GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

Uploaddatum: 13.10.2023
Uploadzeit: 08:55

Dies ist ein von FlexNow automatisch beim Upload generiertes Deckblatt. Es dient dazu, die Arbeit automatisiert der Prüfungsakte zuordnen zu können.

**This is a machine generated frontpage added by FlexNow.
Its purpose is to link your upload to your examination file.**

Matrikelnummer: 23301512

Learning to analyse and synthesize landscapes for ecological simulation

Lernen, Landschaften für ökologische Simulation zu analysieren und synthetisieren

Author:
Jonathan Gehret

Matriculation number:
23301512

Master's Thesis at the Faculty for
Forest Science and Forest Ecology of the
Georg-August University Göttingen

Date of Submission:
13.10.2023

First supervisor:
Prof. Dr. Kerstin Wiegand

Second supervisor:
Prof. Dr. Fabian Sinz

Abstract

Landscape metrics are measures to quantify landscapes in landscape ecology. They are traditionally computed by mathematical methods from neutral landscape models. This however is costly in time, hardware and energy. To solve these constraints, experiments with Deep Learning computer vision methods, namely a modified U-Net-based architecture, are undertaken. This thesis examines the feasibility to predict such landscape metrics for any neutral landscape model using aforemnetionend technology, and do so more efficiently than previously. Furthermore, the inverse optimization of a neutral landscape model from given landscape metrics is investigated. The results show that landscape metrics closely matching the target values can be predicted, with magnitudes in improvement in generation time. Predicting landscapes from metrics is of limited success for now, however the viability is displayed. Building on these findings, numerous future inquiries are thinkable.

Kurzfassung

Landschaftsmetriken sind Maßeinheiten um Landschaften in Landschaftsökologie zu quantifizieren. Sie werden traditionell mittels mathematischer Methoden aus neutralen Landschaftsmodellen entwickelt. Dies ist jedoch kostenaufwändig, was Zeit, Hardware und Energie angeht. Um diese Beschränkungen zu lösen, werden Experimente mittels Deep Learning-basiertem maschinellen Sehen durchgeführt. Darauf basierend untersucht diese Arbeit die Machbarkeit, solche Landschaftsmetriken für beliebige neutrale Landschaftsmodelle zu erstellen, und zwar effizienter als zuvor. Weitergehend wird die inverse Optimierung eines neutralen Landschaftsmodels aus gegebenen Landschafts-metriken erforscht. Die Ergebnisse zeigen, dass beinahe die Zielwerte erreichende Landschaftsmetriken vorhergesagt werden können, mit Magnituden in Verbesserung was die Berechnungszeit angeht. Landschaften aus Metriken vorherzusagen ist bisher von eingeschränktem Erfolg, jedoch wird die Durchführbarkeit aufgezeigt. Auf diesen Befunden aufbauend sind zahlreiche zukünftige Untersuchungen denkbar.

Table of Contents

Abbreviations

CCL……..Connected component labeling

CNN……...Convolutional neural network

COL……...…Conventionally/continuous inversely optimized landscape

HL……Huber Loss

LPbP……Largest patch by patch labeling

ls……...…Landscape/Neutral landscape model

LSM……(Landscape) metric

NaN……...……...…Not a number

NLM……...……Neutral landscape model

r2/R2…..……...Coefficient of determination

1 Introduction

1.1 Neutral Landscape Models

Neutral Landscape Models (NLMs) are used by landscape ecologists to classify landscapes. NLMs are two dimensional arrays consisting of different classes. The landscapes (ls) are made up of patches of connected cells which belong to different classes each. One class can be a forest, a field or any other type of structural element. The minimum number of classes is two, i.e. habitat and nonhabitat. Nonhabitat, i.e. background, is usually called matrix. Landscape configuration and landscape composition describe how the landscape is characterized. While real landscapes consist of continuous change in surface cover, categorical landscapes models (Fig. 1) are created to more easily characterize landscapes in useful ways (Gardner et al., 1987). While models based on real landscapes can be created (Salecker et al., 2019), tools like NLMPy or NLMR allow for the simulation of entirely artificial landscapes (Etherington et al., 2015; Sciaini et al., 2018), enabling the "virtual ecologist approach" (Zurell et al., 2010) , i.e. running ecological simulation experiments to study how biodiversity is impacted by land use (With & King, 1997).

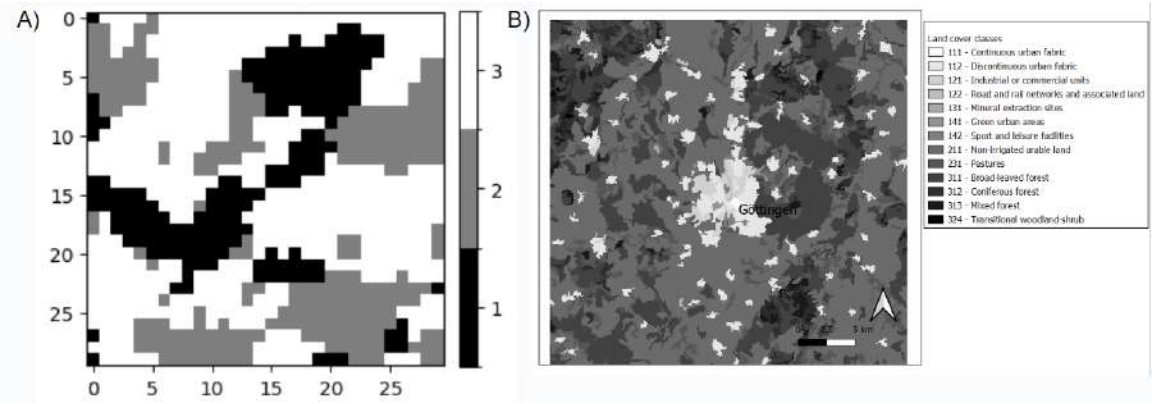

Fig. 1. Example NLM with 3 classes (A); Corine land cover of the greater Göttingen area with 13 classes (B).

1.2 Landscape Metrics

One way to gain insights from NLMs are Landscape Metrics (LSMs, Fig. 2B). These are measures generated to quantify landscape composition and observe changes in spatial and

temporal processes (Turner & Gardner, 2015). They are calculated for purposes such as tracking land use changes, analyzing biodiversity or even water quality among others. Numerous different LSMs exist, numbering into the hundreds (Uuemaa et al., 2009). They can be calculated on patch-, class- or landscape level (Chen & Iannone III, 2020; Hesselbarth et al., 2019). Metrics have been categorized into different categories based on various approaches. One of these categorisations splits them into area and edge metrics, concerned with patch size and edges. Examples for these more basic metrics include total area or total edge; shape metrics, working with the area and perimeter of patches, such as shape or contiguity index; core metrics like core are or core area index for calculating those parts of patches that are not an edge; aggregation metrics about the spatial distribution of patches of the same class - this includes number of patches, patch density or connectance; diversity metrics, such as simpson's and shannon's diversity and evenness indices, for showing the abundance, dominance and rareness of classes; sometimes a further category complexity metrics is also included where LSMs such as shannon, conditional or joint entropy are grouped in with. Additionally, for a number of these metrics a mean, a standard deviation (sd) and a coefficient of variation (cv) are available (Appendix Tab. 1.; Hesselbarth et al., 2019). Many of these metrics correlate strongly with each other requiring thoughtful consideration when deciding which ones to work with (Cushman et al., 2008).

One specific way to calculate metrics from NLMs is to run a moving window over the landscape and calculate a landscape-level metric, filled into the center pixel, for every one of these window-sized partial landscapes (Hagen-Zanker, 2016). For general LSM generation and window-LSMs in particular, stand-alone software FRAGSTATS (Chen & Iannone III, 2020) or the R package landscapemetrics can be used (Hesselbarth et al., 2019).

Among the metrics are Simpson's diversity index (SIDI):

$$SIDI = 1 - \sum_{i=1}^{m} P_i^2$$

Eq. 1: SIDI, where Pi is the proportion of class i and m is the number of classes (Hesselbarth et al., 2019).

or also Percentage of Like Adjacencies (PLADJ):

$$PLADJ = \left(\frac{g_{ij}}{\sum_{k=1}^{m} g_{ik}}\right) * 100$$

Eq. 2: PLADJ, where gii is the number of adjacencies between cells of class i and gik is the number of adjacencies between cells of class i and k (Hesselbarth et al., 2019).

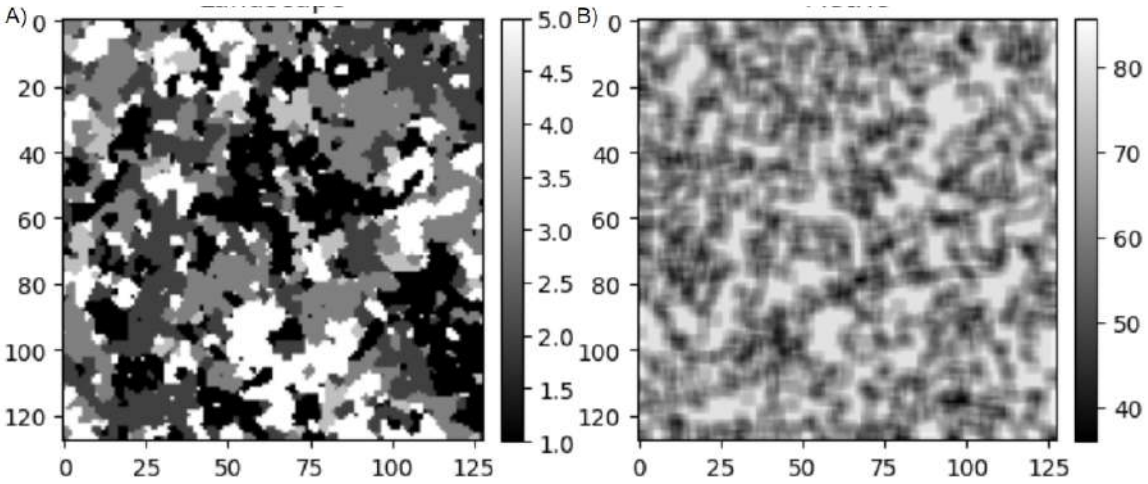

Fig. 2. Test dataset NLM 10 (A), 128x128 with 5 classes. Corresponding example metric pladj (B).

Conventionally, only a small number of metrics for any given use case are calculated (Uuemaa et al., 2009). To determine which ones to use can be achieved by doing a Principal Component Analysis (Cushman et al., 2008).

1.3 Deep Learning and U-Net

While it is possible to generate both NLMs and the respective LSMs (Fig. 2) with the tools described above, an attempt to reduce the computational cost by use of deep learning is made. The deep learning methodology of training a model by calculating a loss from given training data and backpropagating it through the network, updating the weights and hereby optimizing the model is commonly applied in Convolutional Neural Networks (CNN). These are used in computer vision tasks to identify contents of images (Z. Li, Liu, et al., 2022). One popular variant of CNNs is the U-Net for image

segmentation tasks, using a moving window as well (for similarities in terminology between deep learning and landscape ecology, compare appendix tab. 2). It learns to generate an output segmentation mask by first downsampling and then upsampling the input image, chaining (de-)convolutional and pooling layers. Skip connections feed intermediate results from the downsampling path to the upsampling path (Ronneberger et al., 2015). CNNs in the broader sense (Brodrick et al., 2019) and U-Net particularly (Wagner et al., 2019) have been tried and used for ecological tasks, for example in the case of recognizing forests or even individual trees, i.e. oil palms from remote sensing imagery (Freudenberg et al., 2019). To predict metrics, consisting of arrays of continuous values which are not akin to segmentation masks, the U-Net was adapted from image-to-image translation to image regression. In such a task, the goal is to predict a continuous value for each pixel in an image (Angelopoulos et al., 2022).

Deep neural networks can also be worked with to solve the inverse problem. This means predicting the input data from given outputs (Ongie et al., 2020). In this case, instead of optimizing the weights of the network itself, only the input data is set to track gradients and to be optimized. The original weights are left frozen. To predict a fitting landscape from metrics, the approach to solve the problem inversely is applied.

Applying these deep learning approaches, a faster and more efficient alternative appears possible. Consequently, key questions to address include: Can deep learning methods enable a way that LSMs can be reliably predicted given any NLM? If so, is it also possible to inversely predict the corresponding NLM given any number of LSMs from this trained model? Furthermore, is it faster and less computationally costly this way?

2 Material and Methods

2.1 Pre-considerations

In the investigation of loading times for different data formats, namely Comma Separated Values (.csv), Portable Network Graphics (.png), and the numpy file format .npy, when used with a PyTorch dataloader, it was observed that .npy outperformed both .png and .csv in terms of loading speed. As a result of this comparative analysis, the .npy format was selected as the preferred choice for data storage and retrieval (OpenAI, 2023b).

Typically, Deep Learning models operate efficiently with data in float32 format (Johnson, 2018). This choice is well-suited for the present study, considering that the initially generated data were in float64 format, which occupies twice the storage space without substantially enhancing the relevance of the data. Specifically, a single metric dataset containing 100 thousand generated 128x128 matrices in float32 format consumes a mid single-digit amount of GB in storage space. Furthermore, when stacking 67 metrics together, the storage requirements exceed 300 GB (OpenAI, 2023b).

2.2 Generating Training Data

2.2.1 NLM generation

It was decided on 128x128 pixel wide, 5x5 moving windows and landscapes of up to 5 classes. Using a seed (42) for everything for reproducibility, in R and Python. The synthetic training data was generated using the parameters found in table 1 with the nlm_randomcluster algorithm of the NLMR R package (Saura & Martínez-Millán, 2000; Sciaini et al., 2018). The 100 000 training, validation and testing random cluster landscapes were generated first, the 100 000 metrics each one by one for all of these landscapes, together with an annotation file for landscapes and metrics each afterwards.

Tab. 1. Settings for landscape and metric generation

Parameters	Values
p	between 0.1 and 0.6
Number of classes	5 classes
Landscape Size	128x128 pixels
Window size	5x5
ai	0.1 to 0.9

p = proportion of elements randomly selected to form clusters

ai = number and abundance of land cover classes

2.2.2 LSM generation

All compatible landscape-level landscape metrics, excluding patch- and class-level metrics, are trained simultaneously so each of those metrics would be predicted quickly on demand. Only metrics which do not require additional input information, i.e. work with default settings, are ultimately included in this project.

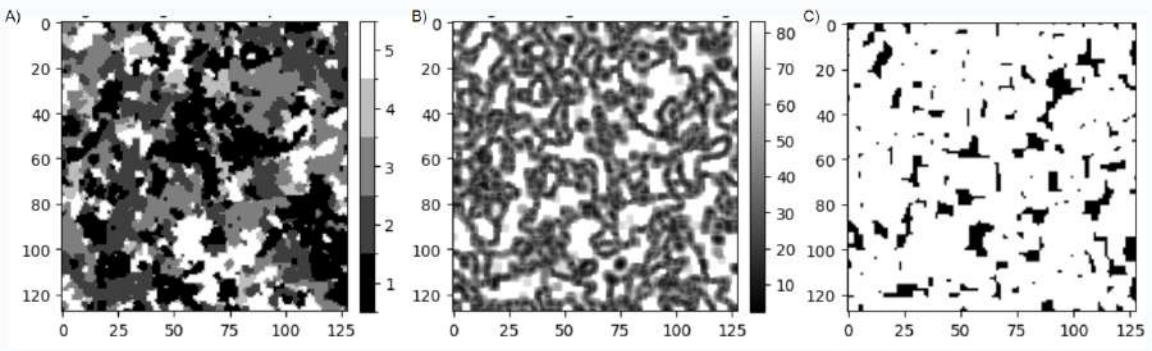

Fig. 3. NLM 95010 (A). Corresponding contag LSM with white NaNs (B). NaN mask (Black == NaN, white == Not NaN) (C).

Some of the metrics are defined to include NaNs, oftentimes in the case where the landscape consists of only one or two classes - which is the case frequently when every moving window landscape sizes 25 pixels. For these NaN values, a binary NaN mask (Fig. 3) is generated, where every pixel that would be filled based on a moving window including only one class is assigned a 1, every other one a 0.

Some metrics were not able to be generated with the chosen settings. Of all the metrics in the landscape metrics R package, the mean, standard deviation and coefficient of variance versions of the contiguity index (contig), the euclidean nearest-neighbor-distance (enn) and gyration (gyrate), as well as the coefficient of variation fractal dimension index are excluded due to various reasons: for the gyrate metrics, a cell center variable needs to be set. For the enn, at least two patches of the same class need to be present in the landscape, which is rarely the case in 5x5 sized landscapes.

Further metrics pafrac, rpr, iji and ta were not excluded but do not work well and can be filtered out manually. Reasons are that iji is defined for only 3 or more classes, which is oftentimes not the case if working with small windows of size 5x5. This prerequisite leads to more NaN values than the 2-class NaN mask used in this project. Pafrac on the other hand isn't defined for less than 10 patches, which in the case of 5x5 windows is the case more often than not. Even more NaN values, i.e. 100% of the rpr-metric, are created if the maximum number of classes parameter isn't set, which means all of them in this case. Lastly, in the case of ta (Total area), 25 is returned every time, because the total area of a 5x5 moving window will always be 25 (Appendix Tab. 1.).

Metrics were generated by loading the next landscape with reticulate numpy. This is done according to the annotation list. Based on the set hyperparameters, including which and how many metrics to compute, the 100 000 metrics are calculated one by one utilizing the parallelization software of choice, and then saved again to numpy format with reticulate. For this, multiple devices were employed (Tab. 2).

Tab. 2. Specifications of different devices used

	Local PC	R server	HPC
CPU cores	4	32	94 * (2x48)
RAM	8 GB	32 GB	384 GB
R version	4.2.0.	4.2.2.	4.2.2.
Parallelisation	furrr	furrr	slurm

For faster generation, parallelisation was utilized. One CPU core can execute one process at a time, given sufficient memory (in this case 450MB). Using complementary parallelisation libraries, i.e. the R packages fuuu, or slurm for HPC, a high reduction in computation time can be achieved. First using the R package fuuu and 32 cores some metrics were generated, later using HPC with slurm and many more potential cores were utilized. For this the script was adapted to work with slurm, the jobs were queued after tunneling into the HPC via SSH.

2.3 Train U-Net for metrics

2.3.1 Training preparations

As some metrics consist of rather high values (Fig. 2B), for each of them, mean and standard deviation values for normalization are calculated and saved in a file mean_std.csv. These metrics are then consolidated into a single tensor of dimensions nx128x128, with the NaN mask positioned at the 0th index. It's worth noting that metrics can be flexibly added at desired dimensions or appended to either the beginning or end when training new models, however not so much when working with pre trained ones. Conversions between tensors and NumPy arrays, including operations like detaching and adjusting gradients, as well as converting to float format, are performed as needed, along with the use of squeeze and unsqueeze operations. This process requires an annotation file (in this case “metr_norm.csv”), containing the names of all metrics to be stacked, assigning to each position in the concatenated array the corresponding metric name. This list can later be used as a key to assign the correct names to the predicted LSMs, based on

the respective position in the output tensor (column P. in Appendix Tab. 1.) (OpenAI, 2023b).

