Univ.-Prof. Dr.-Ing. Matthias Boehm
Technische Universität Berlin
Faculty IV - Electrical Engineering and Computer Science
Berlin Institute for the Foundations of Learning and Data (BIFOLD)
Big Data Engineering (DAMS Lab) Group

1 AMLS SoSe2024: Exercise – Sentinel Building Segmentation

Published: Apr 15, 2024 (last update: Apr 15)
Deadline: Jul 08, 2024; 11.59pm

This exercise is an alternative to the AMLS programming projects and aims to provide practical experience in the exploratory development of machine learning (ML) pipelines. The task is to classify building locations in cities from satellite images. The trained model(s) classify if pixels contain buildings or not. You may choose any programming language(s) and utilize existing open-source ML systems and libraries. The expected result is a zip archive named `AMLS_Exercise_<student_ID>.zip` (replace `<student_ID>` by your student ID) of max 20 MB, containing:

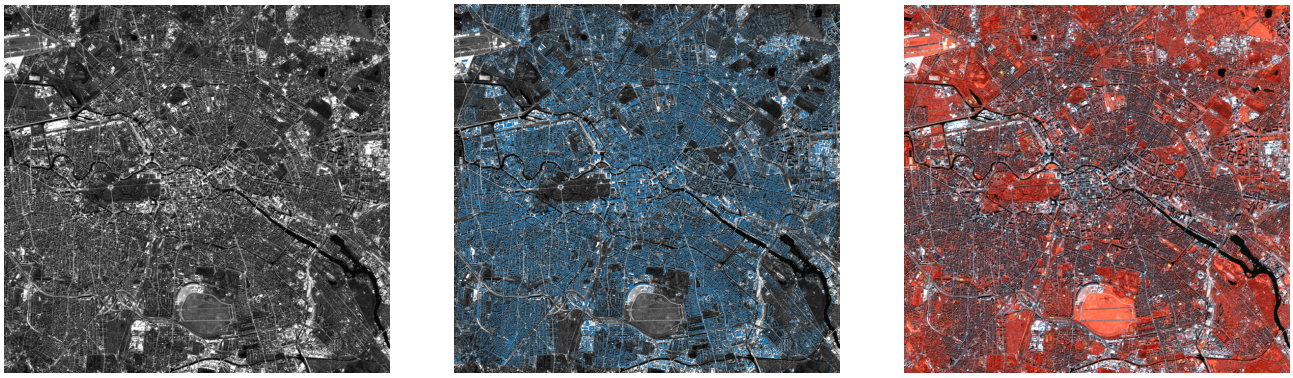
- The source code used to solve the individual sub-tasks
- A PDF report of up to 8 pages (10pt), including the names of all team members, a summary of how to run your code, and a description of the solutions to the individual sub-tasks.

Data: We will be using data from the Sentinel Copernicus program, specifically the Sentinel 2 satellites. Baseline solutions must use the processing level 2a 10-meter spatial resolution bands as inputs. The test data for all accuracy results shown in the report should use the latitude north: 52.574409 south: 52.454927, and longitude west: 13.294333 east: 13.500205.

Grading: This exercise is be pursued in teams of 1 to 3 persons (one submission). The grading is a *pass/fail* for the entire team. Exercises with $\geq 50/100$ points are a pass, and the quality expectations increase with the team size. Exercises with ≥ 90 points receive 5 extra points in the exam.

(a) Buildings from OpenStreetMaps (b) RGB Bands from Sentinel 2

Figure 1: Latitude North: 52.574409 South: 52.454927 and Longitude West: 13.294333, East: 13.500205



(a) Single Band (b) Overlapping Buildings (c) URB

Figure 2: Data Acquisition and Alignment Plotting Tasks

1.1 Data Acquisition and Alignment (15/100 points)

Make a pipeline that constructs training data based on latitude-longitude coordinates. The pipeline has to solve two tasks: Download Open Street Maps files and create map projections of the contained buildings as seen in Figure 1a, and download satellite images from a Sentinel 2 data provider.

We do not encourage you to construct your own Open Street Maps parser and recommend using libraries. You are allowed to use any library of your choosing. One approach to solving the task, using, for instance, pyrosm, is to loop through different cities, download their respective Open Street Map files, extract coordinate bounds for their locations, and then filter everything other than buildings. To get familiar with Sentinel 2, we suggest looking at the interactive copernicus browser. There are multiple sources to download Sentinel 2 patches based on coordinates (e.g., ESA). You can use any provider you want that gives Sentinel 2 images processed to level 2a. Openeo is a library that allows you to specify coordinates, cloud cover percentage, provider URL, and time range to download.

