Documentation LMUnet

Landscape and metric generation in R/Rstudio....1
1. Generate landscapes....1
2. Generate Metrics (furrr)....1
3. Generate Metrics (slurm)....1
4. Download large folders from Rstudio server....1
5. Calculate landscapes metrics from predicted landscapes....2
Model training and inference (Python/JupyterLab)....2
Preparation/Data management....2
6. Upload large folders to jupyterlab (hint: don't)....2
7. Stack/concatenate metrics for multi-metric training....3
8. Example_landscape_corine.ipynb....3
9. Speed Test loading times....3
10. Convert landscapes to binary maps....3
Predict landscape metrics....3
11. Train UNet for metrics....3
12. Test trained UNet metrics (Infer, evaluate, visualize)....3
Optimize Landscape....4
13. Optimize one Landscape inversely ("conventionally")....4
14. Optimize one Landscape inversely ("LPbP")....4
15. Plot line charts and barplots....5
Python scripts used in the notebook operations....5
Csv files....6
Data....6
Outputs....7
Instructions for HPC....8

Landscape and metric generation in R/Rstudio

1. Generate landscapes
    a. Go to Rstudio server
    b. Open document generate_landscapes_only
    c. Adjust:
        i. Number_classes: (between 3 and 6)
        ii. Number of landscapes (default currently 10k) (needs to be tested)
        iii. Window_size: (default 5, could be 3)
        iv. Data_directory: rename
    d. Run the whole script
        i. Landscapes will be saved in the corresponding folder, together with a naming list saved under metric_list

2. Generate Metrics (furrr)
    a. Go to Rstudio server
    b. Open document load_landscapes_generate_metrics
    c. Adjust:
        i. Number of metrics
        ii. Type of metrics (“LSM_METRIC”)
        iii. Data_directory
    d. Run the whole script

3. Generate Metrics (slurm)
    a. Go to Rstudio server
    b. Shh into hpc
    c. Nano gen -> tab
    d. Hashtag previous metric, un-hashtag next one
    e. Ctrl+x, y, enter
    f. Open R (R + enter)
    g. source(“genera…..R”)
    h. q()
    i. Cd _rslurm_lsm…etc
    j. Sbatch submit.sh

4. Download large folders from Rstudio server
    a. Go to Rstudio server
    b. Open zip_folders.R

c. Specify name of the zipped file
d. Specify location of the folder to be zipped
e. Run the zip command
f. Afterwards, in the Files pane:
    i. checkmark the zip-file
    ii. Click on the more-cogwheel
    iii. Click export

5. Calculate landscapes metrics from predicted landscapes
    a. Go to Rstudio server
    b. Upload landscape.npy
    c. Open individual_metric_generation.R
    d. Specify location of the landscape to calculate LSMs from
    e. Specify “name” of the landscape to calculate LSMs from
    f. Run the script
        i. Optionally for multiple landscapes at once

Model training and inference (Python/JupyterLab)

The Data-structure should look like this:

```
├── data
│   ├── 100k_5cl_multi_metrics_66 [training, validation, testing metrics]
│   ├── 100k_landscapes_5_class   [training, validation, testing landscapes]
│   ├── 100k_joinent_5_class_metric_list.csv
│   └── multi_metrics_66_csv
├── outputs
│   ├── landscape_testiing [for inverse outputs]
│   └── saved_models [including the pretrained models and their loss rates]
├── metr_norm.csv
├── mean_std.csv
└── nan_metrics.csv
```

Preparation/Data management

6. Upload large folders to jupyterlab (hint: don’t)
    a. Go to jupyterlab
    b. Navigate to upload folder
    c. Press upload button
    d. Select zip file
    e. Open prepare_metrics.ipynb notebook

f. Adjust directories
g. Execute cell by cell to unzip, move and change to float 32

7. Stack/concatenate metrics for multi-metric training
    a. Go to jupyterlab
    b. Open concatenate_metrics.ipynb
    c. Execute the stack or concatenate part after adjusting folder names

8. Example_landscape_corine.ipynb
    a. Notebook to prepare smaller landscapes (i.e. example landscape) and bigger landscapes (i.e. corine) for inference by padding, cropping, renumbering, normalizing

9. Speed Test loading times
    a. Go to jupyterlab
    b. Open dataformat_speedtest.ipynb
    c. Execute the code
    d. Look at time

