SML Project 2

Start Date: May 5
Due Date: June 6

Stochastics and Machine Learning 2025

1 Introduction

The tranquil campus of ETH Zurich has been hit by a storm, causing a sudden disappearance of cherished ETH- and Zurich-themed merchandise. These items have seemingly got scattered around various locations at ETH. However, there are some customers who are waiting for their merchandise to be delivered. Among the lost items, there are some coffee mugs with the ETH logo (Fig. 1a), some t-shirts with the ETH logo (Fig. 1b), and some other coffee mugs with the Zurich cantonal flag (Fig. 1c). The most urgent task is the delivery of the **ETH mugs** - and your task is to find and identify the ETH mugs in images taken at ETH.

(a) ETH Mug (b) ETH T-shirt (c) Zurich Mug

Figure 1: Collection of ETH and Zurich-themed merchandise

You can help us by training an image segmentation model that aims to identify the ETH mugs in a given picture. More specifically, the task here is binary image segmentation. Given an RGB input image (Fig. 2a), the output should be a binary mask of the same dimensions (Fig. 2b), in which each image pixel that belongs to an ETH mug is assigned the value 1 in the segmentation mask, and the rest of the pixels are assigned the value 0.

You are asked to propose an approach for this problem, and compare it with at least one other approach (that can be e.g., the u-net you will see in the lecture). Also, you have to evaluate at least two different configurations (E.g., depth or width of a network you are using). Describe your findings in a 2-page report following the structure of the template.

For the performance evaluation, we will use Kaggle. To join the competition, you must use an email address from the following domains: **ethz.ch**, **mavt.ethz.ch**, or **student.ethz.ch**.

2 Data

2.1 Data Sets

In the **train_data** folder, there are images (in the sub-folder **rgb**) and their corresponding ground truth (GT) segmentation maps (in the sub-folder **masks**). You can use these images

(a) *Input:* RGB image (b) *Output:* Binary segmentation

Figure 2: Task: Given an RGB image as input (left), we want to identify the pixels that correspond to ETH mugs, and return a binary segmentation map (right) as output.

while developing and training your model. The test images can be found in the **test_data** folder. Note that in this folder, only the images are provided (in the **rgb subfolder**). You need to use your trained model to predict the masks for these images.

2.2 Encodings

Each RGB image I has the resolution of 378×252, and can be loaded as an array of dimension $378 \times 252 \times 3$. Each GT segmentation mask M_{gt} is an array of dimension 378×252, and the value of each element in the array is in $\{0, 1\}$. The values are assigned in the following way:

$$\mathrm{M}_{\mathrm{gt}}[i, j] = \begin{cases} 0; & \text{if pixel } I[i, j] \text{ does not belong to an ETH mug} \\ 1; & \text{if pixel } I[i, j] \text{ belongs to an ETH mug} \end{cases} \tag{1}$$

3 Implementation

For the implementation of the model we will use the PyTorch deep learning framework in Python. The PyTorch library comes with simplified methods and classes for setting up a dataloader, setting up the model and training the model. You will not need to implement any of the lower level algorithms such as backpropagation. You can also use architectural building blocks such as convolutional layers and fully connected layers directly from PyTorch. Similar to the project 1, you will simply need to use the existing functions and models in a correct way, and adapt them to your data. This could mean modifying existing architectures, adding data augmentation steps, and tuning the parameters of your training pipeline.

3.1 Project Folder

The project folder we have provided you with comes with a number of files and can be downloaded from Kaggle. Here is a short description for each:

`README.md` A file containing further instructions on technical details for setup and executing your code.

`eth_mugs_dataset.py` In this file, you will need to define your own DataSet class which will load your data. You can also define any preprocessing in this class. We recommend you follow the given structure in this template. Check out the following link for more information: here.

`train.py` In this file, you will need to define your own neural network and design a training loop which makes this network learn how to produce the segmentation masks. This loop saves the parameters of the neural network after each epoch. This is called **checkpointing**. Please see the `README.md` for more information on how to run this file.

`utils.py` This file contains some helper functions.

`sample_submission.csv` This is an example of submission. Your submission to Kaggle should follow this format.

`dataset` This folder contains the training and test data introduced before.

`eth_mugs_dataset_simplistic`, `unet_simplistic` **and** `train_unet_simplistic` contains a very simple implementation based on the unet architecture you have seen in the lecture. It is intended for you as example of the entire process, however the model has a very bad performance.

`Instructions_GoogleColab.ipynb` A file describing the specifics of running the project on Google colab.