The stacking process for seven sets of 100,000 metrics took approximately 1,000 seconds, while stacking more sets (33) required 3 378.987 seconds, representing a slightly faster operation. Notably, augmentations or transformations, such as cropping or rotation, have not been applied in this specific scenario; however, they can be employed to augment the training dataset size if deemed necessary (OpenAI, 2023b).

2.3.2 U-Net

All the computational work, which encompasses Python programming, PyTorch-based operations, including tasks such as training, testing, and optimization of the U-Net model, was conducted within a Jupyter Docker environment. The foundational code base for this work was significantly modified but remains rooted in the original source. This is true for the code for the U-Net itself, where multiple blocks of convolutions and ReLU are chained first by the encoder and then the decoder (Aman Arora, 2020b), as well as for the code used for training and inference (Aman Arora, 2020a; OpenAI, 2023b).

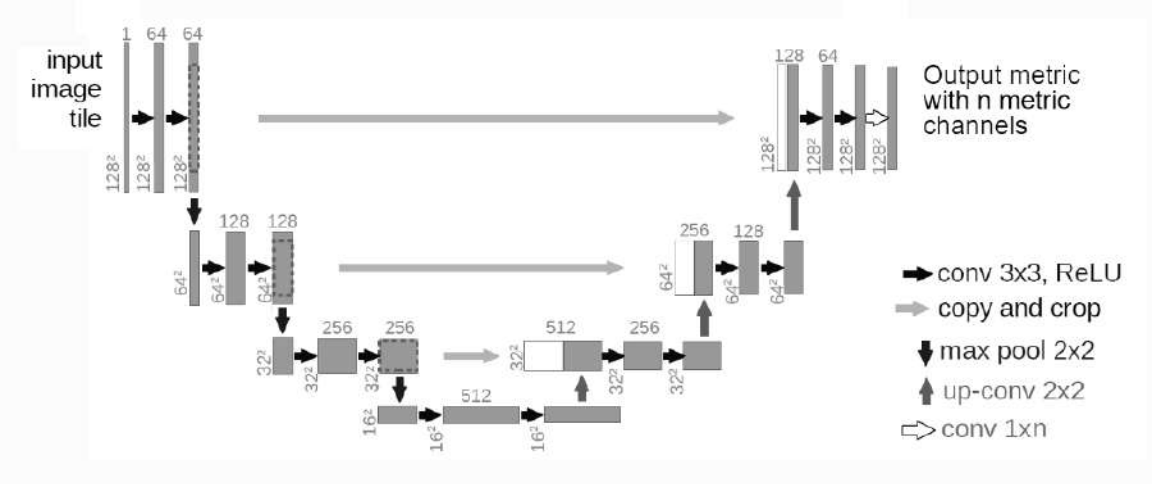

Fig. 4. U-Net, adjusted from Ronneberger et al., 2015.

The U-Net architecture underwent several adjustments to align with the specific requirements of this study (Fig. 4). Given that the input images have dimensions of

128x128, one convolutional layer less than the original design was employed. Particularly, the first layer was omitted, resulting in a maximum channel amount of 512. Additionally, the output is as many channels n wide as metrics were trained for, in this case 57 (OpenAI, 2023b).

To adapt the model for image regression, the omission of the final classification layer, the removal of the binary mask, thereby working directly with raw values, and the adoption of a regression loss function are introduced. Several loss functions were tested, including Mean Squared Error (MSE), Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and Huber Loss (HL). Ultimately, Huber Loss was selected as the most effective choice. Model validation was assessed using the coefficient of determination (r2), with values approaching 1 indicating a good fit, or alternatively, MSE with values nearing zero signifying better performance (OpenAI, 2023b).

The model's initialization employed default settings for PyTorch Modules, particularly for weights. The model was trained to accommodate up to five classes but was also evaluated on landscapes with fewer classes (e.g., 3 classes in the example landscape) and more classes (e.g., >20 classes in the corine dataset). Data loading was facilitated through a custom PyTorch DataSet class, which included on-the-fly metric normalization as well as leveraging the default PyTorch DataLoader for efficient GPU-based data loading (OpenAI, 2023b).

The dataset underwent random splitting into training, validation, and test sets, with a consistent random seed of 42 employed for reproducibility. This seed is especially critical as it determines the composition of the test set. Common split ratios included 80k for training, 15k for validation, and 5k for testing. The PyTorch DataSet and DataLoader were utilized to expedite data loading directly onto the GPU (OpenAI, 2023b).

An essential data transformation within the dataset involved converting numpy arrays to tensors. It is important to highlight the diversity in metric values, ranging from 0 to 1 for metrics like Simpson's diversity index to higher values exceeding hundreds for others.

Consequently, these values were normalized using pre-computed mean and standard deviation values, stored in a CSV file used for training within the dataset (Fig. 5). After making predictions for new metrics, denormalization was easily achieved using the same mean and standard deviation values. Additionally, landscapes and metrics of smaller dimensions (i.e. smaller than 128x128) were padded to meet target dimensions. While larger dimensions were feasible, practical limitations for processing large landscapes necessitated the splitting of these larger landscapes into smaller segments (OpenAI, 2023b).

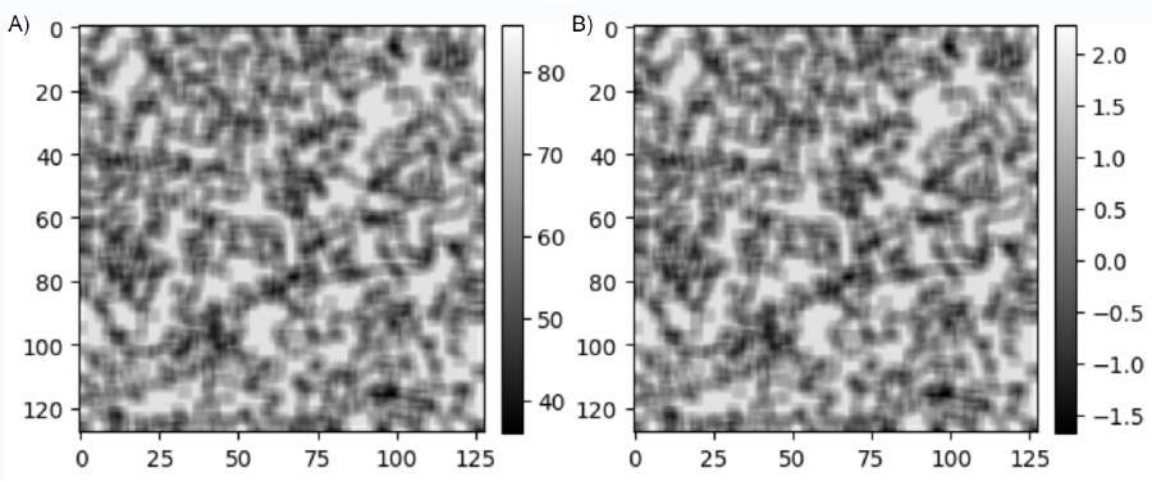

Fig. 5. Example pladj LSM for testing LSM 95010: original target (A) and normalized (B).

The activation function used was ReLU, working in PyTorch with the Adam optimizer. The batch size was set to 16, as that is as fast as 32 while 8 takes twice as long. The U-Net convolutional layer kernel size was set to 5x5, the same size as the moving window which is used to calculate the metrics in the original approach. Input: landscape with one channel. Output: metrics with one channel for each metric.

The fill value which is used for the window metric calculation in NLMR, is NA. However, when using padding in U-Net for the convolutional kernels using NaN is not possible. Alternative options like reflect, replicate or even circular exist, however padding with zeros of size 2 was used, to keep the dimensions when using the 5x5 Kernel (Tab. 3).

Tab. 3. Select Model Hyperparameters

	Model training	**Landscape optimization**
Batch size	16	1
Initial learning rate	0.00002	0.1
Loss function	HL (masking NANs) (alternatively MSE, MAE, RMSE)	
Eval Score	R2 Score (alternatively MSE)	
Window size	5x5	
Padding	2	
Output Channels	As many as metrics (i.e. 56 + 1)	
Epochs	50 +	Thousands

Both loss and validation functions are adjusted to skip the inevitable NaN values by masking. As both target metric and predicted metric (and by extensions also the input landscape) are required to have the same shape, the mask is created based on all NaN values present in both landscape (usually none there) and target metric. This mask is then laid over the target metric and the predicted metric for computation of the loss.

2.3.3 Training

Early experiments showed that it would be possible without (noticeable) loss in performance to train a model to predict multiple metrics simultaneously in one stacked output tensor. The model is trained for 56 metrics plus NaN mask until no more significant improvement is measured using those hyperparameters by letting it learn to predict the generated metrics from the generated landscapes. During training the model is validated on the 15 thousand item validation dataset, afterwards it is tested on the Testing dataset of 5 thousand NLM/LSM pairs afterwards to verify that the model has been taught as intended (Fig. 6).

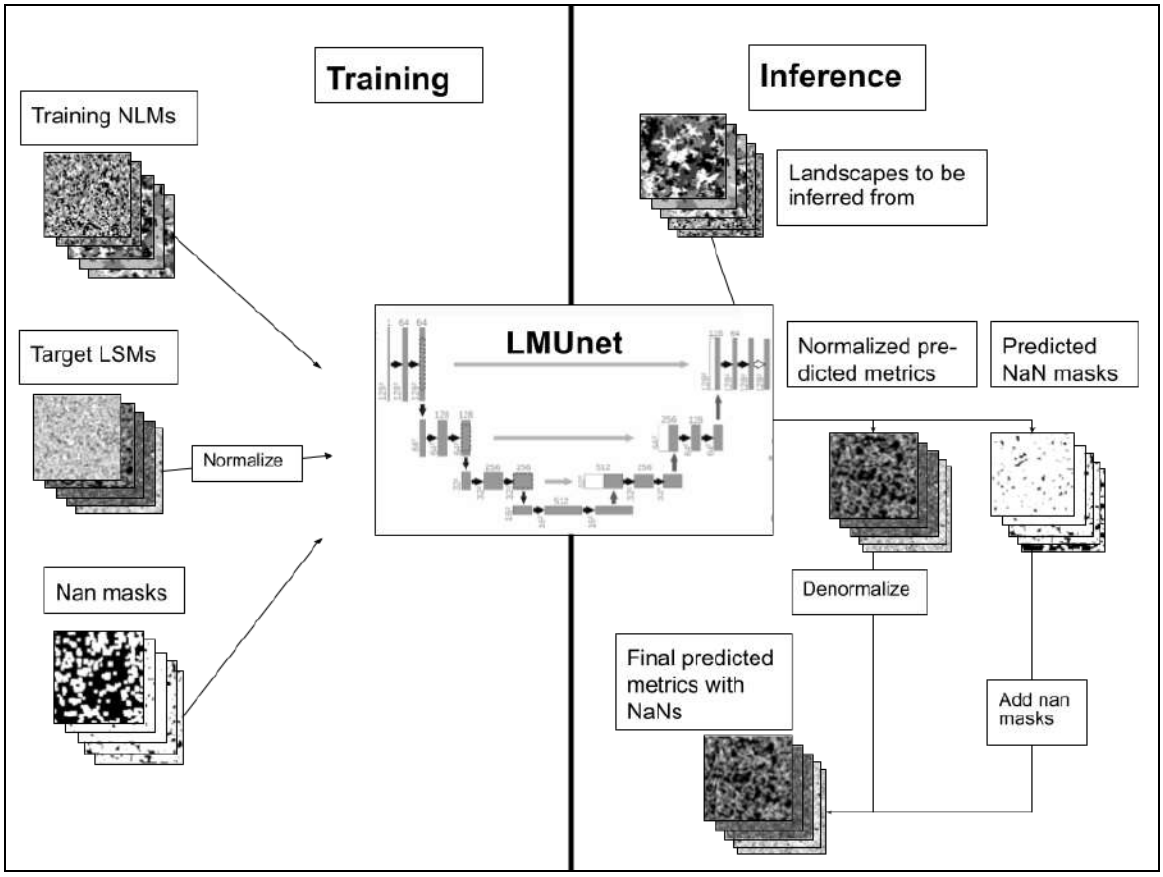

Fig. 6. Training and inference of the modified landscape metrics U-Net (LMUnet).

A number of different variants were trained. The default one, on which most of the experiments in this thesis have been worked on, is labeled pad_zeros. As the name suggests, it includes convolution layers which utilize padding with zeroes to retain the input dimensions. Distinctively, the variant named reflect is utilizing padding with reflect instead. Analogously, both of these have also been combined with landscape normalization, i.e. normalizing the input landscape classes to values between 0 and 1. Two further versions to be mentioned here are the add_noise variant, where before training random values between -0.5 and 0.5 are being added to the input landscapes, in an attempt to facilitate discrete value prediction. A last variant one_hot was trained not with inputting landscapes NLMs consisting of the class values 1 through 5 in one channel, but instead 4 input channels, one for the classes 1 to 4 each, where every patch of that class is labeled with 1 while the rest of the landscape is labeled with zeroes (Fig. 8B/C)

Once the NLMs and the NaN mask have been predicted, the latter is rounded to 0 and 1 and then for all metrics where NaNs are required, these are filled in for by replacing the predicted values with NaN in wherever the NaN mask equals 1. As the predicted metrics will consist of the normalized values, they are denormalized for use (Fig. 14). In the case that the input NLM is of smaller dimensions, the padding is cropped away to original size. If it was bigger, the predicted partial LSMs are merged again.

2.3.5 Testing smaller and bigger landscapes

For smaller landscapes multiple approaches can be considered. As the metric calculator depends on the cell size, simply interpolating it to fit the desired dimension is not plausible. Instead, padding the landscape to the input size is more realistic. After prediction is done, it can be cropped again to return the original size array. Padding can be done using 0 or also reflect or extend. In this study, reflect was chosen for padding as it seemed the more suitable choice, especially to produce more accurate results for the borders of the landscape. NaN padding, as is done in the original calculator using R, is not a viable option as in PyTorch NaNs need to be masked which would defeat the purpose of padding and shrink the size. Exemplary for smaller landscapes with fewer classes (i.e. 3) the example landscape from the neutral landscape models R package has been chosen for demonstrative purposes, in the following simply referred to as example landscape (Fig. 7A/B).

In the case of bigger landscapes, firstly also padding the landscape to fit a multiple of the desired size can be helpful before splitting it into multiple smaller ones of the input size. These parts can then be stacked into one tensor at the batch channel location, enabling efficient generation of all the parts simultaneously. This also theoretically allows for overlap between the pieces to reduce boundary inaccuracies. Padding and splitting could be integrated into the dataset class, however in this study, separate functions to achieve this were used. After prediction, the slices are merged, with respect to overlaps if those were implemented. Eventual paddings are removed.

To showcase this approach for working with bigger landscapes and also working with landscapes of > 5 classes, a cutout of the greater Göttingen region from the Corine Land cover dataset was prepared. The full dataset, including a legend, can be accessed freely (European Union & European Environment Agency (EEA), 2018). From this, an area including Goettingen and surrounding was clipped from it with an extent roughly three times the size of the 128x128 default dimensions (Fig. 1B). This array was then adjusted in size by the methods described above to create 9 128x128 arrays for inference with U-Net (Fig. 7C).

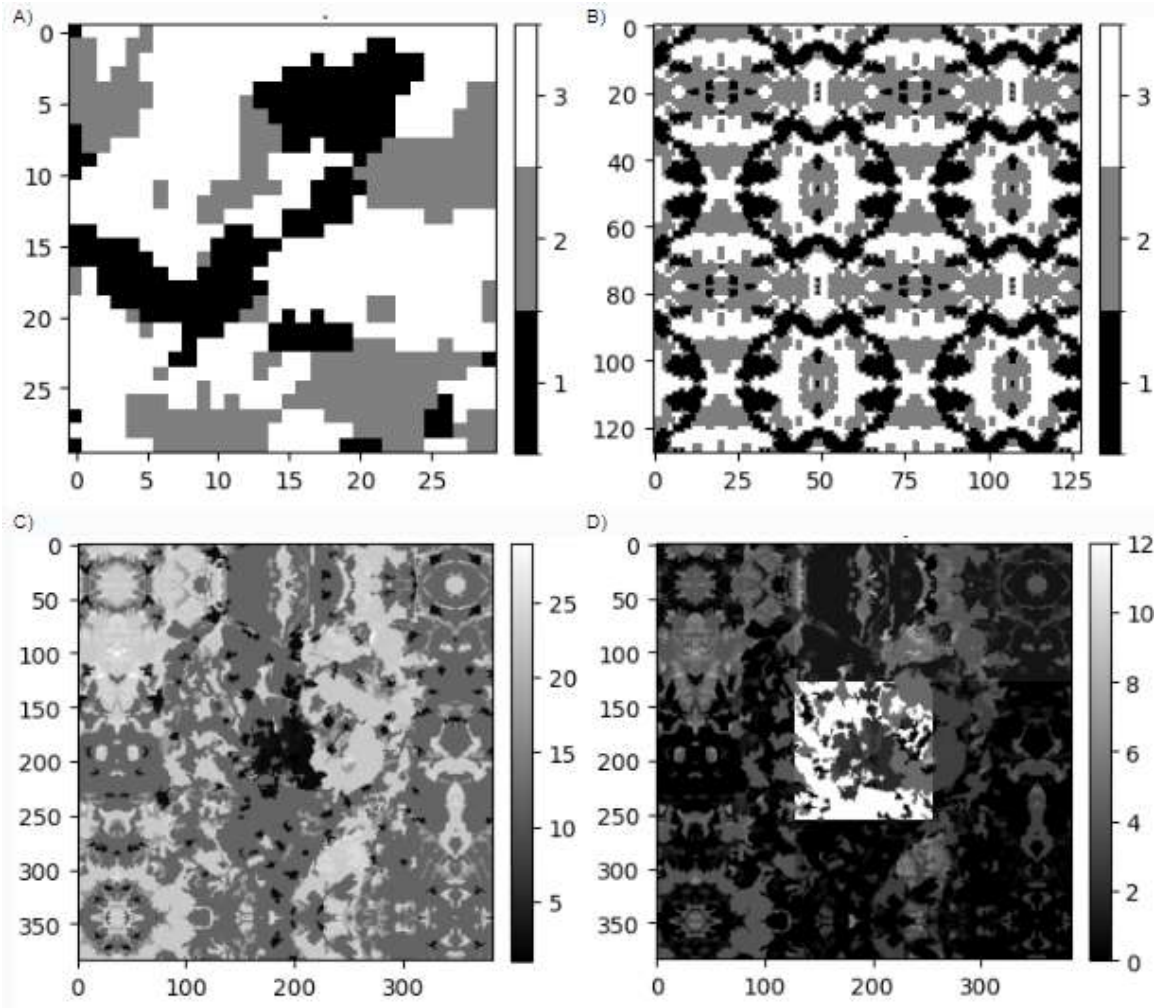

Fig. 7. Example landscape (A) padded to n*128 (in this case 1*128) (B). Corine Göttingen, padded with reflect to 9*128x128 (C) and the 9 corine cutouts renumbered to values 0 through 12 and put together again. Note the higher values in the center piece (D).