10. Convert landscapes to binary maps
    a. Go to jupyterlab
    b. Open Landscapes_to_binary.ipynb
    c. Execute the code to generate binary landscapes

Predict landscape metrics

11. Train UNet for metrics
    a. Go to jupyterlab
    b. Open LMUnet_nice.ipynb
    c. Set the right lossfunction (i.e. huber loss with nans)
    d. Set location where to save hyperparams
    e. Set TRAIN_MODEL = True and TEST_MODEL = False and PRETRAINED = False
    f. Restart Kernel & Run all cells (>>), saving the model at every other step

12. Test trained UNet metrics (Infer, evaluate, visualize)
    a. Go to jupyterlab

b. Open LMUnet_nice.ipynb (also allows for datasets)
c. Load the landscapes and target metrics to test
    i. Make sure the directory names and metric list is correct!
d. Set TRAIN_MODEL = False and TEST_MODEL = True and PRETRAINED = True
e. Set PRETRAINED_PATH for model to test.
f. Restart Kernel & Run all cells (>>)
g. Look at visualizations at the end of the file

Or:

h. Go to jupyterlab
i. Open LMUnet_LSM_inference.ipynb (individual landscapes only)
j. Make sure the directory names and metric list is correct!
k. Set TESTING LANDSCAPE_NAME and METRIC_NAME
l. Set PRETRAINED_PATH for model to test.
m. Restart Kernel & Run all cells (>>)
n. Look at visualizations
o. Denormalize and add NaNs
    i. If necessary, remove padding and put sliced parts together again
p. Compare denormalized metrics
q. Look at visualizations at the end of the file

Optimize Landscape

13. Optimize one Landscape inversely (“conventionally”)
    a. Go to jupyterlab
    b. Open the LMUnet_inverse_final.ipynb
    c. Load the target landscape and it’s metrics
        i. For example via selecting one NLM/LSM pair from the test dataset
        ii. Or setting TESTING_MODE to “example landscape”
        iii. Or to ‘corine’
    d. Set Pretrained path to correct trained model
    e. Initialize the random noise landscape
    f. Run loop
    g. Visualize (ls and lsm), evaluate r2, save

14. Optimize one Landscape inversely (“LPbP”)
    a. Go to jupyterlab
    b. Open the LMUnet_inverse_final.ipynb
    c. Load the target landscape and it’s metrics
        i. For example via selecting one NLM/LSM pair from the test dataset

ii. Or setting TESTING_MODE to “example landscape”
iii. Or to ‘corine’

d. Set Pretrained path to correct trained model
e. Add to optimizer and enable gradient either:
    i. directly the conventionally optimized landscape right after generation (AFTER_CONV = True)
    ii. Load previously conventionally optimized one (AFTER_CONV = False)
f. Run loop (takes a few hours, depending on how many LPbP_epochs)
g. Visualize (ls and lsm), evaluate r2, save

15. Plot line charts and barplots

a. Go to jupyterlab
b. Open plot_loss_rates_and_R2_bars.ipynb
c. Run line_plot_multiple for line charts of loss, eval scores or learning rate
d. Run baR2plots to plot r2 scores
    i. For this, us load_r2 to load r2 eval scores from file

Python scripts used in the notebook operations

- Dataset_test_many_metrics.py
    - Includes the LandscapeMetricsDataset to load the testing, evaluation and training data arrays, convert to tensors and modify it if so desired:
        - Normalize
        - Add noise
        - Padding
- Dataset_test_png.py/dataset_test_csv.py
    - Datasets for speed testing loading times
- DenormNameApplyNan.py
    - Function DenormNameApplyNan to denormalize the predicted LSMs and add the nan mask where necessary
    - Function ReplaceHighValues to reorder classes in an array
- LMUnet_connected_component_labeling.py
    - Includes the CCL function
    - Includes function to find biggest patch
- LMUnet_custom_loss_functions.py
    - Includes the MSE_nan and huber loss nan custom loss functions, which are used
- LMUnet_graph_operations.py
    - Includes add edges, to add edges between nodes, representing adjacencies between respective patches
    - Function get_largest_neighbor to find the largest neighbor based on the edged graph
- LMUnet_optimize_one_landscape.py

