      ```
      $ ssh <username>@irmo.et8.tu-harburg.de
      ```

   2. Mount Pallando Volume:

      ```
      $ gio mount smb://pallando.local/share/
      ```

   3. Access repository:

      ```
      $ cd /run/user/1000230/gvfs/smb-
      share:server=pallando.local,share=share/students/Korte/IDDATDLOCT
      ```

   4. Copy the training data (or alternatively the entire repository) to your home directory on the irmo server:

      ```
      $ cp -r ./data/ /$HOME/IDDATDLOCT/data/
      ```

   5. Run training and testing script:

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

## System Requirements

- either or:
  - Windows 10 build is 17063 or later
  - MacOS Mojave or later

    - Ubuntu 20.04 or later
- required:
    - NVIDIA Graphics Card
    - Minimum 16GB RAM
- recommended:
    - 250 GB SSD Storage

## Academic work

See Academic work

## Features

This pipeline provides a simple way of training and testing deep learning models. It offers a number of options to customize the training processes. There are checkpoints and logs to observe and monitor more of its details. A number of parameters may be adapted in a specified configuration file or in the code itself to customize the pipeline in almost any way, i.e. add methods for data augmentation or imege preprocessing techniques. The pipeline can additionally generate examples of the processed images to visualize modifications on images by the human eye. More detailed information can be found in the academic work linked above or in the usage description below.

### Logging

For every Cross-Validation Iteration there is an new folder under the given name. The training output and progress is protocolled in the specified directory under `training.log`. Tests results will be stored in the specified working directory under `test_results.log`. Additional Logging and visualization can be enabled by providing a W&B API key as an argument and activating W&B in the configs.

### Checkpoints

The current model is saved in `latest_checkpoint.pt` after each epoch, while finished training progress is indicated by a renaming to `last_chackpoint.pt`. Checkpoints with the best validation accuracy are preserved in `best_checkpoint.pt`. The checkpoints will be saved under `./data/checkpoints/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

```
IDDATDLOCT/
│
├── academic work/        - Bachelor Thesis and defense
│
├── data/                 - holds large data files like training data and
DA Examples
│
├── exe/                  - holds executable bash and batch files
│
├── src/                  - holds all python source code and standard
configuration
│   ├── checkpoint.py             - defines training checkpoints with
models, optimizer, etc.
│   ├── config_standard.json      - specifies the standard configuration
of application
│   ├── config.py                 - evaluates configurations, arguments
and provides Config object
│   ├── create_samples.py         - enables creation of samples of images
│   ├── da_techniques.py          - implements data augmentation
techniques
│   ├── data_loaders.py           - defines dataloaders for training,
validation and testing
│   ├── dataset_preparation.py    - prepares the dataset for the
dataloaders
│   ├── dataset.py                - represents the dataset object to
dataloader
│   ├── eval.py                   - calculates metrics to assess
performance
│   ├── logger.py                 - for logging result files and printing
tasks
│   ├── main.py                   - entrance point for application
│   ├── models.py                 - defines all models to train on
│   ├── train_and_test.py         - Main training loop file
│   ├── utils_wandb.py            - Wandb logging class
│   └── utils.py                  - Helper functions and utilities
│
├── tests/                - holds all tests and corresponding
configurations for academic work
│
├── .dockerignore         - manages build files for docker
│
├── .gitignore            - specifies intentionally untracked files Git
should ignore
│
├── colab_setup.ipynb     - Notebook file for Colab setup
│
├── config.json           - specifies all non standard configurations
│
├── Dockerfile            - contains all the commands to assemble an image
│
├── initial inspiration   - code this project got inspired by
│
├── LICENCE               - contains legal notice
│
```

```
├── requirements.txt      - contains all modules required for applications
execution
│
└── ...
```

## Usage

**Note:** All commands in the following are intended to be executed in the bash command line.

### Prerequisites / Installation

Check if NVIDIA graphics card is available with:

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

**For running locally**

1. Install Python
   - Windows: Install Python via Microsoft Store.
   - Ubuntu:

     ```
     $ sudo apt-get update

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

   - MacOS:

     ```
     $ /bin/bash -c "$(curl -fsSL
     https://raw.githubusercontent.com/Homebrew/install/HEAD/install.s
     h)"

     $ brew install python3
     ```

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:** NVIDIA Container Toolkits are Linux only drivers. Therefore containerization as well as this guide is Linux only.

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 add - \
    && curl -s -L https://nvidia.github.io/nvidia-
docker/$distribution/nvidia-docker.list | sudo tee
/etc/apt/sources.list.d/nvidia-docker.list

$ 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:

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

**For running in Colab**

To run the application in colab, setup the mounted machine with the provided commands in the `colab_setup.ipynb`-file. Git access rights to this repository are required to follow these steps. Alternatively the repository can be copied to the mounted Google-Drive folder manually. The application is eventually ready to run the same way as a local run.