3.2 Restrictions

You are free to use any network architecture — be it examples and ideas from the lecture, the exercise groups, or anything you find on the internet. However, **you are not allowed to use any pre-trained model. Only solutions from models that you have trained yourself using the provided datasets will be considered valid.**

Your report has to be 2 pages (plus an additional page for reference) according to the following document template: `https://www.overleaf.com/read/bnjhwknzggdj#ff64b7`. You might want to use `https://www.overleaf.com/` to edit your LaTeXdocument, or install a distribution locally. We recommend `https://tobi.oetiker.ch/lshort/lshort.pdf` as general LaTeXreference.

3.3 Compute Infrastructure

You are free to run the project on any infrastructure. Note that JupyterHub will most likely not have enough resources, or be very slow to train your model (however details depend on the model you design).

We have set up an environment on the GPU cluster run by the IT support group of the computer science department. Alternatively, you might also use an online service such as Google colab (offers powerful GPUs for training and inference, but sometimes supply is limited). Furthermore, depending on your what device you have, running the project locally might also be a good option (especially if your computer has a GPU).

3.3.1 GPU Cluster by D-INFK

You can find the most relevant information and instructions here: `https://www.isg.inf.ethz.ch/Main/HelpClusterComputingStudentCluster`.

If you use the cluster, the training and test data is available (read-only) in the directory `sml_data/datasets` - so if you only need read-access, you don't need to copy this data anywhere. Also, you find the example templates as well as a (very simple) implementation using the simplistic unet as discussed in class. Feel free to modify whatever you think is reasonable on these files; they are just intended as an example for how to run the code on the cluster. Note that you will not be able to save anything in this folder.

Using Jupyter Notebook

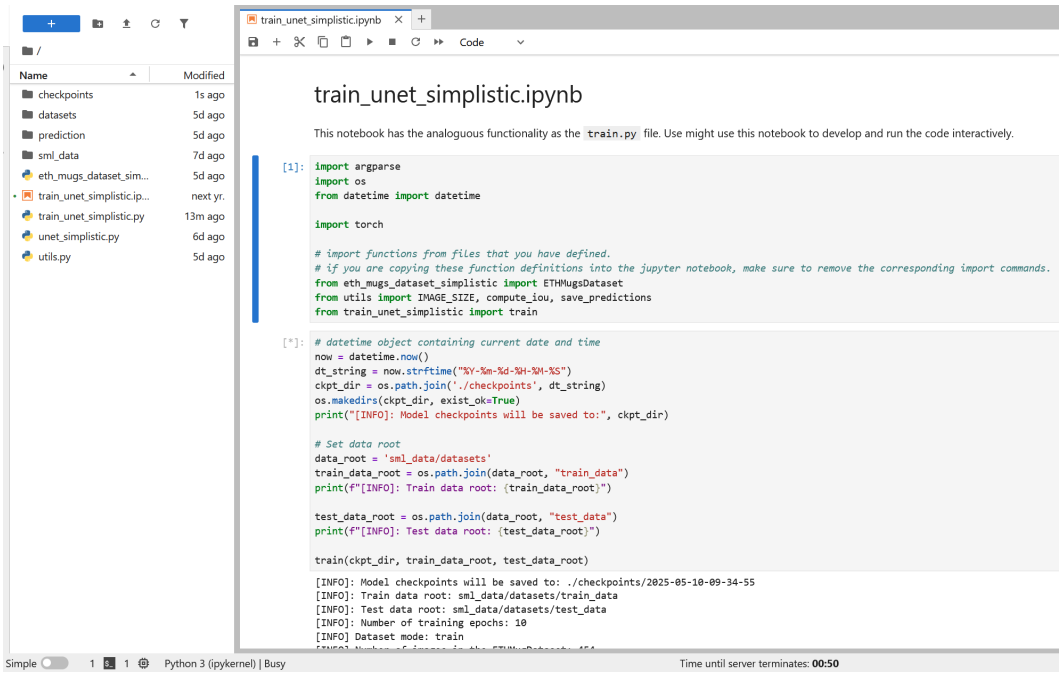

Figure 3: View when you login to `https://student-jupyter.inf.ethz.ch/` and running the provided sample training in the file `train_unet_simplistic.ipynb`.

Go to `https://student-jupyter.inf.ethz.ch/`, login with your ETH credentials, and choose 'sml' as both course and environment. You will get to your home directory (see Figure [**?**]); on the left side you will see a folder 'sml_data' (the other folders are not yet there when you login the first time). You can click on this folder, open the files you would like to see, and then copy them into your home directory ("File" → "Save Notebook as").

To run the provided simplistic unet (training on the training data, and testing on the test data), copy the files `eth_mugs_dataset_simplistic.py`, `train_unet_simplistic.ipynb`, `train_unet_simplistic.py`, `unet_simplistic.py` and `utils.py` to your home directory. Then, open `train_unet_simplistic.ipynb` to run the provided sample implementation using the simplistic unet architecture. You can now modify the files see the effect.