We require you to collect data from at least ten 'big' cities. We encourage, but do not require, experiments with different numbers of cities in the training data.

Expected Results: Code for data acquisition and reprojection into equal earth projections, as well as plotting pipelines for the report of single bands (Figure 2a), buildings (Figure 1a), RGB (Figure 1b), URB (Ultraviolet, Red, and Blue) (Figure 2c), and overlapping buildings (Figure 2b).

1.2 Data Preparation (25/100 points)

Preprocess the input data to prepare the data for model training and testing. Given the pipeline from Task 1.1, prepare the large images for training by slicing out tensors containing smaller training images of equal width and height, for instance, 32, 64, or 128 pixels. Note that patching is only for training because the model should be able to process any input size (segmentation of individual pixels). Furthermore, include in the pipeline a simple cloud-cover classifier and remove all patches containing clouds. The result should be:

- A tensor with the aligned spectral bands. You choose which dimension contains the channels. Dimensions [N, H, W, C] or [N, C, H, W]
  – N: Number of Images, H: Image Height, W: Image Width, and C: # Channels
- A tensor with the target buildings (one class therefore no channels). Dimensions [N, H, W]

Divide the data into train/validation/test sets of similar data distributions (e.g., label distribution and potentially other properties such as visual similarity or data provenance). For the sake of comparison, everyone must use the latitude and longitude ranges specified as test data. Ensure that the test data is not leaking into the train and validation sets.

Expected Results: Code for data preparation of disjoint train, validation, and test data with similar distributions of statistics. The report should justify the chosen selection procedure and how it preserves the similarity of specific properties.

1.3 Modeling and Tuning (35/100 points)

Construct an ML pipeline—using the prepared train and validation sets—for classifying specified coordinates as building or no building. You are required to use a baseline model of four padded CNN layers. The initial three layers should use a width and height of 3 for their kernels and a single stride. Each CNN layer should increase the channels to 32, 64, and 128. The final CNN layer uses a kernel size of 1x1 and produces a single value channel. The single value should indicate if the pixel contains a building. You are not required to use any specific ML system, but an example in Keras is:

```
from tensorflow.keras import layers, models
p='same'
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', padding=p, input_shape=(H, W, C)),
    layers.Conv2D(64, (3, 3), activation='relu', padding=p),
    layers.Conv2D(128, (3, 3), activation='relu', padding=p),
    layers.Conv2D(1, (1, 1), padding=p)])
```

Or pytorch:

```
import torch.nn as nn
model = nn.Sequential(
    nn.Conv2d(C, 32, kernel_size=3, padding=1), nn.ReLU(),
    nn.Conv2d(32, 64, kernel_size=3, padding=1), nn.ReLU(),
    nn.Conv2d(64, 128, kernel_size=3, padding=1), nn.ReLU(),
    nn.Conv2d(128, 1, kernel_size=1, padding=0))
```

Subsequently, choose an appropriate loss function and evaluation metric and evaluate this metric on both the train and validation data. Train the model multiple times and tune the hyper-parameters of your model (e.g., regularization parameters, channels, learning rate, optimizers, early stopping). Some points are given for parallelization of training multiple models with different parameters. Finally, report the evaluation metric on the test data. You need at least to explore one additional segmentation model architecture to get full points. It has to be either U-Net, SegNet, FastFCN, Gated-SCNN.

Expected results: Code for model training and running, as well as descriptions of the used model architecture, ML pipeline, and its evaluation. The report must include statistics or plots of the quality of your models with and without hyper-parameter tuning. For full points, you have to justify and reason why and how the selected models work.

1.4 Data Augmentation (25/100 points)

Furthermore, improve your model quality via data augmentation techniques such as rotations, reflections, modulated noise, zoom, mixup, and shearing/distortions. For this task, add only data preparation steps, whereas the model training and hyper-parameter tuning should remain unchanged. Conduct systematic experiments on the train and validation sets, and properly document the intuition and impact on model quality of the applied techniques. To reduce the runtime of the experiments, you may use proxy models that are not trained to full convergence. Finally, evaluate the quality of your model with and without data augmentation on the test data.

Expected Results: Code for applying data augmentation techniques and a summary of their impact on model quality. Full points are only possible by evaluating the impact of the individual augmentations and compound combinations.