Different variants for dealing with the up to 13 different classes of varying two-digit values were tested: The corine cutout NLMs were input without any changes into the trained model. Alternatively the NLMs were renumbered starting with 0, rising consecutively, so no empty classes would remain, with the central, urban area consisting of more different classes compared to the surrounding more natural areas (Fig. 7D). A third option was to normalize corine and input it into the model that was trained with normalized landscapes, as well as fourthly into the one trained with noise.

2.4 Inverse landscape optimization

2.4.1 Conventional optimization

The model in this study has been trained to generate metrics from landscapes. In order to investigate the inverse problem of generating landscapes from metrics, the model is loaded with frozen weights. Simultaneously, a randomly generated noise landscape is created, characterized by values ranging from 0 to 1, and is used as input (Fig. 8A). Only the randomly generated noise landscape is enabled for gradient computation and is incorporated into the optimization process (OpenAI, 2023b).

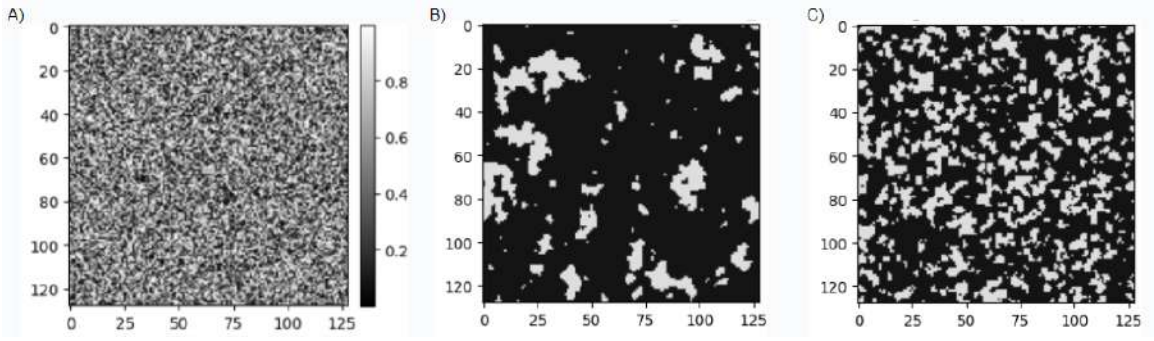

Fig. 8. Randomly initialized landscape consisting of random noise (A). Examples for binary partial NLMs, depicting one class each (B, C).

The model then proceeds to generate metrics based on this randomly generated noise landscape. At each iteration, the Huber Loss is computed by comparing the generated landscape with the desired metrics for which a corresponding landscape needs to be predicted. This random noise landscape is iteratively optimized to better approximate a landscape capable of producing the specified metrics (Fig. 9; OpenAI, 2023b).

The target metric can either be the original metric of interest or a metric predicted in a previous step. The landscape optimization process has been tested on metrics and landscapes from the test dataset, as well as the example landscape and the corine landscape. This approach enables the model to iteratively refine the noise landscape to produce landscapes that closely align with the specified metrics, thereby addressing the inverse problem of landscape generation from given metric data. In this study, the inverse problem solving approach is demonstrated on the test landscapes 1 through 10 from the testing dataset (landscapes 95001 to 95010 of the full 100k dataset if using the seed 42; both names are used interchangeably throughout this study). Primarily the main model trained with zero padding is used, as well as the one trained with noise and the one with one hot encoding. For testing purposes, the optimization approach is also tried on the original landscape as input data, instead of random noise (OpenAI, 2023b).

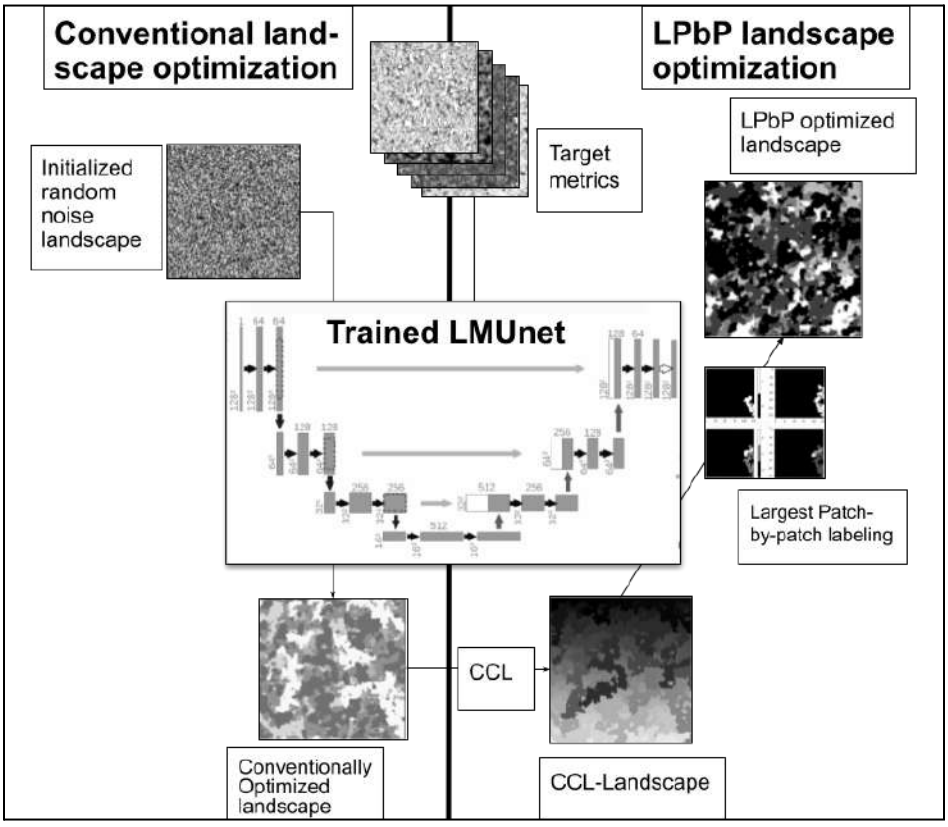

Fig. 9. Diagram of conventional and LPbP inverse optimization.

2.4.2 Digitizing the landscapes

As this previous conventional approach produces continuous float values, the conversion of predicted landscapes generated through conventional inverse optimization methods into discrete categorical values is a critical step in this research. Various strategies have been explored to accomplish this task, incorporating heuristic approaches to achieve the desired outcome (OpenAI, 2023b).

One straightforward approach involves rounding the continuous landscape values or discretizing them to integers based on predefined constraints. Additionally, a clamping procedure can be applied to ensure that values remain within a specified range, or outliers can be redistributed randomly to align with the desired categorical values. This process can be iteratively integrated within the conventional optimization framework to refine the landscape conversion (OpenAI, 2023b).

Alternatively, a more systematic method entails labeling the landscape patch by patch, typically starting from the top left corner and proceeding in a sequential manner to the bottom right. An even more comprehensive labeling approach involves considering the largest patches first and subsequently labeling their neighbors in a sequential manner (OpenAI, 2023b).

To implement the latter approach, a connected component labeling (CCL) algorithm is exemplarily utilized to identify and assign unique identifiers to individual patches depending on a divisor variable. Subsequently, the largest Patch-by-Patch (LPbP) algorithm is employed to generate landscapes that closely align with the desired categorical format. It numbers every one of these individual patches with a different integer value. Their size is counted and stored in a dictionary. To facilitate tracking and labeling of neighboring patches for this labeled landscape, a complementary graph with nodes connected by edges representing adjacent patches is generated (OpenAI, 2023b).

Then the LPbP optimization loop is started: Starting with the biggest patch, all n possible patch classes (i.e.: 5) are filled in as values for that patch into the CCL-landscape. Those

5 landscapes are then stacked into one tensor and run through the model. The same value as neighboring patches is allowed to enable patches to merge in case this yields better results. The version with the lowest loss score is retained. Subsequently, the conventional inverse optimization is run with this filled in landscape for a variable number of iterations. Next, utilizing the graph edges, of all neighboring patches the largest one is identified and the same steps as described are continued. This continues until all neighboring patches are filled. If no more unlabeled patch is found, the biggest one of all the remaining ones is chosen instead and the process continues with that one and its neighbors. After every patch has been iterated over, a landscape of discrete patches of the values of the n classes is returned. Finally, the complete process can be looped over again, using a new CCL after every round to generate better results (Fig. 9; Algorithm 1). This approach ensures a more systematic and controlled transformation of continuous landscape data into the desired categorical representation (OpenAI, 2023b).

Algorithm 1. Pseudocode for LPbP (OpenAI, 2023a).

```
for each epoch in 1 to EPOCHS:
    if C is empty:
        break
    current_patch = patch in C with the largest size
    for each label_num in 1 to N:
        Label(current_patch, label_num)
        Train model on labeled patch
        Calculate Loss(F, label_num)
        if Loss(F, label_num) < best_loss:
            best_loss = Loss(F, label_num)
            Update F with the label
    Update M with the label
    Optimize F
    current_patch = patch in C with the largest neighbor to current_patch
```

M: The final mask, indicating the labels assigned to patches.
F: The continuous landscape image being optimized.
C: The set of patches that have not been labeled yet.
best_loss: The lowest loss achieved during optimization.
current_patch: The current largest patch in C.
label_num: The label assigned to a patch.

The resulting discrete inversely LPbP-optimized landscapes are then visualized. The final LSMs are predicted from that landscape through the model. These can also be visualized. For comparison with the target metrics, the evaluation r2 scores are calculated and plotted. Additionally, from these generated landscapes the respective LSMs are calculated with the existing R software, from which the r2 values are plotted as well and compared with the ones predicted from the model.

3 Results

3.1 Results metric Training/prediction

3.1.1 Results training

Comparing the different training approaches described above, there are some noticeable differences. The default variant pad_zeroes reaches an R2 value of almost 0.95 after 50 training epochs, while most other variants reach lower values at that threshold, with the exception being one hot where a higher R2 is achieved (Tab. 4).

Tab. 4. Table comparing different training variants for Huber loss and r2 score

Method	Loss (50 epochs)	R2 (50 epochs)	Loss (x epochs)	R2 (x epochs)
ls_norm_zeroes	0.03	0.9361	0.0159 (x=154)	0.965 (x=154)
reflect	0.0379	0.9197	0.0372 (x=59)	0.9213 (x=59)
ls_norm_reflect	0.0292	0.9384	0.0214 (x=81)	0.9552 (x=81)
add_noise	0.0391	0.9133	0.0316 (x=156)	0.9282 (x=156)
one_hot	0.0203	0.958	0.0148 (x=99)	0.9684 (x=99)
pad_zeroes	0.0257	0.9471	0.0133 (x=230)	0.9708 (x=230)

After 100 Epochs the total R2 score yielded an average R2 score of ~95%. Training further while reducing the learning rate led to an average R2 of ~97% (Fig. 10B), with a Huber loss score of below 0.015 (Fig. 10A). The further percentages where achieved with a reduced learning rate (Fig. 10C).

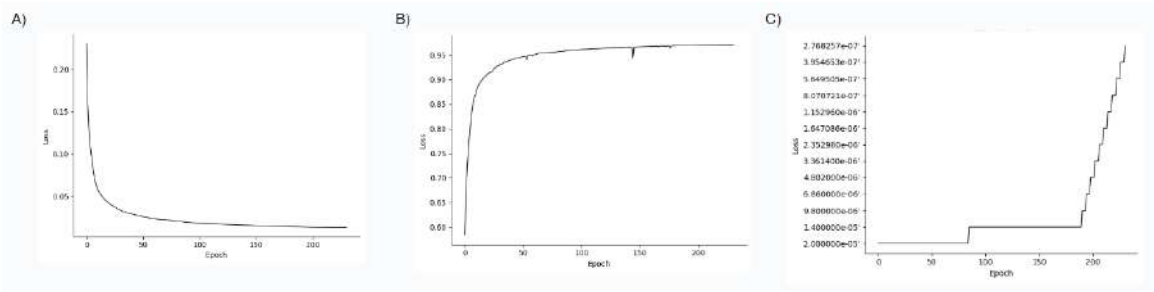

Fig. 10. The pad_zeroes HLtraining curve (A). The validation R2 score curve (B) and reduced Learning rate curve (C).

For most of the metrics in the testing dataset the r2 values for the predictions are close to 1. The NaN masks were generally predicted with high evaluation scores (>99%), as well as most of the metrics in the categories area and edge, core, aggregation (except for IJI), diversity and complexity. Noticeably, many of the so-called shape metrics, such as the lsm_l_circle, lsm_l_frac and lsm_l_shape metrics generally return lower values of 70%-90% (Fig. 11).

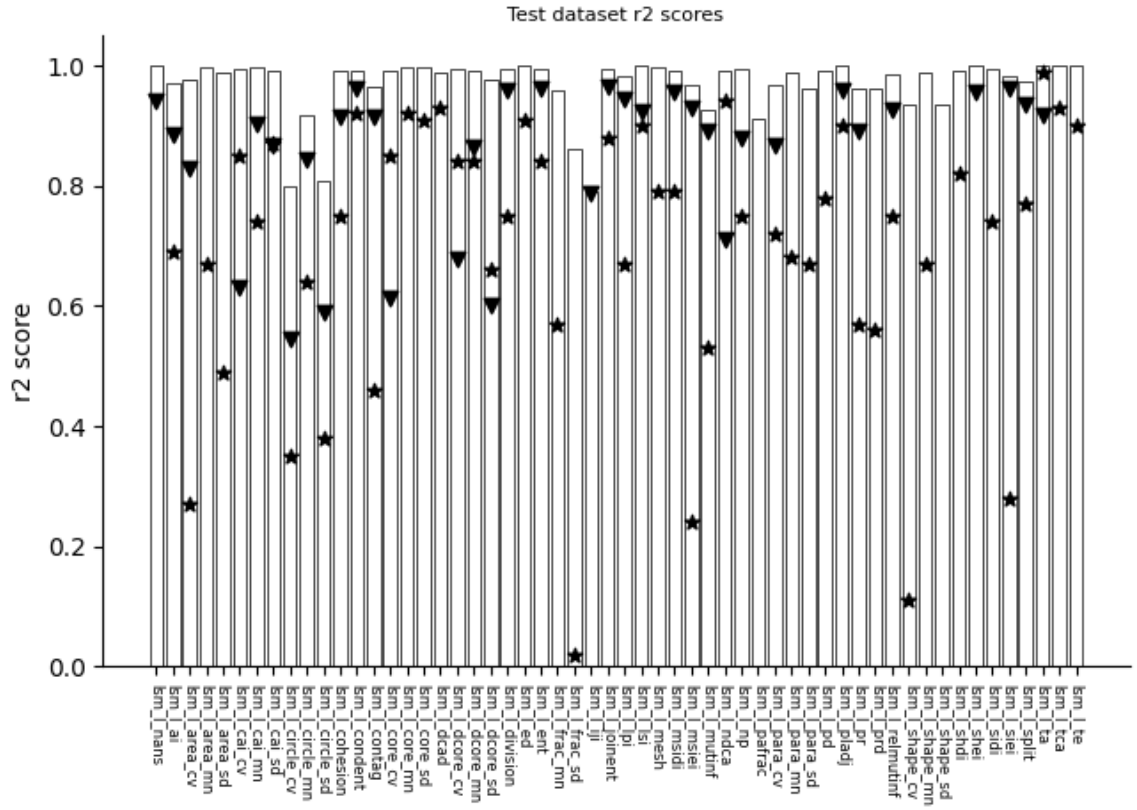

Fig. 11. Test r2 scores for the test dataset on the trained model. Triangles: values for the 30x30, 3 class example landscape. Stars: values for 8 out of the 9 corine cutouts.

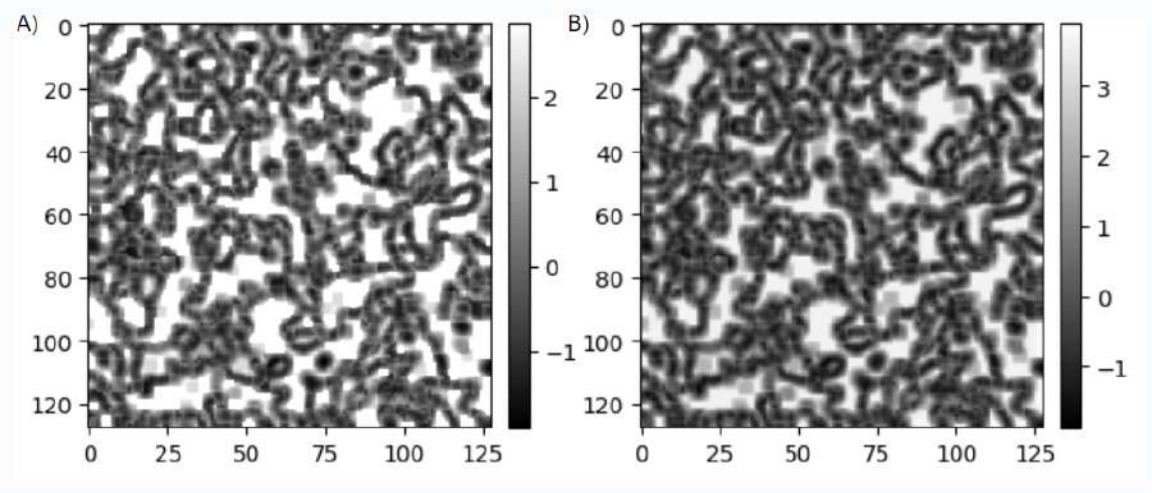

Fig. 12. Comparison of the normalized target contag metric (A) and the respective prediction (B).

The predictions look quite similar to the original LSMs, where some values tend to appear more extreme, in this case the white areas in the original (Fig. 12A) are NaN values, while in the prediction those are some of the larger values (Fig. 12B). The r2 score, pixels with NaNs excluded, is 0.9808 in this case.

