LELEC2885 - Image Processing
Project Guidelines

Academic year 2024-25

Anne-Sophie Collin (anne-sophie.collin@uclouvain.be)
Edouard Couplet (edouard.couplet@uclouvain.be)
Baptiste Standaert (baptiste.standaert@uclouvain.be)

Do not be afraid by the length of this document. The aim is twofold: to explain what is expected in the project and to provide you explanations on more specific points that you may not have seen during lectures. Do not read everything in detail in one go. Instead, consult this document for help when you need it.

Contents

1 Context of the project and general guidelines

The project is divided into two parts:

PART A. A classic segmentation task. For this first part, you will be asked to implement a classic deep learning method for generating segmentation masks out of cats and dogs images. For this purpose, a workflow based on the Pytorch library is provided. To achieve this task successfully, you will need at least to implement your own network architecture (`basicNetwork.py`), implement the training loop (`model.py`) and enhance the evaluation procedure (`model.py`).
This first part of the project should be completed by week 11 and be presented to the teaching assistants during an intermediate oral presentation of 10 minutes.

PART B. An open research question. For this second part, you will be asked to explore a research question based on the work done in part A. This second part is due by week 13.

1.1 The Oxford-IIIT Pet Dataset

In this project, we will be working with the Oxford-IIIT Pet Dataset [1] containing 7390 images of cats and dogs. For each image, we are interested in the task of predicting the binary segmentation mask of the cat or dog as represented in Figure 1.

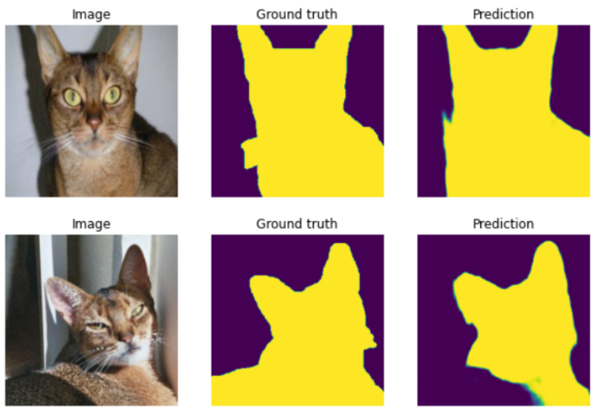


Figure 1: Binary segmentation of cats vs. background.

To carry out this task, you will implement a deep learning-based approach. Your network will be trained to output a binary mask given a cat or dog image as input.

Corrupted Images
Some images in this dataset do not have a proper format. Do not try to handle this issue! Instead, discard the following images (and masks) from the dataset:

- Egyptian_Mau_14, 139, 145, 156, 167, 177, 186, 191;
- Abyssinian_5, 34;
- chihuahua_121;
- beagle_116.

1.2 Schedule and Deliverable

The deadline for submitting the project is Wednesday 11 December at 10 p.m. Even if this deadline seems a long way off, do not wait until the last minute to get started. Training a model can take several hours, depending on your computer.

Multiple Q&A sessions will be organised with teaching assistants. Also, a short (mandatory!) intermediate oral presentation of 5 minutes followed by a discussion of 5 minutes will be organised during week 11. Details will follow on Moodle in due time.

This project can be carried out by group of two or three students. Each group is asked to submit on Moodle:

- A zip file with your code (don't re-upload the dataset!);
- A written report. Further instructions are given in section 1.3.

1.3 Content of the Report

Consider that the person who is going to read your report already has a technical background in deep learning. There is no need to explain what is a network, a convolution or a train/validation/test set. The report is there to justify the choices you have made and present your results.

In a pdf report of max. 3 pages (excluding images and references), comment on the following aspects of your project:

PART A. A classic segmentation task.

1. **Network Architecture.**
   Which network architecture did you use? Describe it briefly.
2. **Network Training.**
   Which loss function did you use? What are your main training parameters (learning rate, number of epoch, batch size)?
   Also show your learning curves in a graph. Did you observed overfitting? If so, how did you mitigate it?
3. **Qualitative Evaluation.**
   Provided some visual examples of the results you obtained. Show both success and failure cases and comment them.
4. **Quantitative Evaluation.**
   Quantify the final results of your method with a suited metric of your choice. Comment them.

PART B. An open research question.
No specific guidelines are provided for this part of the written report. See details in Section 3.