## Training

**Note:** When training has been interrupted, it will be continued from the last checkpoint with the old configuration. A warning will be shown before continuing training in case the configuration changed. The W&B API key as an argument is only required, when W&B is enabled.Furthermore you can view the program manual by calling the program with the help flag, e.g.: `bash exe/run_train_and_eval_docker.sh --help`

**Run locally**

- Windows:

  ```
  $ exe/run_train_and_eval_local.bat -w <WANDB_API_KEY>
  ```

- Mac / Ubuntu

  ```
  $ bash exe/run_train_and_eval_local.sh -wb <WANDB_API_KEY>
  ```

#### Run in Docker

(Linux only)

```
$ bash exe/run_train_and_eval_docker.sh -wb <WANDB_API_KEY>
```

### Testing

The application tests the trained model automatically after it has been training with the best validated model and the last trained model.

### Data Sampling

The Pipeline can automatically generate a small number of samples that are output in '.*h5'- and '*.jpg'-format. For this the flag `--show_samples` has to be added as described above. These samples are equally distributed over the training data set and stored in the data directory under `./data/h5s/data_augmentation/` and `./data/jpgs/data_augmentation/`. They are brightened and so may help to illustrate the preprocessing and data augmentation techniques.

### 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 exe/run_train_and_eval_docker.sh -wb <WANDB_API_KEY>
```

##### Deactivate Training and Testing

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

**Activate sample data augmentation**

```
$ bash exe/run_train_and_eval_docker.sh -smp 2,3
```

**Choose Devices for multi GPU training**

```
$ bash exe/run_train_and_eval_docker.sh -gpu 1,2
```

**Choose configuration file**

```
$ bash exe/run_train_and_eval_docker.sh -cfg ./config.json
```

**Choose indices of image transformations to be used in the specified order**

```
$ bash exe/run_train_and_eval_docker.sh -da [0,2]
```

**Choose configuration file**

```
$ bash exe/run_train_and_eval_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:

```
{
    // Parameters for testseries and research
    "name": "run0",
    "group": "group0",

    // Hardware
    "use_cuda": false,
    "deterministic_training": true,
    "deterministic_batching": true,     // Only effective if deterministic
training is used
```

```
    // Model
    "model_type": "ResNet18",
    "pretrained": true,
    "num_out": 2,
    "loss_function": "cross_entropy",
    "optimizer": "Adam",

    // Data subdivision for Cross-Validation
    "set_percentage_cv": 80,            // Size of training set in
percentage
    "set_percentage_val": 20,           // Size of validation set when
num_cv = 1

    // Dataloader
    "preload": false,
    "batch_size": 128,

    // Wandb
    "enable_wandb": false,
    "send_final_results": true,
    "b_logging_active": false,

    // Logging
    "calc_train_error": true,           // (If calculated also used in
early stopping)
    "peak_train_error": false,
    "calc_and_peak_test_error": false,

    // Training
    "num_cv": 7,                        // If 1: val set size is
set_percentage_val, if >1: (training set size)/num_cv
    "epochs": 50,                       // Maximum number of epochs
    "learning_rate": 3e-6,
    "early_stop_patience": 15,          // Number of epochs to stop after
when ES condition true, 0 if no ES
    "early_stop_accuracy": 7,           // Digits accuracy is rounded to
    "transformations_chosen": "",       // List of Indices of applied
transformations

    // Dataset specifics
    "define_sets_manually": true,
    "c2_or_c3": "ivoct_both_c2/",       // Specify on which data to train
on
    "cart_or_pol": "orig"               // "orig" chooses cartesian format
as source. "pol" uses polar representation source
}
```

## Customization and Modification

### Custom CLI options

If some configurations need to be changhed often or quickly, then it is usefull to have command line options. By registering custom options as follows you can change some of them using CLI flags.

```
CustomArgs = collections.namedtuple('CustomArgs', 'flags type target')
options = [
    CustomArgs(['--lr', '--learning_rate'], type=float, target=
('learning_rate')),
    CustomArgs(['--bs', '--batch_size'], type=int, target=('batch_size'))
    # Add more custom args here
]
```

`target` argument is sequence of keys, which are used to modify that option in the configuration. In this example, `target`
for the learning rate option is `('learning_rate')` because `config['learning_rate']` points to the learning rate. `bash exe/run_train_and_eval_docker.sh -c config.json --bs 256` runs training with options given in `config.json` except for the `batch size` which is increased to 256 by command line options.

### 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:** All modules have to be installed for this command to work propperly which are supposed to be imported (unexpected behaviour).

## License

This project is licensed under the MIT License. See License for more details

## Acknowledgements

This project is inspired by the code base provided by Robin Mieling from the MTEC institute. Almost all code has been reqritten, but many general concepts remain the same.