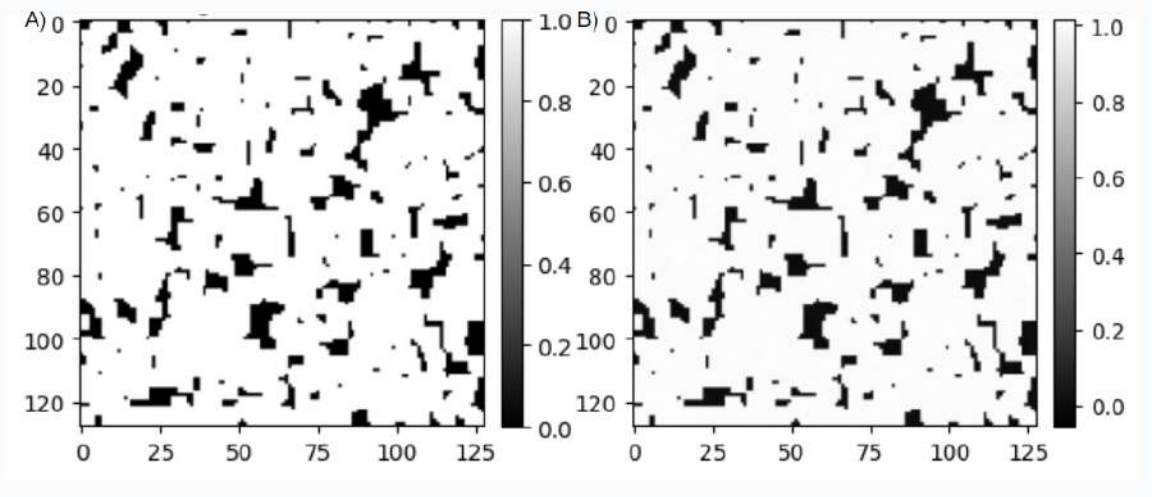

Fig. 13. The original NaN mask (A) and the predicted one (B)..

Pixels colored black (equaling a value of zero) depict the NaN values. The corresponding predicted NaN mask looks almost identical, however noticeably some values are slightly above 1.0 and below 0.0 (Fig. 13).

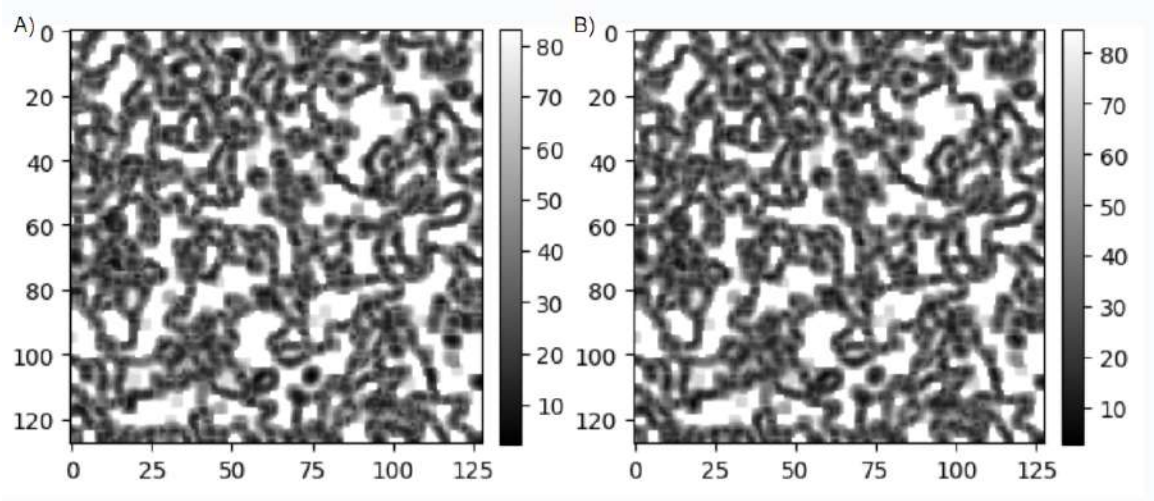

Fig. 14. The original target contag metric (A) next to the denormalized metric with NaNs re-added (B). NaNs are white in both cases.

The denormalized contag metric with NaNs added (Fig. 14B) looks almost identical to the original target metric (Fig. 14A). Some values are slightly higher or slightly lower. For completeness sake, the r2 score between target and prediction is the same as in the normalized case, 0.9808.

Tab. 5. Table comparing metric prediction times

Metric	**Conventional generation time per 56 metrics**		**New time per 56 metrics**
	R32	HPC	Jupyter docker
1 * 128x128	4.5min	<4min	19ms ± 10ms (sd)
100k * 128x128	>300 days	>20 days	35min

Training the model on the training dataset took ±35 minutes per epoch. In total this equaled to 55 hours and more. This comes with the added cost of having generated the training landscapes and metrics over weeks to prepare for training. Meanwhile, predicting the metrics after the model has been trained only takes in the realm of milliseconds for one, or in the order of 35 minutes for 100 000 predictions (Tab. 5).

3.1.2 Results smaller and bigger landscapes

For the example landscape, also all 57 metrics were computed, including the NaN channel. The results shown here were produced when padding with reflect as described above. Visually they appear very similar to the targets in all cases. Numerically, some values fit well, according to the r2 score, i.e. for pladj (Fig. 15A/B) with an r2 score of 0.98. Some other metric values however are very far off the target values, such as area_sd (Fig. 15C, D), where the r2 score and the predicted values are very far off, evidently. Compare fig. 11 for all positive r2 values.

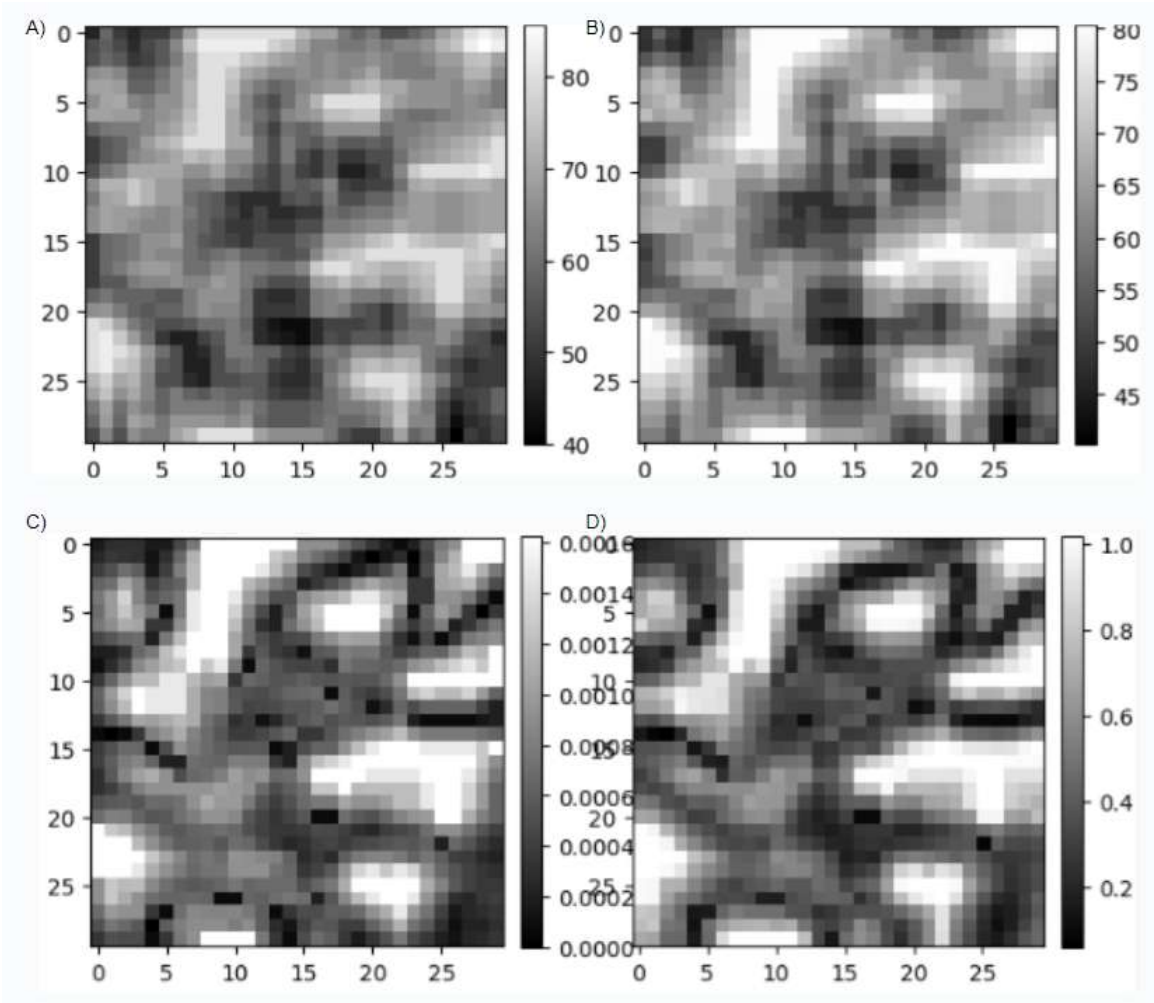

Fig. 15. The original target pladj metric for the example landscape (A) and the predicted one with an r2 of 0.98 (B). Example landscape target area sd original (C) with prediction with R2 score (- 670 million) after denormalization (D)

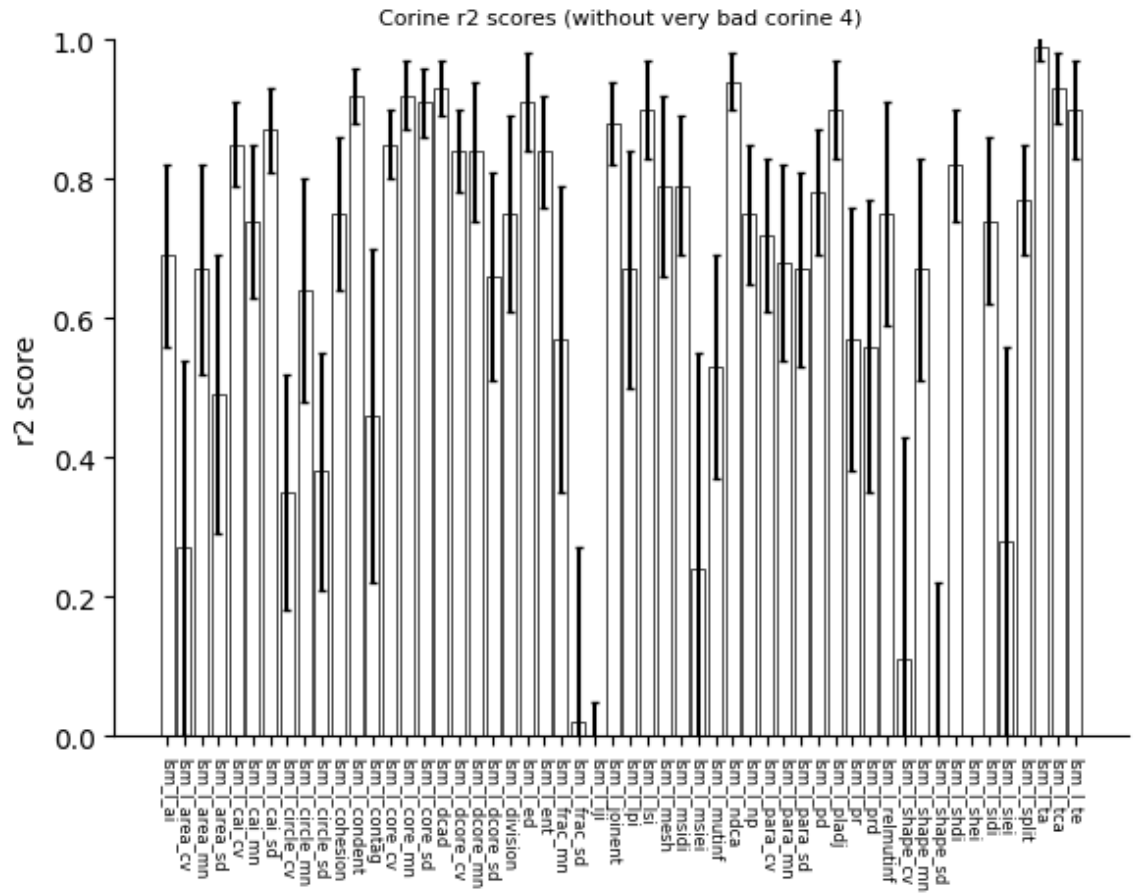

Fig. 16. Corine r2 metrics for 8 out of the nine cut out pieces, excluding the center piece at index 4, which returned negative values almost exclusively. Error bars are standard deviation.

Regarding the corine dataset, comparing the four different methods showcased, the ones working with the normalized landscapes and normalized trained model, the ones working with the unmodified original NLMs and the one with the noise delivered worse r2 scores compared to the renumbered one. The r2 scores for predicted corine LSMs by the lastly mentioned method are generally in the range between 0.6 and 0.9, with some values far worse and few slightly higher. Similarly to the other cases, the standard deviation for the higher values tends to be lower, for the lower values it is higher (Fig. 16). Exemplary for one of the worse results, frac_sd values after denormalization are quite far off, even though the predicted LSM does look similar to the target. For pladj on the other hand the predicted LSM looks very close to the target (Fig. 17).

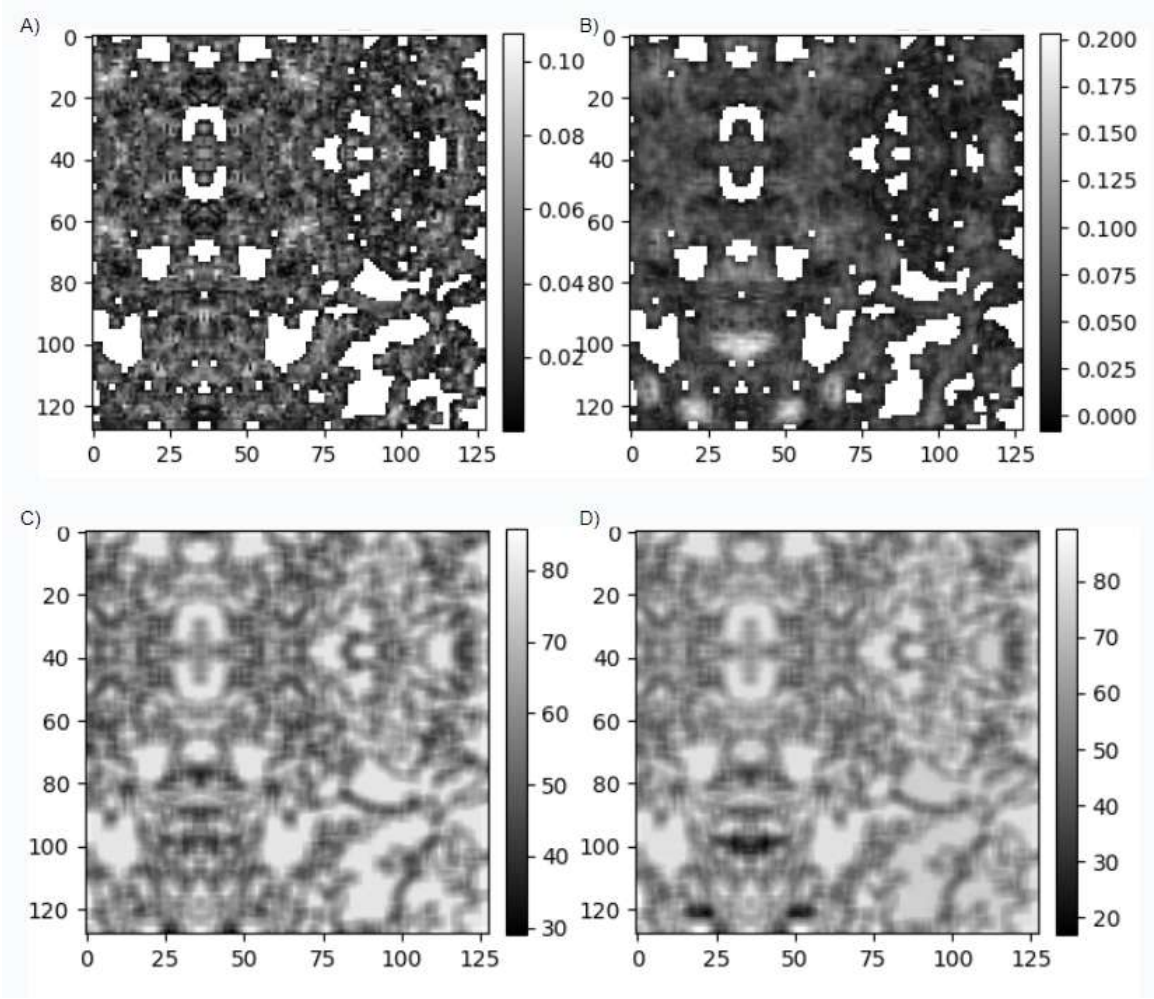

Fig. 17. Corine 0 target frac_sd (A) and denormalized prediction (B). Pladj target (C) and prediction (D)

3.2 Results landscape optimization

Different achieved loss scores for the inverse landscape optimization for different methods show that while the metrics predicted from original NLMs from the testing dataset, when input into the model for testing purposes, achieve HL rates of between 0.01 and 0.02. Conventionally optimizing the landscapes leads to HL of values of below 0.08 for some of the landscapes, but generally higher values. LPbP leads to loss values as low as below 0.04. Of the various discretization variants experimented with, the depicted int constraints every 100 epochs leads to values similar to the conventional approach, sometimes lower. Some outliers have very low values (Tab. 6.)

Tab. 6. Huber Loss of metrics calculated from the specified landscapes. Conventional optimization are continuous values, LPbP and int constraints are discrete NLMs

Landscape	Original landscape	Conventional optimization	LPbP (10 rounds with 20 epochs conv in between)	Int constraints
Test ls 95001	0.0136	0.0888	0.0581	0.075
Test ls 95002	0.0181	0.0957	0.0793	0.1113
Test ls 95003	0.0148	0.1077	0.0821	0.0656
Test ls 95004	0.0103	0.0776	0.0397	0.0867
Test ls 95005	0.016	0.1058	0.0785	0.1063
Test ls 95006	0.0132	0.0866	0.0534	0.0988
Test ls 95007	0.0104	0.0775	0.0356	0.0765
Test ls 95008	0.0148	0.0895	0.0708	0.1114
Test ls 95009	0.0169	0.0844	0.0753	0.0612
Test ls 95010	0.0105	0.0858	0.0497	-

For the original landscape, the loss rate got worse to an HL of 0.04 in one case. The landscape changes away from how it originally was. For the same landscape, optimizing it conventionally led to a HL of around 0.09.