2 PART A. A classic segmentation task

2.1 Towards a first run of the code

The project has been written in order to be directly runnable on your computer. We trained a dumb network for you (defined in the `basicNetwork.py` file). After a successful run, you should be able to see the results it produces in the `Results > DefaultExp > test` folder.
Your job will be to improve this fool deep learning-based method. However, running this first basic experiment will allow you to determine whether or not the installation of the needed libraries was successful. In Section 2.1.1, we provide information about the Pytorch library and, in Section 2.1.2, explanations are given for the Albumentations library. You can simply run the provided code and install missing libraries as stated in error messages.

How to launch an experiment?
In the terminal, run the following command[a]:

```
$ python main.py -exp DefaultExp
```

The exp argument defines the the name of the yaml file (in the `Todo_list` folder) with the wanted configuration (here `DefaultExp.yaml`)

[a]If you receive a lot of warnings, use this command:

```
$ python -W ignore main.py -exp DefaultExp
```

2.1.1 Pytorch library

In this project, you will rely on the Pytorch library to write your deep learning framework. This library is well documented and tutorials are available to learn how to use it. You can also find installation instructions here.
First of all, you should know that Pytorch can run on CPU or GPU. The procedure for installing Pytorch on a GPU can be long and (sometimes) complex. However, it will allow your code to run up to 20 times faster than on a CPU...

Installing Pytorch on CPU is straightforward. On this website, choose your OS (Linux, Mac or Windows) and select "CPU" as Compute Platform. The proper command line will be written below. To run pytorch on GPU, you will first need to install dedicated GPU drivers. CUDA is the name of the drivers that will enable the parallelization of your code. The installation can be complex.

Can your computer support Pytorch on GPU?
Not all GPU are compatible with CUDA. To check whether you computer has a CUDA compatible GPU, run this command in your Python environment:

```
$ import torch
$ print(torch.cuda.is_available())
```

If the output is True, then your system has a compatible GPU and it is possible to run Pytorch on GPU. Otherwise, only your CPU can be used to run the code.

In our project, you can choose between of the use of the CPU or the GPU by changing the `DEVICE` parameter in the `Todo_List > DefaultExp.yaml` file. If `DEVICE` is set to "cpu" then the code will run on CPU. If `DEVICE` is set to "cuda" then the code will run on GPU.

2.1.2 Albumentations library

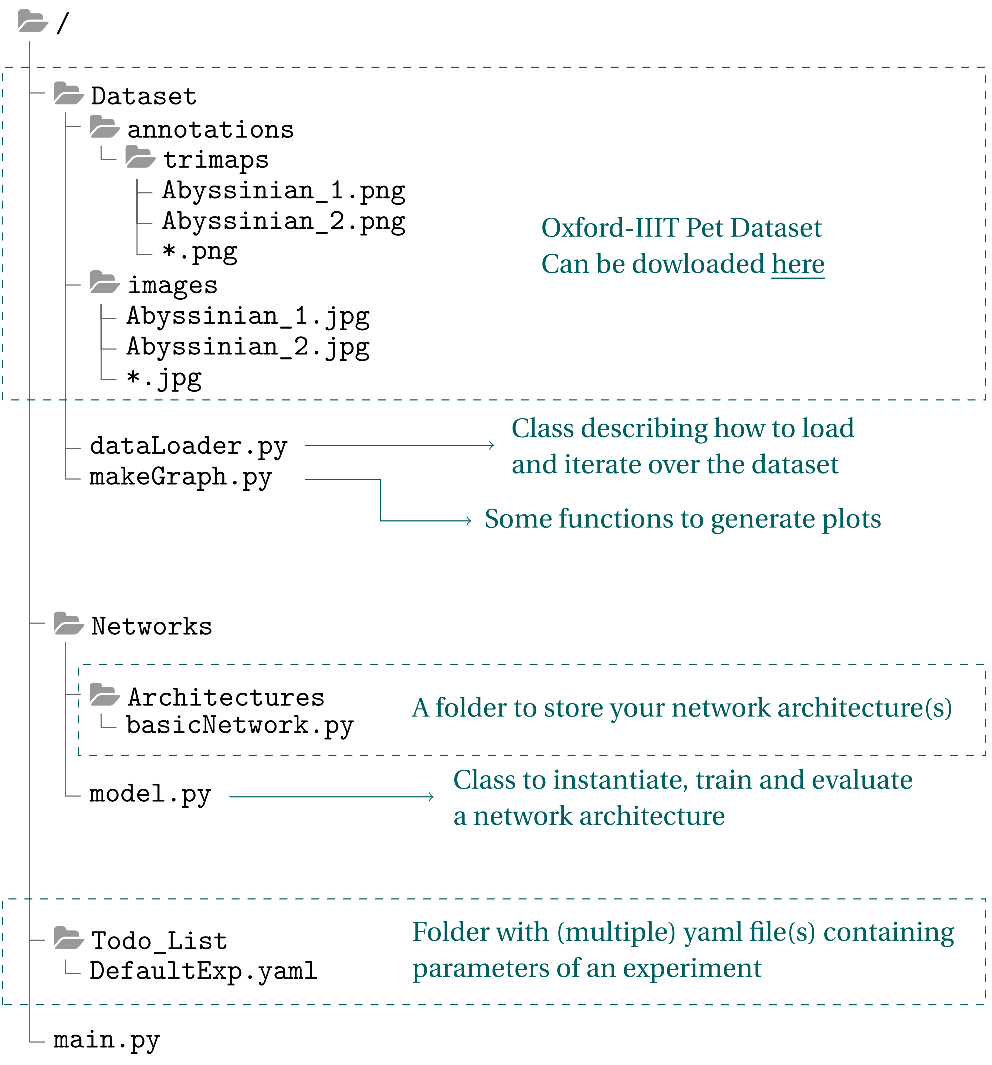
The second deep learning-dedicated library to install is Albumentations. This library will be used to perform the data augmentation process. Do not worry, the installation is straightforward and is not specific to the use of the CPU or the GPU.