Note that the jupyter server will restart after 60 minutes. You see the count-down in the bottom middle of your browser window. Save the notebooks before it reaches 0, so you can restart the server and continue your work.

For longer computations (namely, the training of a model you have defined), we recommend you submit your job to the scheduling system.

Console Access and Batch Submission System

To login to a login node for the cluster, open a console window and type in

```
ssh <eth-user-name>@student-cluster.inf.ethz.ch
```

You will get to your home directory. If you want to use the prepared environment, you have to first activate it using the following command (note the dot):

```
. /cluster/courses/sml/jupyter/bin/activate
```

Again, you see (read-only mode) the example files by typing `ls sml_data` and pressing enter. Use the commands `cp sml_data/*.py .` and |cp sml_data/my_job.sh .—

Using `sbatch my_job.sh` will submit a job into the queuing system. Depending on the workload of the cluster, you might have to wait for your job to be processed, but usually jobs get started quickly.

The job runs the file `train_unet_simplistic.py` with argument `-d sml_data/datasets/` (the directory where the data sets are stored) for 1h30min (`01:30:00` - it will actually need less than 10min). When you submit a job, you should get an output similar to this one:

```
astreich@student-cluster:~$ sbatch my_job.sh
sbatch: GPU count: 1
sbatch: CPU count: 2
sbatch: Memory: 24576MB
Submitted batch job 22040
```

After completing the run, the file `vi slurm-<jobID>.out` (i.e., `vi slurm-22040.out` in the above example) will contain all the console output your job created.

You should only change the other lines if you know what you are doing. Every student that was enrolled in the SML course in mid-April should have access to the GPU cluster, and gets a budget of 50 GPU hours for batch jobs and 50 GPU hours to run Jupyter notebooks. If you do not have access to the cluster, please reach out to Andreas Streich.

4 Tips for solving this project

4.1 Creating Splits

When training models and evaluating different hyperparameters and model architectures, be careful not to do overfitting. It is common practice in machine learning to divide the data in train, validation, and test splits. The train split is used to learn the parameters of the model while the validation split is used to decide on the hyper-parameters of the model. Hyper-parameters are parameters that are not learned from the data during the training process but are set prior to training. They control various aspects of the learning algorithm like the learning rate and the image resizing factor. The test split is the data you use to evaluate your final model.

For this project, you will get a set of labeled data `train_data`, and the a second set of data called `test_data` without labels. If you need further splits of the data, please create them yourself, and document the splitting in the report.

4.2 Preprocessing data

Once you have your data divided in train and validation splits, the next step is to preprocess the data you load in the dataloader by applying different types of transformations to them.

1. You might need to consider whether or not you should normalize the color values of your image before passing it as an input to your model. In order for a machine learning model to perform well, it is important that the features are scaled in a common range. Therefore, you might need to use a scaling method.

2. You can consider applying data augmentation functions (you can take a look at this link) to your images. Don't forget that some augmentation functions will need you to apply the same transform to the GT mask as well. For instance, if you rotate or flip the RGB image, the positions of the ETH mugs in the new image will be different than their positions in the original segmentation mask. But slightly changing the color values (e.g. jittering) of the RGB image should not have any affect on the segmentation mask.

5 Evaluation Criteria

Students have to develop, implement and evaluate at least one own approach to solve the image segmentation task, and compare it to at least one other approach (can be e.g., a simple unet as discussed in class). For their own approach, they have to investigate the impact of at least one variation / hyperparameter (e.g., number of layers, width of layers, regularization hyperparameter, etc.) using at least 3 different values of that variation / hyperparameter.

Reviewers will make a reasonable effort to read and understand the report. **However, they can only judge and grade what is in the report. Therefore, make sure to clearly describe all relevant steps in your thoughts and experiments.** A poor report might thus not only affect the criterion "Quality of report", but also the criteria of creativity and evaluation.

5.1 Creativity: 20 points

The creativity of the best approach will be granted a maximum of 20 points according to the following criteria:

- ○ A slight variation of an approach discussed in class (e.g., one more layer, variation of filter size/layer width): 5 points
- ○ An approach found in literature as a result of independent literature search: 10 points
- ○ A variation of a literature approach: 15 points
- ○ A non-standard combination of at least 2 approaches, and/or significant variation of a standard approach: 20 points

5.2 Experiments: 30 points

The evaluation of the proposed approach will be granted a maximum of 30 points. The following criteria will be checked, and 5 points will be granted for every fulfilled criterion.

- ☐ At least 2 different approaches are described (one can be a standard approach as discussed in class, e.g. the unet architecture).
- ☐ Relevant metrics are used for the comparison.
- ☐ At least 3 relevant variations / hyperparameters are evaluated.
- ☐ The model comparison / hyperparameter selection is done on validation set
- ☐ The final evaluation is done on separate test set (public leaderboard data)
- ☐ Cases of poor / good model performance (i.e., when does the model work well, or not) are described.

