Deep Learning-Based Dike Identification using U-Net and Virtual Layers

Konstantin Engelmayer
Universitaet Marburg

April 17, 2024

Abstract

This paper investigates the potential of the U-Net convolutional neural network architecture for large-scale dike identification and mapping, addressing the escalating flood risks due to increased heavy precipitation events in Germany. Despite the crucial role of dikes in flood management, many regions, lack comprehensive digital dike databases, which impedes efficient flood risk management. This study pioneers the application of the U-Net for dike identification, employing a hybrid approach of R and Python tools to manage and process spatial and elevation data across the Rhine region in Hessen. Through detailed data preprocessing, including the use of false color composite images and rank-based normalization, the study explores the effectiveness of U-Net in recognizing dike structures. The results demonstrate the model's capability to identify dikes with high precision, though challenges remain in differentiating dikes from similar structures in urban areas and along roads. The study underscores the necessity of refining the model's architecture to enhance its practical application for developing extensive dike databases, which is vital for improving flood defense strategies in response to changing climatic conditions. The research sets a foundation for future advancements in the application of deep learning techniques in dike identification.

Contents

1 Introduction

An analysis by the German Weather Service in 2016 indicated a significant 25 % increase in the frequency of heavy precip-

itation events in Germany over the previous 65 years, particularly during the winter months [1]. Projections from regional climate models suggest that this trend is likely to continue at a similar rate until the end of the century [1]. Recent flooding in Germany highlight the urgent need to update flood management strategies to this trend. Dikes play a crucial role in flood management systems in Germany, acting as primary barriers against floodwaters and protecting inland areas. However, the effectiveness of dikes depends largely on their condition [2]. The recent floods and extensive literature have clearly shown that poorly maintained dikes are more susceptible to failure [3, 4, 5, 6]. To improve dike maintenance, comprehensive, digitally managed databases are essential. Yet, most German federal states lack these digital databases. In North Rhine-Westphalia, for example, dike data can only be obtained from local authorities or through an analog dike cadastre, and only upon request, significantly slowing the information gathering process [7]. A dike information system for addressing questions beyond regional boundaries is notably absent, leaving the analysis of dikes at national or international levels almost impossible given the existing data infrastructure.

Given the absence of literature on the use of U-Net for dike identification, this paper pioneers in testing the potential of U-Net in identifying dikes on a large scale. This initiative aims to facilitate the creation of extensive dike databases, enhancing the management and maintenance of these critical flood defense structures.

2 Materials and Methods

2.1 Data and Research Area

A U-Net model is a type of convolutional neural network known for its effectiveness in image segmentation tasks [8]. To test the U-Net's capability of identifying dikes, the model in this study was initially designed and tested for identifying dikes in a limited area within the Hessen region (see figure 1). Following successful testing in this smaller area, the intention was to apply the model on the whole Rhine region in Hessen to evaluate and enhance its scalability and accuracy in broader geographical applications. Here, the training, validating and testing of the model were carried out using dike data from the Bergstraße district (see figure 1) that had been digitized by hand, based on the Plan of Rhine dike system by the city of Darmstadt [9]. In the digitizing process, short dike segments looking not dike like due to near by buildings or other structures were omitted to reduce false dike identifications. In this study, dikes are characterized as man-made constructions intended for flood defense, distinguished by their long and continuous structure. Even though road embankments can closely resemble dikes in appearance, they are only defined as dikes within our criteria if they are proximate to a river and explicitly serve the purpose of flood protection. Consequently, short structures, even if they appear dike-like, do not qualify as dikes, as they were most likely not built for flood protection. The primary data layers utilized in this model included elevation data and a virtual layer that represented the calculated distance to the nearest river. each pixel in these layers corresponds to a 1 x 1 meter resolution. The elevation data and water-network layers were sourced from the download center of the Hessian Administration for Soil Management and Geoinformation [10].

2.2 R and Python

In this study, we utilized a combination of R (version 4.3.1) and Python (version 3.10) packages to facilitate the dike identification. We employed the 'envimaR' pack-

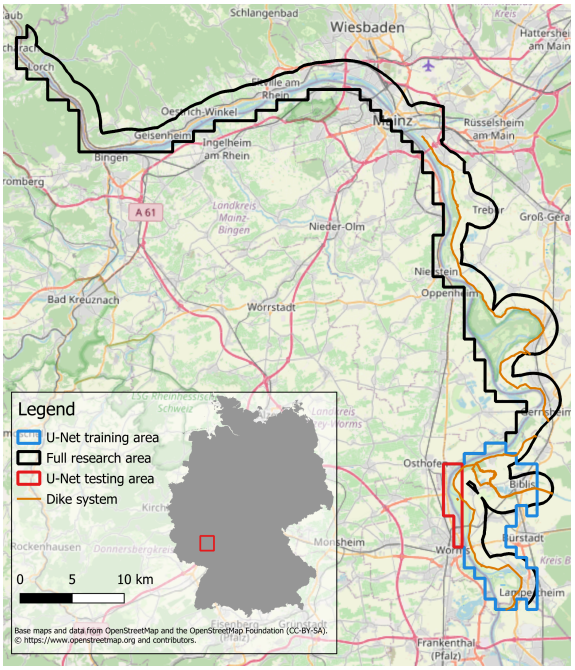

Figure 1: Research area Rhine river in Hessen.

age [11] to set up and manage environments, enabling reproduction. 'Terra' [12] was used for raster data manipulation and 'sf' [13] for vector data handling. The 'reticulate' [14] package provided an integration between R and Python, allowing the use of Python's computational libraries 'keras' [15] and 'tenorflow' [16] directly within the R workspace. 'Keras' alongside 'tensorflow' were used for developing and training the deep learning model.