3.2.1 Conventional optimization

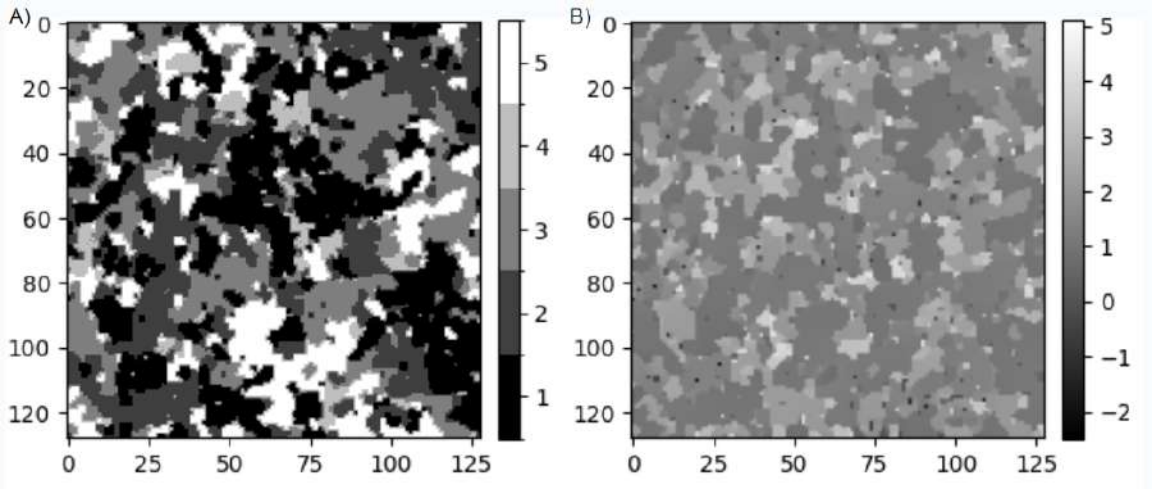

Fig. 18. Depicted here is target landscape 10, consisting of fewer, larger patches (A), next to the conventionally predicted landscape consisting of continuous values (B).

In the conventional inversal approach using the main model trained with zero padding, individual patches do emerge, however these are not attributed correctly to the correct classes: non-touching patches that belong to different classes in the original NLM may have very similar values, and those that originally belong to the same class may have very different values. Furthermore, they consist of continuous instead of the desired integer class values. These are exceeding the values for the highest target class values, and negative in some cases (Fig. 18).

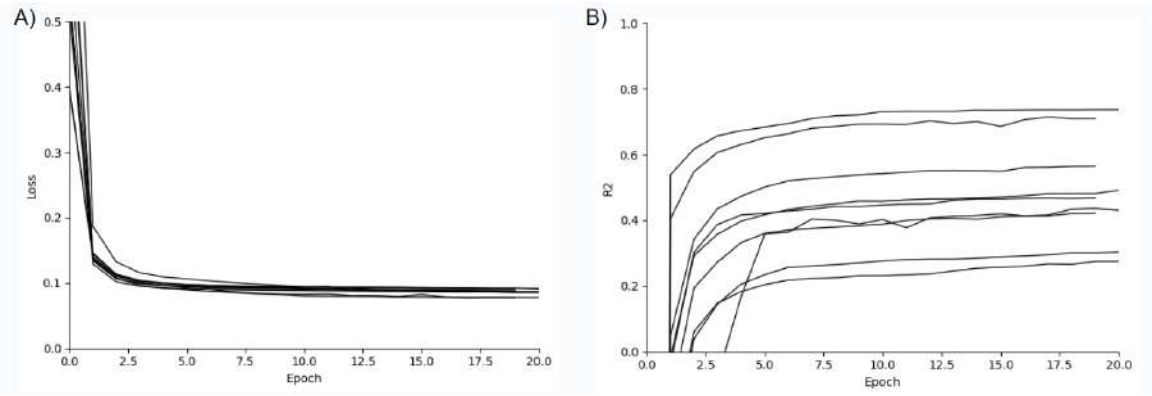

Fig. 19. Huber training loss (A) and r2 scores (B) for the conventional optimization approach, epochs in 100 steps.

Zooming in into the training loop loss and r2 curves shows that the loss is converging around 0.1 while the average r2 scores vary to a larger extent between 0.2 and 0.8 (Fig. 19). Looking at the r2 scores by individual metrics shows that certain metrics that are calculated from these optimizations achieve higher values, while the standard deviation between the results of the different landscapes is large (Fig. 20).

Regarding training with the one hot trained model, the results lead to binary landscape outputs, however many of those patches were overlapping.

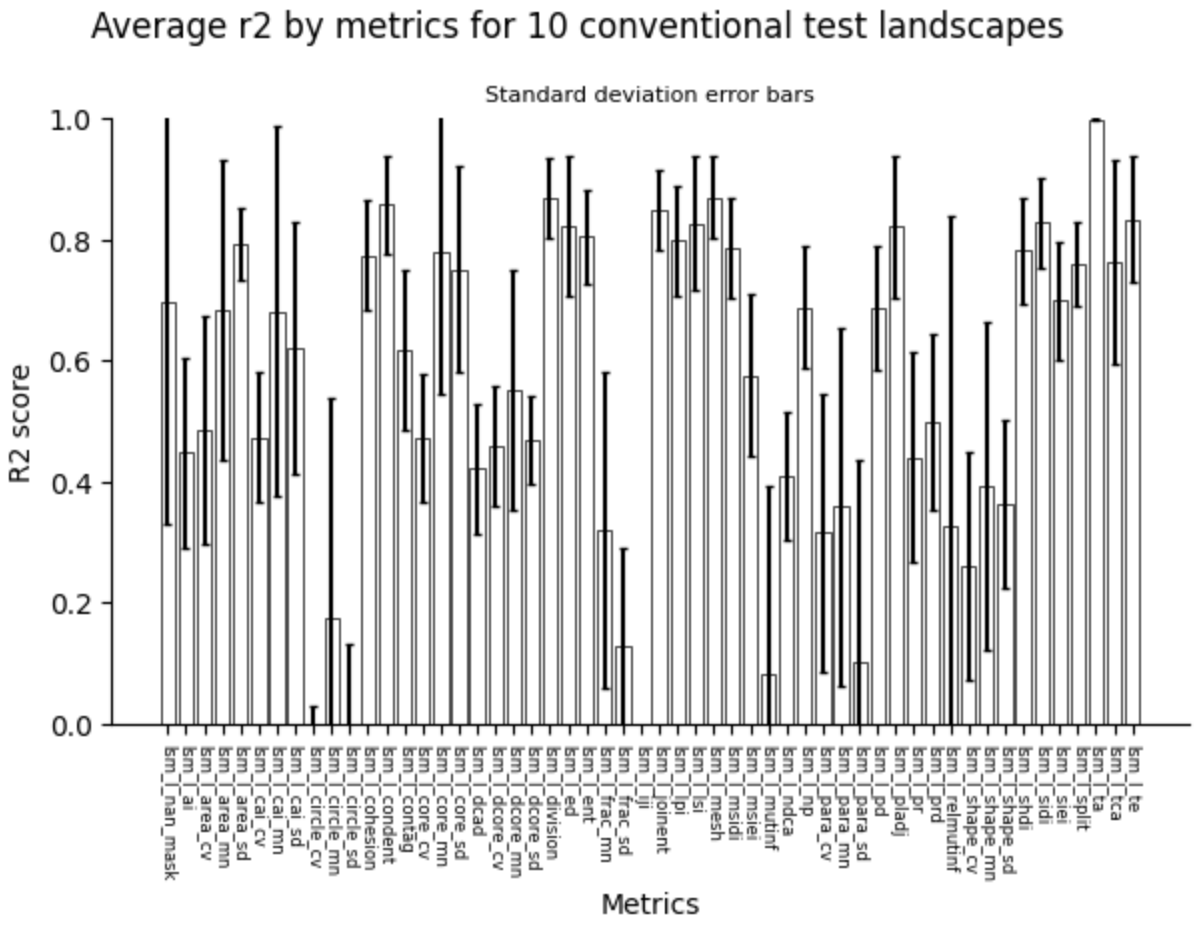

Fig. 20. Depiction of r2 values for ten All 10 conventional optimized from continuous values. Note that ta (total area) will always be equal to moving window size, can therefore be disregarded.

3.2.2 Digitizing the optimized landscape

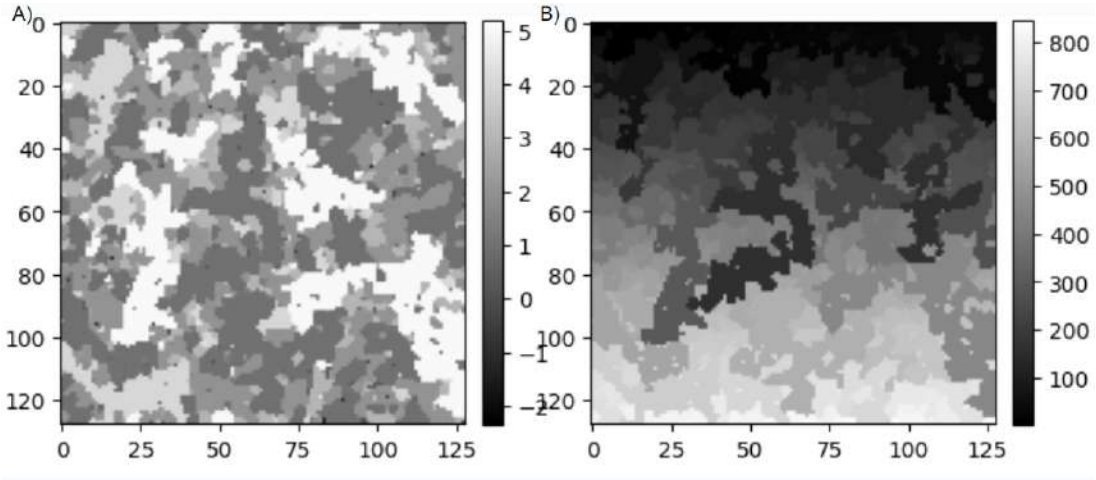

Fig. 21. Applying the CCL to the conventionally optimized NLM (A) creates a CCL-landscape (B).

The CCL algorithm identifies the different patches based on the conventionally, inversely optimized landscape (COL) and labels each of them with a different value (Fig. 21). As described above, the patches are then labeled one by one, starting with the biggest one (Fig. 22)

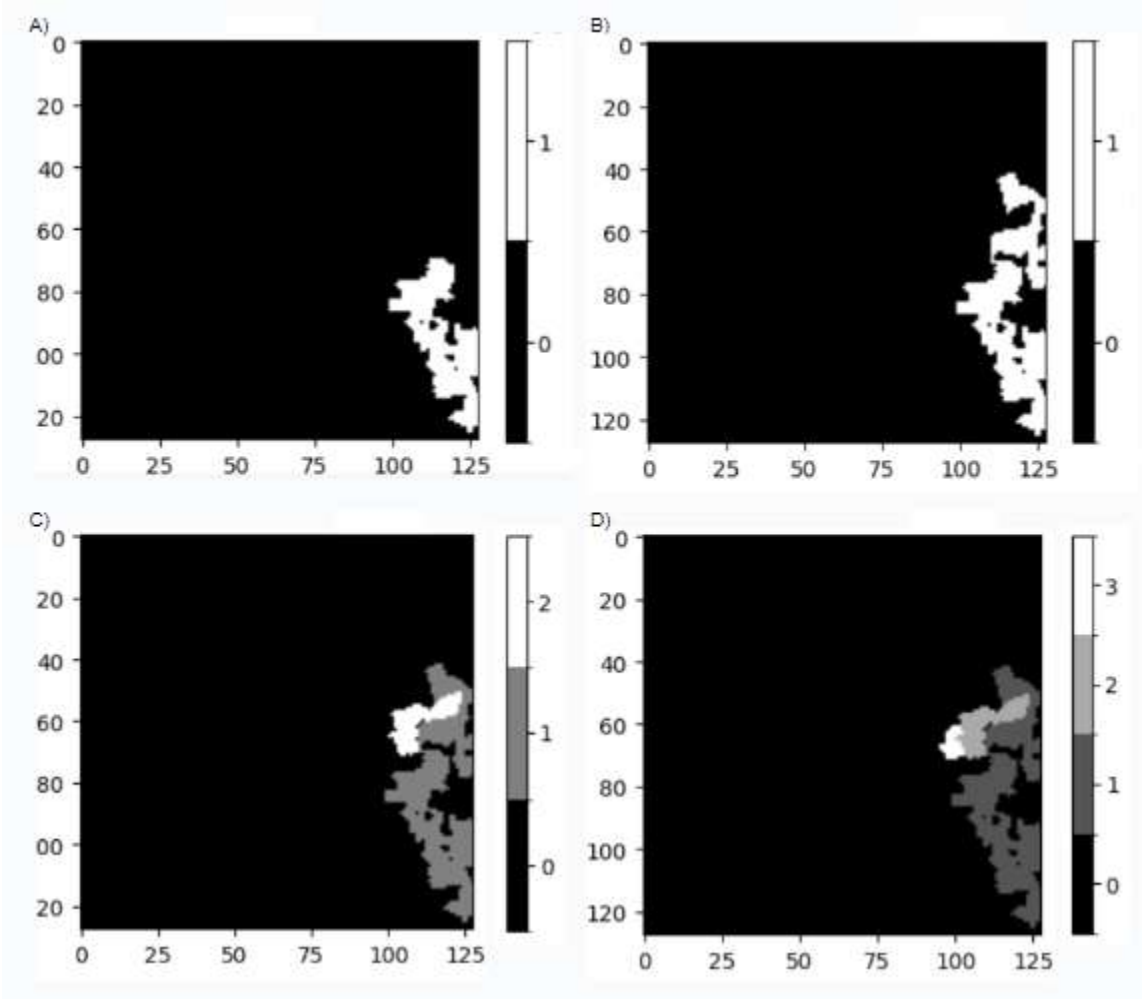

Fig. 22. Exemplary first four steps of LPbP for test ls 10.

Depending on how many patches there are, it can take 5 hours to do 10 LPbP, or also 12 hours or more (without skipping any). Exemplary results when working with such a CCL-NLM to do largest patch by patch labeling (LPbP) can be found in figure 21. Some patches do have a very similar shape to the targets. Some of them are labeled with the same class label, others meanwhile are assigned a different class. Consequently, classes are not simply shuffled around (i.e. 3 is labeled to 1 and so on) moreso they are mixed up,

Some patches are bigger than they should be, some are smaller. Some are merged that shouldn't be (Fig. 23).

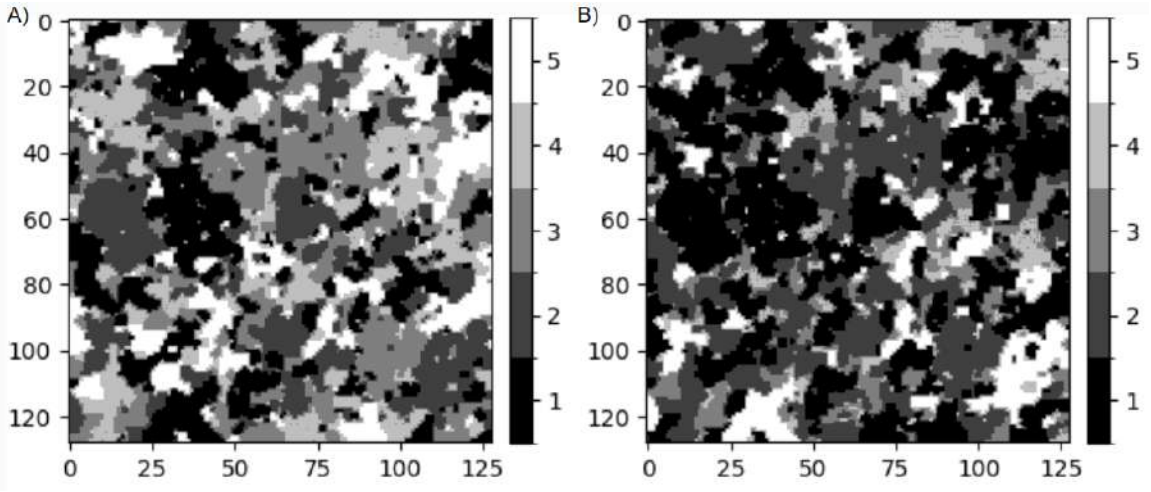

Fig. 23. Target test dataset NLM 7 (A). NLM 7 after 10 LPbP loops (B).

The huber loss score for the continuous landscapes, as they are optimized as part of the LPbP-loop, are initially far lower than those for the discrete landscape. However, after multiple full iterations, these numbers converge at a low loss rate. During the first LPbP run the losses were still high, while every following iteration they are decreasing. It can be seen that in LPbP, the loss rates go down during conventional optimization and then slowly get worse again for discrete patch by patch. The sharp lines signify the beginning of a new iteration, including new CCL, which produces lower results. Some landscapes, generally those with fewer, larger patches, reach lower loss scores than others using this approach. This also reflects in the R2 scores of the calculated metrics. The overall R2 score from the metrics calculated from the optimized landscape improved from 0.47 to 0.81, reaching individual metric r2 values greater than 0.95. Landscapes with fewer larger patches improved to Huber loss values below 0.04, while those with more smaller patches barely dipped below 0.1 (Fig. 24).

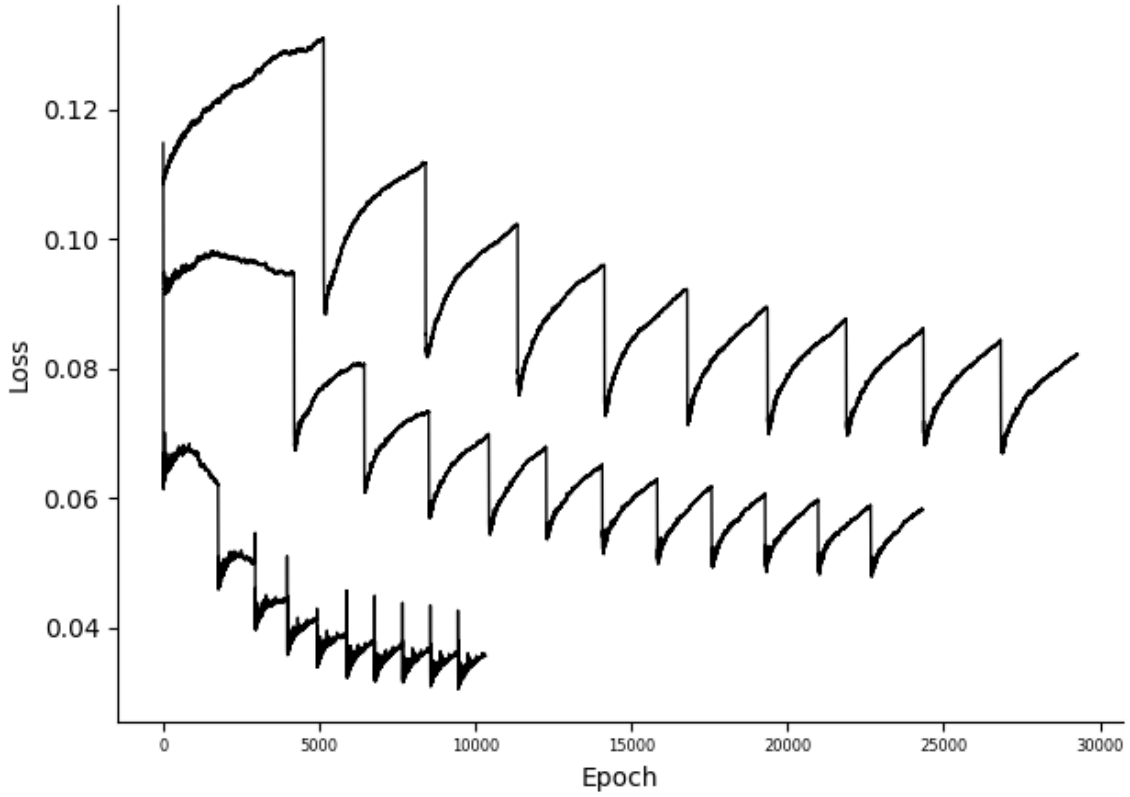

Fig. 24. Exemplary Huber losses for test NLM 1 (middle), 3 (top, many small patches) and 7 (bottom, few larger patches) for LPbP. Compare tab. 6.