2.1.3 Other libraries

You will need to install termcolor. We also recommend you to use the Lovely tensors library to print information on your pyTorch tensors.

2.1.4 Code structure

It is common for a deep learning framework to require a larger amount of code than a "classical" computer vision project. If no particular attention is paid to it, the code can look messy and may not be practical to use. To help you, we already provide you with a project structure that improves the code modularity. The objective is to run multiple experiments by launching one unique main function with different configuration files (yaml files in the Todo_List folder).



/
Dataset
annotations
trimaps
Abyssinian_1.png
Abyssinian_2.png
*.png
images
Abyssinian_1.jpg
Abyssinian_2.jpg
*.jpg
Oxford-IIIT Pet Dataset
Can be dowloaded here
dataLoader.py
makeGraph.py
Class describing how to load and iterate over the dataset
Some functions to generate plots
Networks
Architectures
basicNetwork.py
A folder to store your network architecture(s)
model.py
Class to instantiate, train and evaluate a network architecture
Todo_List
DefaultExp.yaml
Folder with (multiple) yaml file(s) containing parameters of an experiment
main.py

Figure 2: Structure of the provided code.

The code structure is presented in Figure 2. There are 3 folders in the root directory:

1. **Dataset.** This folder contains functions used to download the dataset, iterate over the train/-validation/test sets and show some images.
2. **Networks.** This folder contains functions used to defined the network architecture + train and evaluate it.
3. **Todo_List.** This folders can contain as many `yaml` files as you want. Each one will correspond to one set of hyperparameter to test.

2.2 Expectations in terms of code

You are authorised to modify any provided files if necessary. However, we expect you to implement at least the missing parts of the code described in the box below.

You are expected to complete the following code sections in the `model.py file`:

1. **`__init__` function**
   You should complete :
   - line 67 to add the loss function
   - line 68 to add the optimizer

   that will be considered for training your network in the segmentation task.
2. **`train` function**
   You must write the loop to train and validate your model for a given number of epoch. At the end of the training, you are asked to print your train and validation loss curves into a graph.
3. **`evaluate` function**
   We have written some code to show you some examples of masks generated by your model. This is a qualitative evaluation.
   We ask you to complete this `evaluate` function with a quantitative evaluation: identify a suitable metric to quantify the performance of your model.

Also, you should improve the basic network architecture that is provided in the `basicNetwork.py` file. To do this, you will need to do some research of your own. Do not forget to mention your sources in the report.

3 PART B. List of open research questions

The first part of the project is marked out of 12/20. In the second part, you can choose an extension track that will enable you to improve this initial grade to reach the total amount of 20/20. Choose only one track from those listed.

Integrate your main conclusion in the written report and create a new project folder to modify your code from part A.