5.3 Quality of the Report: 20 points

The quality of the report is evaluated according to the following criteria (5 points each):

- ☐ The proposed approach is motivated and clearly described.
- ☐ All relevant aspects of the experiments are clearly described.
- ☐ The results are interpreted, and the findings are consistent
- ☐ The text and graphics are of good quality.

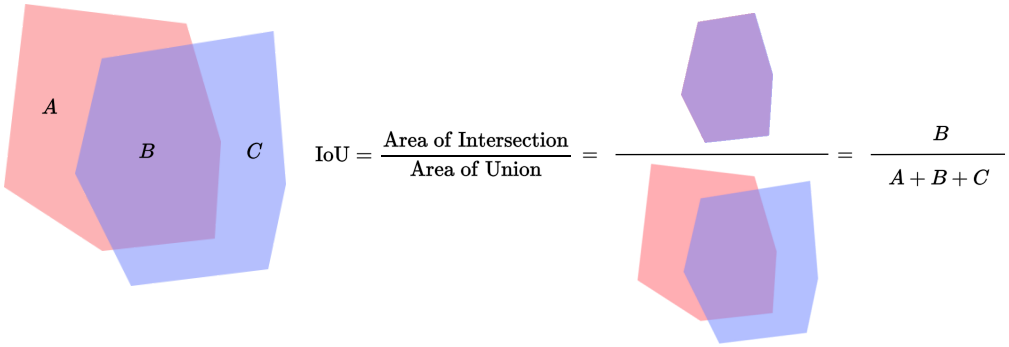

Figure 4: Illustration of the Intersection-over-Union (IoU) metric computation. Red area $(A + B)$ refers to the ground truth mask, and the blue area $(B + C)$ refers to the predicted mask. The area of intersection is the overlapping area, i.e. B.

5.4 Model Performance: 30 points

The evaluation metric used for the model performance is the **Intersection-over-Union (IoU)**, which is defined as the following:

$$\text{IoU} = \frac{\text{Area of Intersection}}{\text{Area of Union}} \tag{1}$$

In Fig. 4, we illustrate the computation of the IoU metric. For our segmentation task, the IoU will be computed between the ground truth segmentation mask and your predicted segmentation task. In that case, pixels that are assigned the value 1 in your predicted mask will be one set, and the pixels that are assigned the value 1 in the ground truth mask will be another set. The **area of intersection** of these two sets is defined as the *number of pixels* (i, j) such that $\text{M}_{\text{gt}}[i, j] = 1$ *and* $\text{M}_{\text{pred}}[i, j] = 1$. The **area of union** of these two sets is defined as the *number of pixels* (i, j) such that $\text{M}_{\text{gt}}[i, j] = 1$ *or* $\text{M}_{\text{pred}}[i, j] = 1$.

For the overall grading, the IoU will be converted to points as follows:

$$p = \min\left\{30, \frac{100}{3} \cdot \text{IoU}\right\}$$

6 Submitting your predictions

6.1 Submission to Kaggle

To evaluate your model, you must submit a CSV file to Kaggle, which includes the following two columns:

- **ImageId (str)**: a four-digit image ID which is the prefix of the filename. For example, the ImageId for the image file named "0001_rgb.jpg" is "0001".
- **EncodedPixels (str)**: Run-length encoding of the predicted binary mask.

You can use the `save_predictions()` function provided in `utils.py` to generate your submission file in this format. After submitting to Kaggle, you will receive an IoU score on Kaggle's public leaderboard. However, note that this score is calculated using only a subset of the test images (i.e., the public test set). The private leaderboard, which evaluates your predictions on the remaining test images (i.e., the private test set), is not visible until the project ends. Your final grade will be based on your score from the private leaderboard.

6.2 Submission to Moodle

Please submit to Moodle a `.zip` file containing:

- **Source code**, containing all program code (not including libraries) needed to run your model.
- **Model checkpoint**, containing all parameters of your model.

6.3 Submission to Openreview

Describe your ideas and the experiments / results on a short report. Please follow the structure given in the template `https://www.overleaf.com/read/bnjhwknzggdj#ff64b7`, and stick to the limit of 2 pages of text (you can have one more page for the references if needed). You can copy the template and work in overleaf or any other tool of your choice. Write the report such that a fellow student attending the course "Stochastics and Machine Learning" is able to understand it.

For the reviewing of the report, we will use the platform openreview. Submit your report here: `https://openreview.net/group?id=ETHz.ch/2025/Course/SML` If you do not have an account yet, please create one. Make sure you add all the members of your team as author, and upload your report to openreview.