Looking into the predicted metrics from the LPbP optimization approach, they again look quite close to the targets. Some, like area_mn, also has a high r2 score of 0.97 in the case of NLM 10 (Fig. 25 A,B), while others, like circle_cv, despite showcasing similar looking scale bars and general layout (if ignoring the white NaN values which are masked in evaluation), only achieve r2 values of 0.57 and below (Fig. 25 C,D).

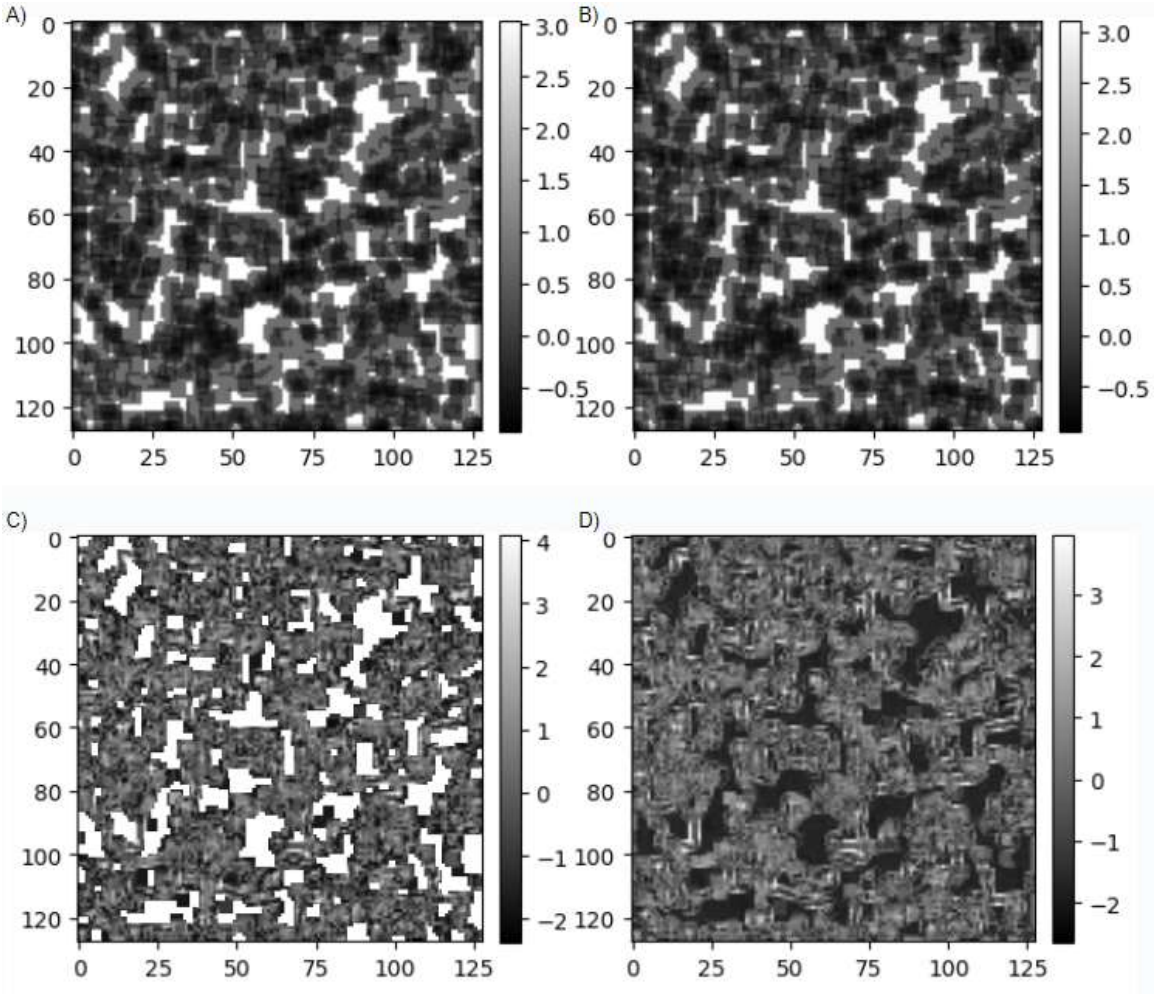

Fig. 25. Landscape 10 area mn target metric (A) and LPbP predicted metric (B). Normalized values on the bar. Target circle cv LSM as an example for worse performance (C). White are the NaN values, which have not yet been replaced in the predicted metric (D) in this case.

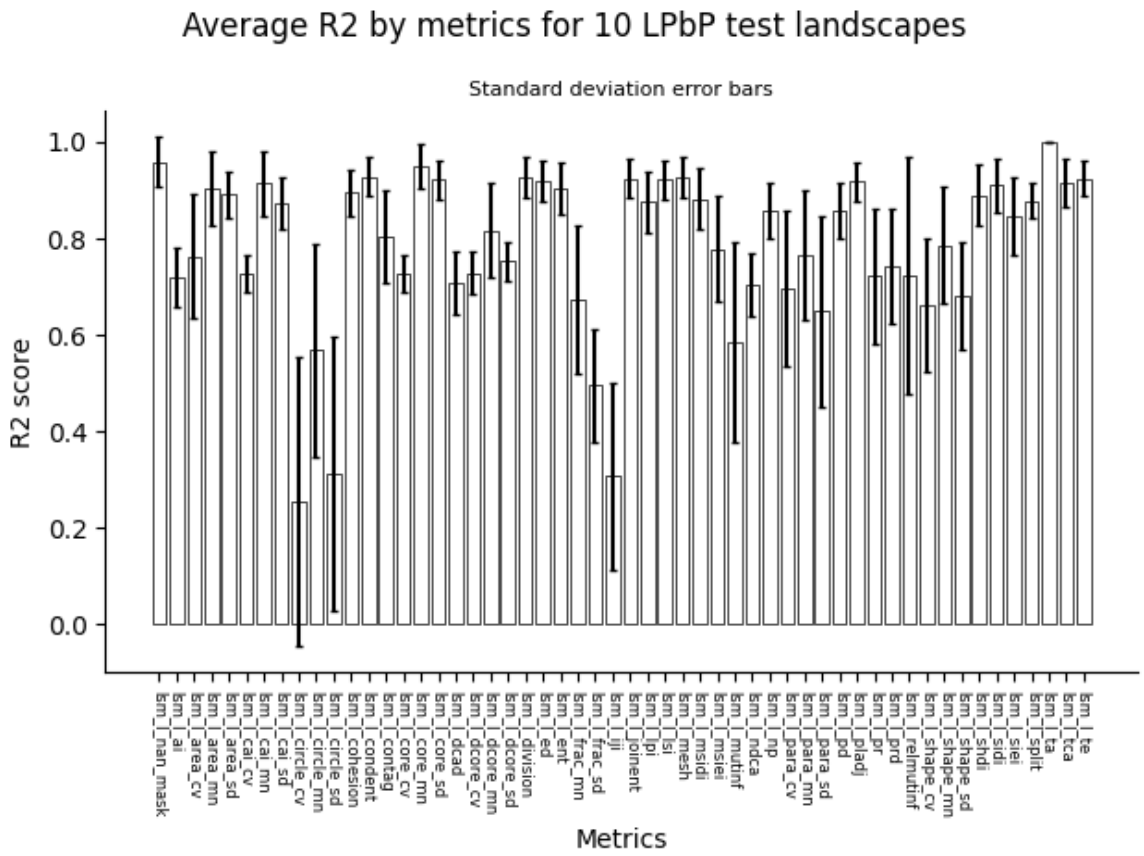

Fig. 26. The average r2 score for the individual metrics calculated from the ten exemplary test dataset LPbP - optimized landscapes.

Some of the individual r2 values for the LSMs predicted from the LPbP-optimized landscapes are nearing the ones of the original landscapes. Mostly those shape type metrics which achieve lower values even when working with the original landscapes (compare Fig. 11) also display low average values even below 0.5 at times (Fig. 26). Comparing with the r2 values that are calculated when calculating LSMs from these landscapes landscapemetrics in R returns a very similar plot. Some values, ie. the circle ones, are slightly higher compared with those from the model-predicted metrics (Fig. 27).

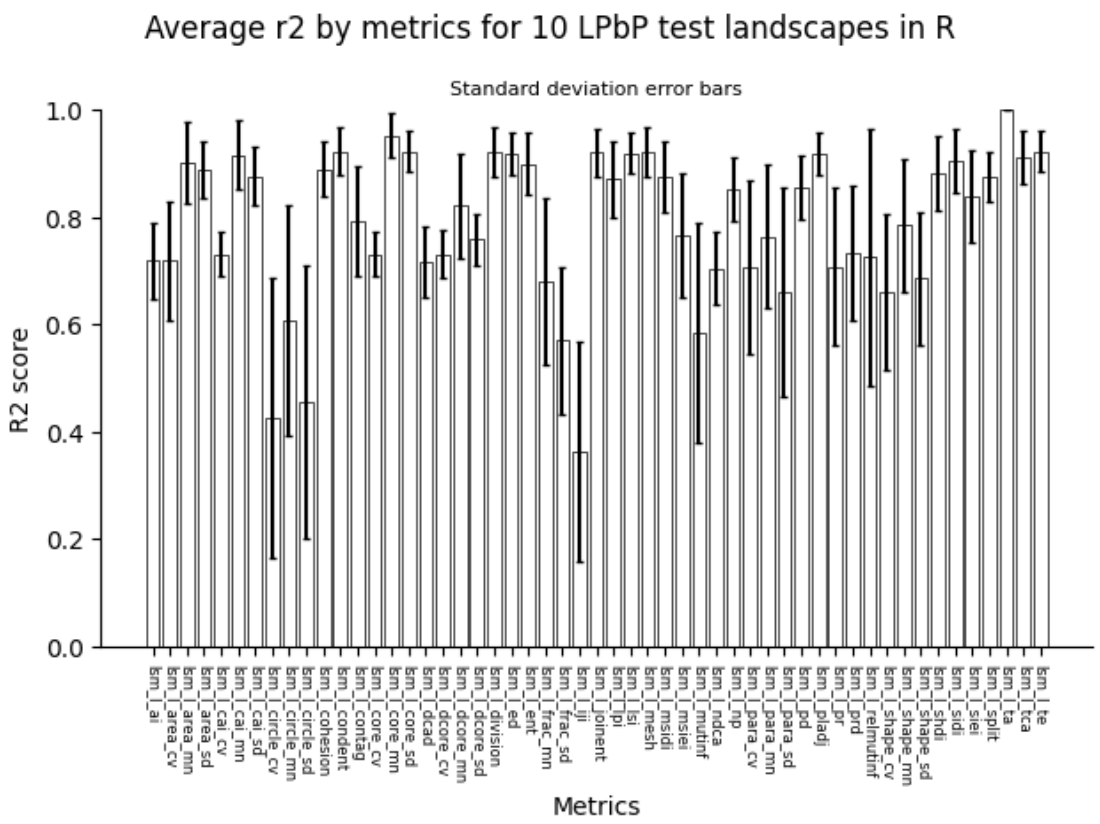

Fig. 27. R2 values for metrics generated in R based on LPbP-optimized NLMs. Note that the NaN mask in position zero is not depicted.

3.2.3 Results Smaller and bigger landscapes optimization

When padding and normalizing the target metrics for generation of the example landscape, the padded optimized landscape displays slightly different elements for each of the different parts of the array (Fig. 28B), even though they should be identical (Fig. 28A). When zooming in with some imagination some of the original patches can be identified (Fig. 28 C, D).

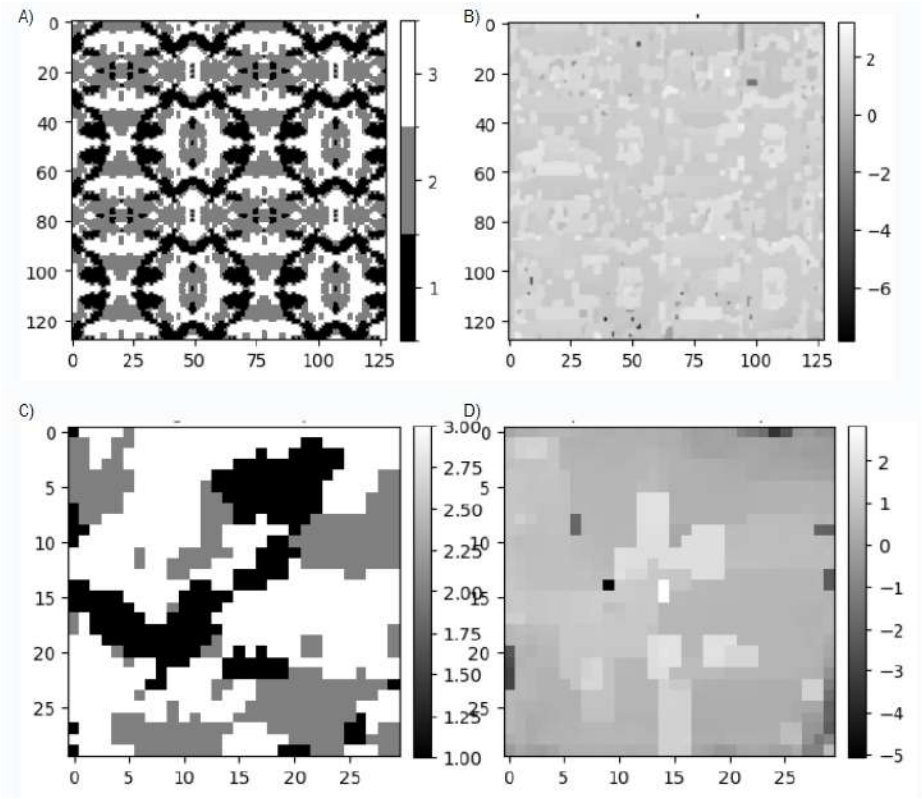

Fig. 28. Reflect padded example landscape (A). Conventionally optimized example landscape (B). When zooming in on the original example landscape (C) and one of the corresponding conventionally optimized areas (D).

When working further with LPbP to predict the example landscape, the results look very different from the target, with one huge patch (black, class label 1) covering the full area (Fig. 29).

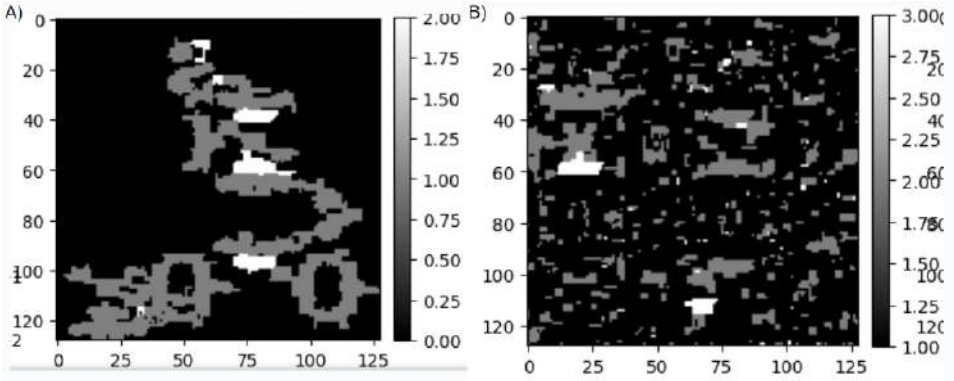

Fig. 29. Largest patch by patch for largest patch during the process (A). The result after multiple iterations (B).

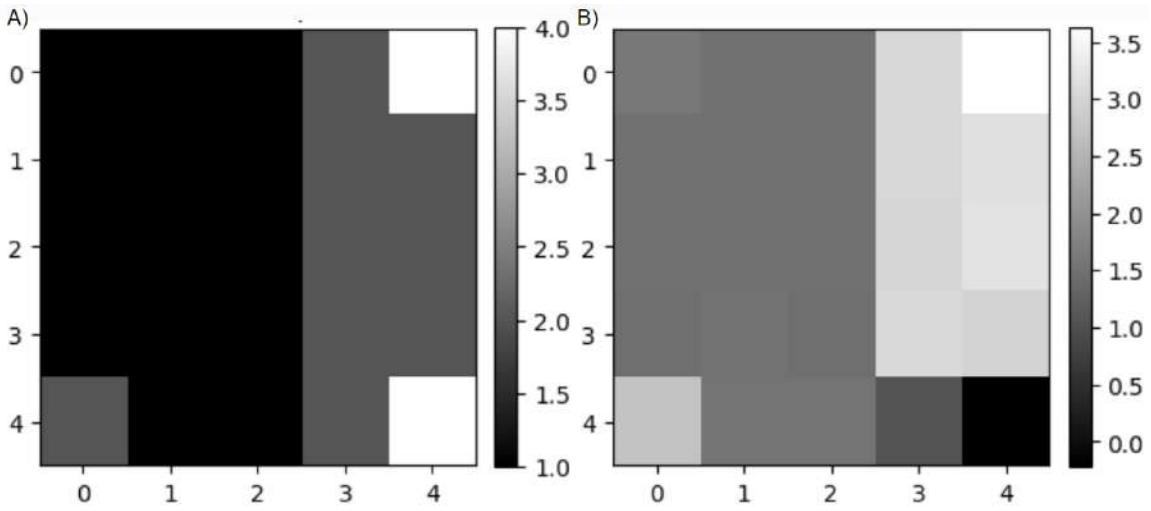

Fig. 30. In the 5x5 windows, patches of the same class are misidentified by the conventional optimization approach. I.e. the two patches of class 4 top and bottom right in the target landscape (A) are identified as patches belonging to class 3.5 and 0 respeciveley in the optimized landscape (B).

Patches that are originally of the same class are identified by the model as separate (Fig. 30).

4 Discussion

4.1 Discussion Metric Generation

4.1.1 Discussion training

Early attempts showed that it would be possible to predict individual metrics quite reliably. Further testing confirmed that it would be possible to train a model for multiple LSMs when stacked together in one single target tensor. This would enable the generation of multiple LSMs for one NLM simultaneously with one GPU, contrary to using multiple CPU cores in the past for such a task.

The results for the training with the different settings normalizing input landscapes, different padding versions, adding noise and working with binary class layers show that all of them return agreeable results with r2 scores after 50 epochs above 90% (Tab. 4). Adding noise performs worse than the other ones, which appears easily explained by added complexity. The binary one hot delivers the best results at that point, indicating that this may be a more solid variant to work with - assuming that the problem of the amount of classes equaling the amounts of binary layers can be solved.