    - Optimization function for conventional and int constraint approach to inferring landscapes
    - Includes function to apply integer constraints
- LMUnet_r2_one_by_one.py
    - Includes r2_nan, to calculate r2 for one pair of metrics, masking nans
    - Includes an evaluation function to calculate R2 Score for each metric one by one, with option of saving the results to .csv
- LMUnet_visualization_functions.py
    - Visualize function to visualize 2D numpy arrays/torch tensors.
    - Includes options for discrete colorbars and to save the plots
- Unet_num_metr.py
    - The U-Net itself, with it's classes block, encoder and decoder
- PaddingEtc.py
    - Function pad_properly to pad arrays to multiples of target shape
    - Function crop_landscape_oversize to crop a larger array into target shapes
    - Function UnStack to combine cropped partial arrays into one full one again
    - Function Uncut to remove the padding

Csv files

- Mean_std.csv
    - List of all calculated mean and std values for normalization,
    - including 0 and 1 for binary mask
- metr_norm.csv
    - Lists all metrics wither their respective position
- Nan_metrics.csv
    - Includes all metrics where the nana mask si to be applied

Data

- data/100k_joinent_5_class_metric_list.csv
    - List with all names that have been used for training and testing (irrespective of actual metric name)
- data/multi_metrics_66.csv
    - List with names of all metrics.
- data/example_landscape
    - Example landscape and corresponding metrics etc.
- data/100k_5cl_multi_metrics_66
    - Main dataset, training, val, testing, 57 metrics
    - Includes nan stacked
- data/100k_landscapes_5_class
    - Main landscapes
- data/example_landscape_corine
    - Corine Göttingen data

Outputs

- Landscape_testiing
    - Results for landscape optimization
- Saved_models
    - All saved models and corresponding lists
    - 100k_5cl_multi_metrics_66_pad_zeroes
        - Original main trained one with R2 of ~95%
    - 100k_5cl_multi_metrics_66_adjusted
        - Further training of multi_metrics_66_pad_zeroes
        - With LR reduce on plateau to R2 of 97%
    - 100k_5cl_multi_metrics_66_one_hot
        - one hot encoding; R2 96%
    - 100k_5cl_multi_metrics_66_ls_norm_reflect
        - Normalized landscapes and reflect padding
    - 100k_5cl_multi_metrics_66_ls_norm_zeroes
        - Normalized landscapes and zero padding
    - 100k_5cl_multi_metrics_66_reflect
        - Reflect padding
    - 100k_5cl_multi_metrics_66_add_noise_reducing_lr_too_much
        - Added noise when training

Instructions for HPC

Required R packages very installed via github using devtools and withr. These included: *Tidyverse, NLMR, landscapemetrics, landscapetools, raster, Rslurm*

SSH tunneling to connect to the HPC was done via the 32 core R server. Work was done on gwdu102. Required R packages very installed via github using devtools and withr. These included: *Tidyverse, NLMR, landscapemetrics, landscapetools, raster, Rslurm.* To adjust the script from furrr to rslurm to be able to submit jobs using sbatch a few things had to be adjusted. Priority was immediate in the beginning, leading to up to a day+ of waiting times later on. During holidays many nodes were free so a lot of progress was made during that time.

Rough HPC workflow:

1. Landscapes and the script are transferred from R server to *home*.
2. Metrics are output to *scratch*.
3. First SSH into HPC, then
4. Copy metric generation script to HPC (with correct metric)/edit it on HPC (Nano gen -> tab,
    a. Hashtag previous metric, un-hashtag next one) and
5. execute it in R on HPC server (open R (R + enter) → source("genera…..R")).
    a. This generates a _rslurm_*metric_name* folder (Cd _rslurm_lsm…etc)
6. to which to navigate and then
7. run: sbatch submit.sh (multiple queable!~4 total, 3 queued).
    a. After 5-10 hours an Email notifies of the change.
8. Then navigate to /data/metrics/ folder (Moved to scratch),
9. zip the metrics by using:
    a. zip -r <metric_name>.zip <metric_folder>,
10. Download metrics.zip by transfer:
    a. scp -r <uname>@transfer-scc.gwdg.de:~/data/metrics/<metric_name>.zip ./.
11. For backup make sure to send the Zip to archive:
    a. mv -i <metric_name>.zip $AHOME/
12. and to keep scratch empty, make space by deleting both original file
    a. rm -rf <metric_folder> and zip
    b. rm <metric_name>.zip
13. There they were compressed with pigz, then
14. transferred back to the R server and
15. moved to the archive AHOME tape robot archive as backup.
16. The original, uncompressed, metrics on scratch were deleted to free up the space.