3.1 Track 1. Uncertainty Estimation

A deep learning network trained over our segmentation task will always output a segmentation mask. We can't imagine a situation where the network's prediction would be "I don't know". Estimating the uncertainty of the network's prediction is a particularly interesting research topic.
In this track, we explore the use of a method called "test time data augmentation" (TTA) to estimate the uncertainty of our network over the segmentation task.

Data augmentation is usually a set of techniques for creating new **training** images in addition to those already available. To do that, transformations such as adjusting the brightness of the image, or rotating the image, or shifting the subject in the image horizontally or vertically are considered. An example is shown in Figure 3. When used during training, data augmentation helps to prevent overfiting and deals with the difficulty of collecting huge amount of data.

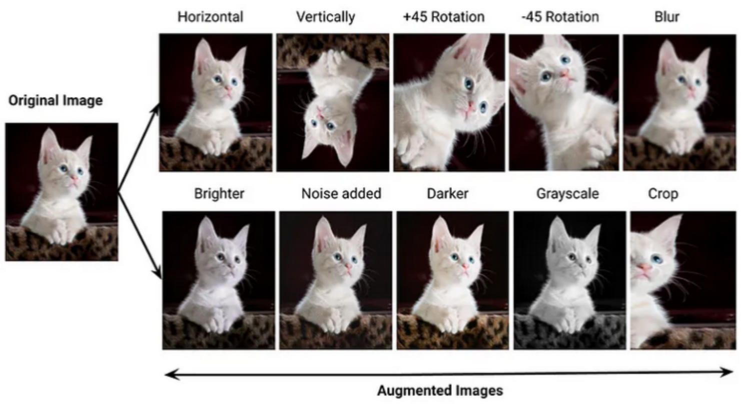


Figure 3: Examples of augmentation transforms that can be considered on images (image source).

In this track, the objective is to use data augmentation techniques during **test** time to estimate the uncertainty of the model about the segmentation mask. This method is described in this article [2]. Main steps are the following:

1. For each test image, generate T augmented versions.
2. Infer each augmented image though the network and get T predicted segmentation masks.
3. Align all prediction masks: if the input image has been rotated 10° to the right, rotate the predicted mask 10° to the left.
   (Think about the transforms that need to be inverted and those that do not).
4. Evaluate uncertainty by computing the entropy of each pixel across the T output aligned masks.

Observe the uncertainty obtained for the images of the test set and comment. How can you interpret your results?

3.2 Track 2. Self-supervised pre-training

In many real-world scenarios, annotating an entire dataset can be time-consuming and resource-intensive. However, the lack of labeled data does not necessarily mean that the data cannot be utilized effectively. Self-supervised learning techniques allow us to exploit the information contained within the data itself, without the need for ground truth labels. By pre-training a model on an unsupervised task, we can capture meaningful features that can then be fine-tuned for specific applications, such as image classification or segmentation.

In this track, your task is to pre-train your segmentation model using a self-supervised learning approach, such as image inpainting, converting grayscale images to color, or denoising. For ease of implementation, you can use the Oxford-IIIT Pet Dataset for the pre-training phase. While other datasets might also be suitable, using a different dataset would require additional work to integrate it into the PyTorch framework.

Here is a description of the process:

1. **Self-Supervised Pre-training.**
   Your first task is to pre-train a CNN on an unsupervised learning task. This task should not rely on the image labels but instead should aim to learn useful representations from the data. Some possible self-supervised tasks include:
   - Denoising an image after corrupting it with gaussian noise;
   - Reconstructing a masked or missing part of an image (image inpainting);
   - Transforming a grayscale image back into color.

   Conduct some research to select and define the self-supervised task you will use. Feel free to consider other pre-training tasks.

2. **Fine-tuning for Image Segmentation.**
   Using the pre-trained model as an initialisation, fine-tune it to perform the segmentation task described in Part A. Compare the performance and training time of the fine-tuned model to that of a model trained from scratch.
   - Does the self-supervised initialization improve the final results?
   - Does it reduce the training time needed to achieve good performance?

References

[1] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, and CV Jawahar. Cats and dogs. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3498–3505. IEEE, 2012.

[2] Guotai Wang, Wenqi Li, Michael Aertsen, Jan Deprest, Sébastien Ourselin, and Tom Vercauteren. Aleatoric uncertainty estimation with test-time augmentation for medical image segmentation with convolutional neural networks. *Neurocomputing*, 338:34–45, 2019.