The loss and evaluation curves when plotted show the well known asymptotic behavior as training is advancing, closing in on zero (HL) and one (r2 score) respectively (Fig. 10A/B), albeit never reaching it. This, combined with the observation that evaluation and testing dataset each produce very similar scores, suggest that no overfitting does occur (Ying, 2019). Another explanation may be that test and evaluation/training metrics where too similar and overfitting did occur, which may explain difficulties for working with the example metric and the corine data which originate elsewhere. Adjusting the learning rate further improves the scores slightly, as has been shown to generally be the case (Smith, 2017). The heuristic nature of Deep learning may be naturally limiting even better scores. Another reason why there may be lackluster results may be the described padding problem, where due to the need to fill in border values to retain the original dimensions, zeros are filled into the convolutional layer kernel, instead of NaN values as is the case in

previous approaches, for example in the focal function of the terra package (Robert J. Hijmans & Roger Bivand, 2023) which is used in the landscapemetrics R package (Hesselbarth et al., 2019).

Trained models for 56 different LSMs, plus the binary NaN mask, are highly accurate in predicting most of the target metrics, and fairly good at the remaining ones (Fig. 11). R2 scores above 0.9, often even higher at 0.98+ imply strong correlation between target metrics and predicted metrics clearly show the strong inference capabilities of these trained models. This is further underlined by the fact that numerous studies have been undertaken, comparing LSMs by correlation to identify which of these are obsoleted by the existence of very similar ones - oftentimes far lower r2 scores are sufficient to declare a metric redundant (Cushman et al., 2008; Hargis et al., 1998). In this context it may be advisable, if these predicted metrics are used, to clarify that they are mere approximations of the original target metrics but shall suffice in presenting the sought for information the original metrics would entail.

The NaN layer is generally predicted very accurately, usually delivering pixel-perfect results when rounding to zero and one respectively (compare pre-rounding Fig. 12). This high accuracy can be easily explained by the fact that those are areas where only one or two classes appear in one 5x5 window in the original landscape. Therefore it could also easily be computed by conventional means, however simply adding it to the tensor reduces the extra step of having to calculate it separately.

Even in cases such as in the example contagion metric where some of the resulting values appear too high, these are in that case simply filling in where NaNs are supposed to be (Fig. 14). Denormalizing and inputting the NaN values presents a suitable resolution, delivering a LSM, closely resembling the target visually as well as by r2 score. This high similarity holds true for many of those LSMs where NaNs are present, as well as for many of those where none are.

Other metrics however proved to be problematic: Most of the worse results are of the shape-metric type, even though some of them, especially the mean variants shape_mn, para_mn or frac_mn return reasonable r2 scores. Why exactly is an open questions, possibly related to the shape of patches being difficult to describe. Therefore, when needing to work with shape metrics, it is advisable to work with those rather than the standard deviation (sd) or coefficient of variation (cv) versions.

The parallelization alone increases computation speed dramatically, combined with computing individually faster also huge increases in speed are produced. In taking a few minutes to generate thousands of NLMs for their respective LSMs, it is much faster than conventional LSM generation (Tab. 5), utilizing only a single high performance GPU even when compared to a HPC cluster of hundreds of cores. Therefore it is plausible to use this approach to predict LSMs from NLMs, and do so by a large factor faster than conventionally possible. As an aside, it is now possible to compress the size of hundreds of GB worth of metrics to a simple model of 80MB, plus 500MB or corresponding landscapes, albeit with relevant loss in information. Using neural networks for data compression has been considered and done before (Goyal et al., 2018; Ma et al., 2020). This Model could now in theory be used to generate more training metrics from input landscapes. While there exist a plethora of works applying deep learning methods in the field of landscape ecology generally (Brodrick et al., 2019) or landscape metrics in particular (Gevaert & Belgiu, 2022; Z. Li, Zhang, et al., 2022; van Duynhoven & Dragićević, 2023), the author is not at this time aware of any similar approach of predicting the landscape metrics themselves.

4.1.2 Discussion bigger and smaller landscapes

The LSMR example landscape with 3 classes of 30x30 pixel dimensions was predicted very well for some metrics (Fig. 11), showcasing that simple reflect-padding and then cutting again is a useful workaround for inputting a smaller landscape into the model. However some of the metric values are very far off, which is possibly explained by either mixups in the denormalization process, or may also be caused by the fact that the normalization values for the 5 class 128x128 landscapes differ significantly from those

which would be required to properly denormalize the example landscape values. As a consequence of these findings, basing the normalization mean and standard deviation on those calculated from the generated dataset may be problematic for NLMs with fundamentally different LSMs.

Regarding the corine example landscape, splitting the landscape by cutting it up and then concatenating the metrics after they have been predicted enables the generation of arbitrarily big metrics with ease. Consider, for example, the full dataset of 100k landscapes - a total size of more than 1.6 billion cells - for which 56 metrics can now be calculated simultaneously in the span of half an hour, compared to weeks, as was the case before (Tab. 5).

However it appeared more problematic to predict for NLMs with more classes. On top of the aforementioned normalization issue, the existence of class values outside of those the model was trained with may inhibit the model's capability to predict satisfying metrics. Normalizing or renumbering the input example corine landscape yields much better values. This clearly suggests that the model is not well prepared to predict metrics from landscapes that consist of class values outside of the 5 it was trained with. During training the classes of the training landscapes could be randomly shuffled to larger values, i.e. two-digits, to prepare the model for such values. Combining this with normalization could deliver better results. Another workaround may also be to customize the convolutional layer class in such a way that it renumbers the input classes for every 5x5 window to the lowest possible, as among those 25 classes there'll rarely be a larger amount of classes present while which class carries which label doesn't matter and is disconnected from patches outside of this window.

4.2 Discussion landscape optimization

4.2.1 Conventionally

The approach to optimize the input landscape inversely from random noise to fit the desired target output leads to an array, which for some metrics that the model has been

trained quite well on, led to acceptable metric predictions. Generally it seems as though the more fragmented with larger numbers of small patches the landscape is, the more difficult it is to predict accurately. This explains the high difference in R2 values (Fig. 19B). The patches of that landscape consist of continuous values while the target landscapes are to have discrete values (Fig. 18). However, it is still possible to find similar landscapes with different distribution of classes but the same shape of patches. This hints at the model weights allowing for non-discrete values to also predict quite nicely.

Not only is the problem that patches with close values may be split up, the extreme values may also be of the same class. Importantly, it cannot discern whether two distant patches belong to the same class (Fig. 30). One reason could possibly be that long distance information is not observable with a window size of 5. For example edge and core metrics only care whether patches on their vicinity are different, but not about patches further away. One idea to deal with this would be using a bigger window size which rapidly runs into limitations when working with U-Net. Alternatively, an attention mechanism - which is known to deal well with long-distance information - or other architectures could be employed for the patch generation (Vaswani et al., 2017). Also, for some metrics the configuration of the 5x5 window is not as important as simply the area occupied by each patch.

It’s clear that just optimizing the landscape like this returns a non-desired NLM. There is some degeneracy, in the neuroscientific term of the word (Edelman & Tononi, 2001), allowing for different NLM configurations to return very similar if not the same LSMs. This poses a difficult obstacle to the goal of predicting the exact same landscape from which the metrics were originally computed from. On the other hand, for the use case of comparing landscapes for one metric, the generation of different landscapes may be a non-issue.

The model calculates values for continuous landscapes. It is indifferent to whether the values are integer or float, it appears to instead care about whether they are different or

similar. Previous software on the other hand, such as landscapemetrics in R, cannot work with continuous landscapes. The question is of course how valuable these values are, but it can be assumed that they are equivalent in meaning to the ones calculated from discrete landscapes. This becomes interesting as it is a necessary inconvenience to first translate a continuous landscape, i.e. from remote sensing imagery, into a neutral landscape model of continuous classes (Hesselbarth et al., 2019). Possibly skipping the discretization step and directly predicting those metrics - which are not even defined for continuous landscapes - is an approach worth pursuing by the means described here. For now however, when looking at the evaluation scores for the metrics predicted from the continuous landscapes, while there are some good values, in general they may be too low to work with. Especially the aforementioned problematic shape metrics fall short (Fig. 20).

Generally it turns out that NLMs with bigger patches are easier to predict. Regarding alternative trained models, the noise one led to disappointing results in the limited testing that it underwent and was therefore not pursued further. It may however be one of the many ways worth pursuing, in the case of dealing with larger class values and for working with normalization and ultimately to predict better NLMs. The proposed training with normalized landscapes to facilitate prediction of arbitrary class values may in turn inhibit the inverse prediction of discrete landscapes, short of discretizing post-processing operations.

Working with different fixed numbers 1 to 5 - each of them being equivalent in value - may pose a problem. One possible solution is the one hot approach, however working with higher channel numbers would then need to be worked with from the get-go. Experiments with one hot encoding in this study showed that patches would overlap when trying to predict such landscapes. The literature for inverse problem solving suggests that more satisfying results should theoretically be possible (Aggarwal et al., 2019; Lucas et al., 2018).

4.2.2 Discussion Digitizing optimized landscape

Various approaches to threshold values were tried, either calculating them according to previously attained percentages of each class, or by rounding and clamping based on the predicted values - or instead of clamping assigning intermediate values to every value outside of the range, or even optimizing only those values outside the range after setting them to random values inside of the range. Or even “minimize” these thresholds either by hand (i.e. guessing the values) or to use an algorithm to return thresholds which appear to return the lowest loss score for every given scenario, or round combined with clamp. Or fill fully one class first, and then add the other ones on top of this - but here again the difficulty lies in not entity solving the question of which patches belong to the same class based on the COL. During optimization the landscape is discretized at every step, or every couple of steps.

The loss scores for the metrics for integer constraints are usually similar to those of the conventional optimization approach without constraints, the difference being that these values are now digitized. They are however below those of the LPbP version (Tab. 6). One problem with the digitization thresholds is that some patches that should have the same value are too far apart in their values: the regional patch values are only ever dependent on the surround values, so the COL float values will always be dependant on those only - therefore it can’t put two patches that are far apart into the same category (Fig. 30). Some extra manipulation is needed. However, even setting only those values that are outside of the desired range to other values also doesn’t lead to nice results. For the integer constraints, varying the amount of epochs in between constraints may depend on the configuration of the chosen landscape without a certain amount being the correct answer.

When running LPbP for multiple runs, some of the results when working with landscapes with many larger patches appear to get close to the desired results - at least predicting some metrics well. Landscapes with larger, fewer patches generally tended to optimize much better than those with many small ones (Fig. 24). Interpretation of the loss curve looks as follows: During conventional optimization the loss is going down, however

during the LPbP it is slowly increasing, as the new patches filled in with new values apparently lead to worse results. Running LPbP iteratively multiple times leads to increasingly better results, however a plateau is reached after < 10 repetitions (Fig. 24).

It can be observed that the patches during this method increasingly merge in subsequent runs, without much loss or even with improvement to the loss of the LSMs r2 scores (Fig. 29). This hints again that there are many possible landscape configurations leading to very similar metrics, as well as the fact that an NLM could completely be labeled with 4 classes only, as per the four color theorem. This implies that the prediction could possibly move in on that (Thomas, 1998).

Even though the LPbP results appear to be better than those of any other tested approach, returning good results for some NLMs and some LSMs (in the case of certain metrics these results really seem exceptionally good), they leave a lot to be desired (Fig. 26). Some considerations include: For CCL, it should be considered that a very low divisor for the threshold values will lead to large patches, reversely, if there are a lot of small patches, even smaller than those of the original landscape, the results will be more accurate, however it will take much longer to go through every single one of those patches. Importantly, if a (larger) patch has been assigned any value, it will mostly stick to that value when optimizing conventionally - no matter which value that is. Only if it conflicts with neighboring patch values will the value change.

A variant of LPbP is to instead digitize all neighboring patches at once. To fill the remaining area, conventional optimization, or filling with zeroes and masking it for the training. Another variant would be to ensure that every further labeled patch is adjacent to previously labeled ones, instead of moving onto the next largest (possibly non-connected) patch one after no more adjacent patches to the latest patch can be identified. Optionally skipping smaller patches in between also seems an option to speed up LPbP. Adjusting the amount of epochs run conventionally in between is a further possible modification to improve performance. Ensuring that not too many patches merge

together, while still allowing for if the threshold was set too low and they got split where they were not supposed to, is another possibility.

A more trivial approach like rounding can be used instead of connected component labeling to identify the patches. Some way to label the individual patches would then also be required though, for which purpose the CCL works quite well. For the CCL on the other hand, a dynamic threshold could be found by minimization which could reduce training time further. Using rounding instead of CCL potentially leads to the patches being merged that should be different in some cases. The step from CCL to LPbP could be smoothened, choosing a different base array to fill in. As the LPbP doesn't work well for landscapes with large amounts of patches, changing the CCL-divisor variable to alter the number of patches may be one way to improve results.

Showing the R calculated values confirms that the model is indeed working in the way of the original metric formulas (Fig. 27). This solidifies that it generally should be possible to produce fitting landscapes using this method, if just the right prerequisites are fulfilled.

As the original landscapes receive HL values of 0.02, the conventional and LPbP optimization values of 0.05-0.1 leave much to be desired. Simultaneously this offered the insight that the model is trained in such a way that it optimizes away from the target landscapes, as its weights would prefer a distinct constellation.

None of the attempted methods to predict a landscape of discrete values which also enables prediction of the same metrics from it delivered truly satisfying results. Some of the metrics from LPbP do indeed have good r2 scores, others not so much. This can be attributed to certain metrics having infinite possibly high values, while others like the shape ones circle and frac have difficulty because of the exact shape. Another detriment to the LPbP approach is the long time (hours) it takes to optimize only one 128x128 pixel wide landscape.

4.2.3 Smaller and bigger landscapes

For smaller landscapes with fewer classes it appears to work roughly analogously to the case for normal sized landscapes with 5 classes, except for the normalization issues which are in effect in both cases. These lead to very bad loss and r2 and limit the efficacy of the inverse optimization. However, as can be observed in fig. 28B, similar shapes do emerge regardless which in this case were not very useful (Fig. 28), should regardless be very well usable for further optimization. Larger landscapes, if split up into multiple smaller ones for predicting the metrics however, additionally to the known inaccuracies in predicting landscapes, showcase a boundary problem where patches supposed to belong to the same class are labeled with separate class values instead.

Looking at the example landscape padded by reflect (Fig. 28 A/B), interestingly the different areas of the landscape are predicted differently - even though they should be very similar. This possibly highlights the inconsistency of the trained model, or may be based in the inherent non-determinism. The effect that the whole landscape gets covered with mostly one class can be caused by running too many CCLs with too little divisor between patches, and subsequently the LPbP merging too much if not stopped by other means (Fig. 30). There is a lot of work to be done to improve upon these results, be it by combining multiple of the presented approaches or by further methods.

4.3 Further research

Instead of training a new full model from the beginning every time with adjusted settings or dataset, it could also be possible to predict new metrics by using the pretrained model and replacing certain channels with different metrics for maximum success by some method of transfer learning (Pan & Yang, 2010). This could be done either to add new metrics that are not currently part of the training stack, or also to input different weightings if those are desired. For either case a new training dataset will be required, possibly of a lower quantity of metrics and retrain the network on only exactly that one metric, in place where the original, unweighted metric has been trained. The inadequate metrics could be replaced with better metrics by such an approach. New metrics could exemplarily entail both versions of gyrate. Similarly, it should also be possible to adjust

the number of landscape classes when fine-tuning a pretrained network, possibly requiring less data compared to training a new model.

During Training the use of various modifications is thinkable, i.e. the use of k-fold validation. Changing the U-Net padding to reflect or extend are further options - or even to pad with NA, or change the window size from 5x5 to 3x3 in the U-Net. Further hyperparameter tuning could be done to find better parameter values (L. Li et al., 2020). Also possible is a general improvement of bottlenecks during training (Leclerc et al., 2023). Changing the data format of the model and training data to float16 may also be an option to reduce memory usage and save disk space (Gupta et al., 2015).

Going even further, utilizing model quantization approaches shows promise. This quantization is usually used in Deep Learning to reduce model size even further by working with integers instead of float weights (Wu et al., 2020). While this could in theory help solve the continuous output problem when inversely optimizing the input NLM by predicting discrete values right away, attempting to optimize that quantized input landscape may prove challenging. So far, no literature could be found attempting to solve inverse problems using quantised models.

Using a different architecture, i.e. using an End-to-End Res-U-Net approach, could facilitate inverse NLM optimization, as such an architecture has been used before to greater success than was achieved in this work for related tasks (Feng et al., 2020). Generative Adversarial Networks have also shown promise in inverse problem solving (Shah & Hegde, 2018). Certainly numerous more state of the art methods can vastly improve upon this approach.

Looking at the rapid recent advances in AI, it may soon be possible to use a multi-modal-model. When tasked to produce a certain LSM for any given NLM, it will be able to return the desired output, based on relevant information landscape ecology which the LLM scrapes from the web, combined with the image generation capability of next generation models (Wang et al., 2023).

5 Conclusion

In this thesis, it was shown that utilizing an adapted U-Net it is possible to closely predict certain window metrics for a given neutral landscape model. While perfect accuracy is hardly achieved, for some metrics the results are very good. Consequently, U-Net is capable of generating metrics based on landscapes, surpassing previous methods in competition speed at the cost of inaccuracies in the LSMs. It can possibly prove to be a useful tool to supplement previous methods.

Meanwhile, optimizing a landscape based on given metrics appears more difficult. The direct inverse problem solving approach of freezing the pretrained model and then optimizing a randomly initialized landscape yields good patch shapes, however the patch values are continuous instead of concrete and classes are connected differently than from how they should be. Various approaches to generate a discrete landscape from this conventionally optimized landscape have been tested, and while there is still room for improvement, an approach based on connected component labeling and one by one labeling of the largest patches each time leads to partially agreeable results. As outlined in the discussion, further research can be done to provide additional insight. Either by pursuing those methods suggested or entirely new approaches, as this thesis has barely scratched the surface of what appears possible to solve this problem.

References

Aggarwal, H. K., Mani, M. P., & Jacob, M. (2019). MoDL: Model-Based Deep Learning Architecture for Inverse Problems. *IEEE Transactions on Medical Imaging*, *38*(2), 394–405. https://doi.org/10.1109/TMI.2018.2865356

Aman Arora. (2020a). *SIIM-ACR PNEUMOTHORAX SEGMENTATION*. GitHub. https://github.com/amaarora/amaarora.github.io/blob/master/nbs/Training.ipynb

Aman Arora. (2020b, August 30). *U-Net A PyTorch Implementation in 60 lines of Code*. https://amaarora.github.io/posts/2020-09-13-unet.html

Angelopoulos, A. N., Kohli, A. P., Bates, S., Jordan, M., Malik, J., Alshaabi, T., Upadhyayula, S., & Romano, Y. (2022). Image-to-Image Regression with Distribution-Free Uncertainty Quantification and Applications in Imaging. *Proceedings of the 39th International Conference on Machine Learning*, 717–730. https://proceedings.mlr.press/v162/angelopoulos22a.html

Brodrick, P. G., Davies, A. B., & Asner, G. P. (2019). Uncovering Ecological Patterns with Convolutional Neural Networks. *Trends in Ecology & Evolution*, *34*(8), 734–745. https://doi.org/10.1016/j.tree.2019.03.006

Chen, B., & Iannone III, B. V. (2020). FRAGSTATS: A Free Tool for Quantifying and Evaluating Spatial Patterns. *EDIS*, *2020*(6), 9. https://doi.org/10.32473/edis-fr431-2020

Cushman, S. A., McGarigal, K., & Neel, M. C. (2008). Parsimony in landscape metrics: Strength, universality, and consistency. *Ecological Indicators*, *8*(5), 691–703. https://doi.org/10.1016/j.ecolind.2007.12.002

Edelman, G. M., & Tononi, G. (2001). *Consciousness: How Matter Becomes Imagination*. Penguin.

Etherington, T. R., Holland, E. P., & O'Sullivan, D. (2015). NLMpy: A python software package for the creation of neutral landscape models within a general numerical framework. *Methods in Ecology and Evolution*, *6*(2), 164–168. https://doi.org/10.1111/2041-210X.12308

European Union, & European Environment Agency (EEA). (2018). *CLC 2018—Copernicus Land Monitoring Service* [Land item]. https://land.copernicus.eu/pan-european/corine-land-cover/clc2018

Feng, J., Deng, J., Li, Z., Sun, Z., Dou, H., & Jia, K. (2020). End-to-end Res-Unet based reconstruction algorithm for photoacoustic imaging. *Biomedical Optics Express*, *11*(9), 5321–5340. https://doi.org/10.1364/BOE.396598

Freudenberg, M., Nölke, N., Agostini, A., Urban, K., Wörgötter, F., & Kleinn, C. (2019). Large Scale Palm Tree Detection in High Resolution Satellite Images Using U-Net. *Remote Sensing*, *11*(3), Article 3. https://doi.org/10.3390/rs11030312

Gardner, R. H., Milne, B. T., Turnei, M. G., & O'Neill, R. V. (1987). Neutral models for the analysis of broad-scale landscape pattern. *Landscape Ecology*, *1*(1), 19–28. https://doi.org/10.1007/BF02275262

Gevaert, C. M., & Belgiu, M. (2022). Assessing the generalization capability of deep learning networks for aerial image classification using landscape metrics. *International Journal of Applied Earth Observation and Geoinformation*, *114*, 103054. https://doi.org/10.1016/j.jag.2022.103054

Goyal, M., Tatwawadi, K., Chandak, S., & Ochoa, I. (2018). *DeepZip: Lossless Data Compression using Recurrent Neural Networks* (arXiv:1811.08162). arXiv. https://doi.org/10.48550/arXiv.1811.08162

Gupta, S., Agrawal, A., Gopalakrishnan, K., & Narayanan, P. (2015). *Deep Learning with Limited Numerical Precision* (arXiv:1502.02551). arXiv. https://doi.org/10.48550/arXiv.1502.02551

Hagen-Zanker, A. (2016). A computational framework for generalized moving windows and its application to landscape pattern analysis. *International Journal of Applied Earth Observation and Geoinformation*, *44*, 205–216. https://doi.org/10.1016/j.jag.2015.09.010

Hargis, C. D., Bissonette, J. A., & David, J. L. (1998). The behavior of landscape metrics commonly used in the study of habitat fragmentation. *Landscape Ecology*, *13*(3), 167–186. https://doi.org/10.1023/A:1007965018633

Hesselbarth, M. H. K., Sciaini, M., With, K. A., Wiegand, K., & Nowosad, J. (2019). landscapemetrics: An open-source R tool to calculate landscape metrics. *Ecography*, *42*(10), 1648–1657. https://doi.org/10.1111/ecog.04617

Johnson, J. (2018). *Rethinking floating point for deep learning* (arXiv:1811.01721). arXiv.

https://doi.org/10.48550/arXiv.1811.01721

Leclerc, G., Ilyas, A., Engstrom, L., Park, S. M., Salman, H., & Mądry, A. (2023). *FFCV: Accelerating Training by Removing Data Bottlenecks*. 12011–12020. https://openaccess.thecvf.com/content/CVPR2023/html/Leclerc_FFCV_Accelerating_Training_by_Removing_Data_Bottlenecks_CVPR_2023_paper.html

Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Ben-tzur, J., Hardt, M., Recht, B., & Talwalkar, A. (2020). A System for Massively Parallel Hyperparameter Tuning. *Proceedings of Machine Learning and Systems*, *2*, 230–246.

Li, Z., Liu, F., Yang, W., Peng, S., & Zhou, J. (2022). A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. *IEEE Transactions on Neural Networks and Learning Systems*, *33*(12), 6999–7019. https://doi.org/10.1109/TNNLS.2021.3084827

Li, Z., Zhang, S., & Dong, J. (2022). Suggestive Data Annotation for CNN-Based Building Footprint Mapping Based on Deep Active Learning and Landscape Metrics. *Remote Sensing*, *14*(13), Article 13. https://doi.org/10.3390/rs14133147

Lucas, A., Iliadis, M., Molina, R., & Katsaggelos, A. K. (2018). Using Deep Neural Networks for Inverse Problems in Imaging: Beyond Analytical Methods. *IEEE Signal Processing Magazine*, *35*(1), 20–36. https://doi.org/10.1109/MSP.2017.2760358

Ma, S., Zhang, X., Jia, C., Zhao, Z., Wang, S., & Wang, S. (2020). Image and Video Compression With Neural Networks: A Review. *IEEE Transactions on Circuits and Systems for Video Technology*, *30*(6), 1683–1698. https://doi.org/10.1109/TCSVT.2019.2910119

Ongie, G., Jalal, A., Metzler, C. A., Baraniuk, R. G., Dimakis, A. G., & Willett, R. (2020). Deep Learning Techniques for Inverse Problems in Imaging. *IEEE Journal on Selected Areas in Information Theory*, *1*(1), 39–56. https://doi.org/10.1109/JSAIT.2020.2991563

OpenAI. (2023a). *ChatGPT: LPbP Algorithm Mathematical Description*. https://chat.openai.com/share/d80552cd-86db-4e3d-a23a-504381a27fa4

OpenAI. (2023b). *ChatGPT: Optimizing Landscape Categorization Methods*. https://chat.openai.com/share/57c59fe3-14e5-486a-8c4e-9e7d9d10fdc1

Pan, S. J., & Yang, Q. (2010). A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data*

Engineering, *22*(10), 1345–1359. https://doi.org/10.1109/TKDE.2009.191

Robert J. Hijmans & Roger Bivand. (2023). *Package "terra."* https://cran.uni-muenster.de/web/packages/terra/terra.pdf

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In N. Navab, J. Hornegger, W. M. Wells, & A. F. Frangi (Eds.), *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015* (Vol. 9351, pp. 234–241). Springer International Publishing. https://doi.org/10.1007/978-3-319-24574-4_28

Salecker, J., Dislich, C., Wiegand, K., Meyer, K. M., & Pe´er, G. (2019). EFForTS-LGraf: A landscape generator for creating smallholder-driven land-use mosaics. *PLOS ONE*, *14*(9), e0222949. https://doi.org/10.1371/journal.pone.0222949

Saura, S., & Martínez-Millán, J. (2000). Landscape patterns simulation with a modified random clusters method. *Landscape Ecology*, *15*(7), 661–678. https://doi.org/10.1023/A:1008107902848

Sciaini, M., Fritsch, M., Scherer, C., & Simpkins, C. E. (2018). NLMR and landscapetools: An integrated environment for simulating and modifying neutral landscape models in R. *Methods in Ecology and Evolution*, *9*(11), 2240–2248. https://doi.org/10.1111/2041-210X.13076

Shah, V., & Hegde, C. (2018). Solving Linear Inverse Problems Using Gan Priors: An Algorithm with Provable Guarantees. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 4609–4613. https://doi.org/10.1109/ICASSP.2018.8462233

Smith, L. N. (2017). Cyclical Learning Rates for Training Neural Networks. *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 464–472. https://doi.org/10.1109/WACV.2017.58

Thomas, R. (1998). *An Update on the Four-Color Theorem*. *45*(7).

Turner, M. G., & Gardner, R. H. (2015). *Landscape Ecology in Theory and Practice: Pattern and Process*. Springer. https://doi.org/10.1007/978-1-4939-2794-4

Uuemaa, E., Antrop, M., Roosaare, J., Marja, R., & Mander, Ü. (2009). Landscape metrics and indices: An overview of their use in landscape research. *LIVING REVIEWS IN LANDSCAPE RESEARCH*, *3*(1), 1–28.

van Duynhoven, A., & Dragićević, S. (2023). A landscape metrics-based sample weighting approach for forecasting land cover change with deep learning models. *Geocarto International*, *38*(1), 2240283.

https://doi.org/10.1080/10106049.2023.2240283

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is All you Need. *Advances in Neural Information Processing Systems*, *30*. https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

Wagner, F. H., Sanchez, A., Tarabalka, Y., Lotte, R. G., Ferreira, M. P., Aidar, M. P. M., Gloor, E., Phillips, O. L., & Aragão, L. E. O. C. (2019). Using the U-net convolutional network to map forest types and disturbance in the Atlantic rainforest with very high resolution images. *Remote Sensing in Ecology and Conservation*, *5*(4), 360–375. https://doi.org/10.1002/rse2.111

Wang, X., Chen, G., Qian, G., Gao, P., Wei, X.-Y., Wang, Y., Tian, Y., & Gao, W. (2023). Large-scale Multi-modal Pre-trained Models: A Comprehensive Survey. *Machine Intelligence Research*, *20*(4), 447–482. https://doi.org/10.1007/s11633-022-1410-8

With, K. A., & King, A. W. (1997). The Use and Misuse of Neutral Landscape Models in Ecology. *Oikos*, *79*(2), 219–229. https://doi.org/10.2307/3546007

Wu, H., Judd, P., Zhang, X., Isaev, M., & Micikevicius, P. (2020). *Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation* (arXiv:2004.09602). arXiv. https://doi.org/10.48550/arXiv.2004.09602

Ying, X. (2019). An Overview of Overfitting and its Solutions. *Journal of Physics: Conference Series*, *1168*(2), 022022. https://doi.org/10.1088/1742-6596/1168/2/022022

Zurell, D., Berger, U., Sarmento Cabral, J., Jeltsch, F., Meynard, C., Münkemüller, T., Nehrbaß, N., Pagel, J., Reineking, B., Schröder, B., & Grimm, V. (2010). The virtual ecologist approach: Simulating data and observers. *Oikos*, *119*, 622–635. https://doi.org/10.1111/j.1600-0706.2009.18284.x

Appendix

Appendix Tab. 1. All landscape metrics from landscapemetrics R package (Hesselbarth et al., 2019)

LSM abbr.	LSM name	LSM type	function_name	P.	Note
ai	aggregation index	aggregation	lsm_l_ai	1	NaNs
area_cv	patch area	area and edge	lsm_l_area_cv	2	NaNs
area_mn	patch area	area and edge	lsm_l_area_mn	3	NaNs
area_sd	patch area	area and edge	lsm_l_area_sd	4	NaNs
cai_cv	core area index	core area	lsm_l_cai_cv	5	NaNs
cai_mn	core area index	core area	lsm_l_cai_mn	6	NaNs
cai_sd	core area index	core area	lsm_l_cai_sd	7	NaNs
circle_cv	related circumscribing circle	shape	lsm_l_circle_cv	8	NaNs
circle_mn	related circumscribing circle	shape	lsm_l_circle_mn	9	NaNs
circle_sd	related circumscribing circle	shape	lsm_l_circle_sd	10	NaNs
cohesion	patch cohesion index	aggregation	lsm_l_cohesion	11	NaNs
condent	conditional entropy	complexity	lsm_l_condent	12	NaNs
contag	contagion	aggregation	lsm_l_contag	13	NaNs
contig_cv	contiguity index	shape	lsm_l_contig_cv	-	excl.
contig_mn	contiguity index	shape	lsm_l_contig_mn	-	excl.
contig_sd	contiguity index	shape	lsm_l_contig_sd	-	excl.
core_cv	core area	core area	lsm_l_core_cv	14	NaNs
core_mn	core area	core area	lsm_l_core_mn	15	
core_sd	core area	core area	lsm_l_core_sd	16	NaNs
dcad	disjunct core area density	core area	lsm_l_dcad	17	
dcore_cv	disjunct core area	core area	lsm_l_dcore_cv	18	NaNs
dcore_mn	disjunct core area	core area	lsm_l_dcore_mn	19	NaNs
dcore_sd	disjunct core area	core area	lsm_l_dcore_sd	20	NaNs
division	division index	aggregation	lsm_l_division	21	
ed	edge density	area and edge	lsm_l_ed	22	
enn_cv	euclidean nearest neighbor distance	aggregation	lsm_l_enn_cv	-	excl.
enn_mn	euclidean nearest neighbor distance	aggregation	lsm_l_enn_mn	-	excl.
enn_sd	euclidean nearest neighbor distance	aggregation	lsm_l_enn_sd	-	excl.
ent	shannon entropy	complexity	lsm_l_ent	23	
frac_cv	fractal dimension index	shape	lsm_l_frac_cv	-	excl.
frac_mn	fractal dimension index	shape	lsm_l_frac_mn	24	
frac_sd	fractal dimension index	shape	lsm_l_frac_sd	25	NaNs

LSM abbr.	LSM name	LSM type	function_name	P.	Note
gyrate_cv	radius of gyration	area and edge	lsm_l_gyrate_cv	-	excl.
gyrate_mn	radius of gyration	area and edge	lsm_l_gyrate_mn	-	excl.
gyrate_sd	radius of gyration	area and edge	lsm_l_gyrate_sd	-	excl.
iji	interspersion and juxtaposition index	aggregation	lsm_l_iji	(26)	NaNs
joinent	joint entropy	complexity	lsm_l_joinent	27	
lpi	largest patch index	area and edge	lsm_l_lpi	28	
lsi	landscape shape index	aggregation	lsm_l_lsi	29	
mesh	effective mesh size	aggregation	lsm_l_mesh	30	
msidi	modified simpson's diversity index	diversity	lsm_l_msidi	31	NaNs
msiei	modified simpson's evenness index	diversity	lsm_l_msiei	32	NaNs
mutinf	mutual information	complexity	lsm_l_mutinf	33	
ndca	number of disjunct core areas	core area	lsm_l_ndca	34	
np	number of patches	aggregation	lsm_l_np	35	
pafrac	perimeter-area fractal dimension	shape	lsm_l_pafrac	(36)	NaNs
para_cv	perimeter-area ratio	shape	lsm_l_para_cv	37	NaNs
para_mn	perimeter-area ratio	shape	lsm_l_para_mn	38	
para_sd	perimeter-area ratio	shape	lsm_l_para_sd	39	NaNs
pd	patch density	aggregation	lsm_l_pd	40	
pladj	percentage of like adjacencies	aggregation	lsm_l_pladj	41	
pr	patch richness	diversity	lsm_l_pr	42	
prd	patch richness density	diversity	lsm_l_prd	43	
relmutinf	relative mutual information	complexity	lsm_l_relmutinf	44	
rpr	relative patch richness	diversity	lsm_l_rpr	(45)	NaNs
shape_cv	shape index	shape	lsm_l_shape_cv	46	NaNs
shape_mn	shape index	shape	lsm_l_shape_mn	47	
shape_sd	shape index	shape	lsm_l_shape_sd	48	NaNs
shdi	shannon's diversity index	diversity	lsm_l_shdi	49	NaNs
shei	shannon's evenness index	diversity	lsm_l_shei	50	NaNs
sidi	simpson's diversity index	diversity	lsm_l_sidi	51	
siei	simspon's evenness index	diversity	lsm_l_siei	52	NaNs
split	splitting index	aggregation	lsm_l_split	53	
ta	total area	area and edge	lsm_l_ta	(54)	
tca	total core area	core area	lsm_l_tca	55	
te	total edge	area and edge	lsm_l_te	56	

Appendix Tab. 2. Listing parallels and differences between deep learning and landscape ecology terminology.

Term	Deep Learning	Landscape ecology
Metric	The evaluation metric (here: r2)	Landscape metric (LSM)
Matrix	Mathematical matrix (i.e. matrix multiplication in CNN)	Background of landscape (non-habitat)
Moving Window	Kernel in CNN, moving over the input tensor	Moving window for (local) LSM calculation
Class	Class in python is an object; Classes are classified by the classifier i.e. in object detection in deep learning	Landscape class, all patches with same value belong to one ls class
Patch	Patch in CNNs is a part of the input image with size equal to kernel dim, i.e. 3x3. (CNN)	Landscape patch, connected cells with same value

Annex: Declaration on the use of ChatGPT and comparable tools in the context of examinations

In this paper, I have used ChatGPT or another AI as follows.:

[ ] not at all

[x] during brainstorming

[ ] when creating the outline

[ ] to write individual passages, altogether to the extent of ...% of the entire text

[x] for the development of software source texts

[x] for optimizing or restructuring software source texts

[x] for proofreading or optimizing

[ ] further, namely: …

I hereby declare that I have stated all uses completely. Missing or incorrect information will be considered as an attempt to cheat.

Verification of examination registration in FlexNow

Name: Mr Jonathan Gehret
Matriculation No.: 23301512

Semester: WS23/24
Degree Course: Forest and Ecosystem Sciences (Master of Science)
Module: Master's Thesis - Specialization 1: Ecosystem Analysis and Modelling
Exam: Master's Thesis
Lecturer: Prof. Dr. Kerstin Wiegand

Declaration

I hereby declare that I have produced this work independently and without outside assistance, and have used only the sources and tools stated.

I have clearly identified the sources of any sections from other works that I have quoted or given in essence.

I have complied with the guidelines on good academic practice at the University of Göttingen.

If a digital version has been submitted, it is identical to the written one.

I am aware that failure to comply with these principles will result in the examination being graded “nicht bestanden”, i.e. failed.

Göttingen, 13th October 2023 Jonathan Gehret

Verification of examination registration in FlexNow

Name: Mr Jonathan Gehret
Matriculation No.: 23301512

Semester: WS23/24
Degree Course: Forest and Ecosystem Sciences (Master of Science)
Module: Master's Thesis - Specialization 1: Ecosystem Analysis and Modelling
Exam: Master's Thesis
Lecturer: Prof. Dr. Fabian Sinz

Declaration

I hereby declare that I have produced this work independently and without outside assistance, and have used only the sources and tools stated.

I have clearly identified the sources of any sections from other works that I have quoted or given in essence.

I have complied with the guidelines on good academic practice at the University of Göttingen.

If a digital version has been submitted, it is identical to the written one.

I am aware that failure to comply with these principles will result in the examination being graded “nicht bestanden”, i.e. failed.

Göttingen, 13th October 2023 Jonathan Gehret