For specific tasks related to calculating relative heights, Python libraries were used for their efficiency and specialized capabilities. 'NumPy' [17] played a crucial role in performing numerical operations on arrays, enabling the efficient manipulation of digital elevation model (DEM) data. 'Rasterio' [18] was instrumental in reading, writing, and conducting window-based processing of raster datasets, essential for managing large spatial datasets. For spatial indexing and nearest-neighbor searches, critical in pinpointing the nearest river pixels to any given DEM pixel, we utilized the 'cKDTree' module from 'SciPy' [19]. The 'binary_dilation' function from 'scikit-image' [20] was applied for dilating binary masks. Additionally, 'tqdm' [21] provided a valuable utility for progress tracking during long-running data processing tasks, enhancing usability and monitoring. These tools collectively supported a streamlined, efficient approach to calculate the relative heiht of every pixel to relative to the next river in Python.

To further enhance the training speed of the model, NVIDIA's CUDA toolkit for GPU acceleration was used [22]. By utilizing CUDA, significant reduction in calculation times were achieved, approximately 8 times faster than CPU-only execution. This acceleration was crucial in handling large-scale data processing and complex model training more effectively and in a timely manner.

2.3 U-Net

For the identification of dikes in this study, the U-Net convolutional neural network architecture was employed. The U-Net, developed for biomedical image segmentation in 2015 by Ronneberger et al. [8], had been initially used for tasks such as segmenting neural structures in electron microscopic stacks and cell tracking in light microscopy images. Its design, characterized by a contracting path to capture context and a symmetric expanding path that enables precise localization (see figure 2), proved highly effective in dealing with spatial structures and varying shapes found in biomedical imagery [23].

The architecture's suitability for dike identification lays in its ability to address similar challenges in geospatial data [24]. As effectively as it segments and identifies biological structures, it was expected that the U-Net could accurately delineate shapes of dikes. Additionally, the network's capability in learning from limited datasets was particularly beneficial for specialized applications like dike identification, where extensive labeled data might not always be avail-

able.

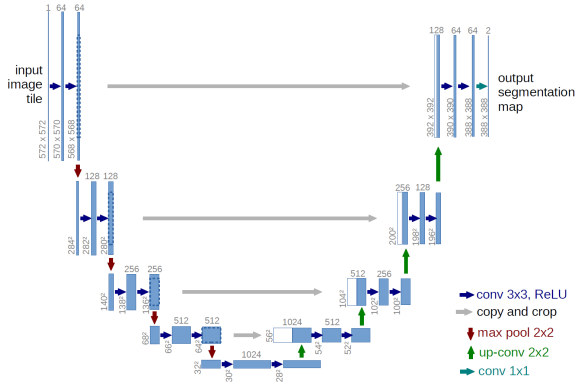

Figure 2: U-net architecture, example for a minimum resolution of 32x32 pixels. Each blue rectangle represents a multi-channel feature map, with the quantity of channels indicated above the rectangle. The dimensions in the x-y plane are noted on the bottom left corner of each rectangle. White rectangles symbolize duplicated feature maps, and the various operations are illustrated by arrows [8].

2.4 Data Preprocessing

In the original design, the U-Net model was specifically tailored for RGB images, well-suited for biomedical segmentation tasks. However, in the context of dike identification, the segmentation challenges are markedly different. Standard RGB imagery often falls short in distinguishing features such as meadows and dikes, as they can appear remarkably similar looking on them from above. This similarity is primarily due to the absence of elevation information in RGB images. Additionally, road embankments, visually similar to asphalted roads on dikes, further complicate the segmentation process in RGB imagery.

To address this challenge, the approach in this study involved the use of false color composite (FCC) images. These FCC images are not standard RGB images but are instead composed of data layers that include the DEM and the calculated distances to the next river. Initially, a third layer was selected to represent the presence of water during a 100-year flood, known as HQ100, which refers to a high water flow statistically reached or exceeded once every 100 years. HQ100 typically reaches up to dikes but seldom overflows them, which makes it a useful layer for dike identification. However, recognizing the necessity for the model to be applicable universally, and considering the unavailability of HQ100 layers for most regions, this layer was omitted. To comply with the requirements of the U-Net architecture, which mandates input in the form of three layers (analogous to RGB channels in an image), the elevation data was utilized twice. Recent scientific papers have moved away from the original U-Net architecture to accommodate different numbers of input layers, eliminating the constraint of having three layers [25, 26]. Ideally, the architecture should have been adjusted to accommodate just two layers, potentially enhancing the model's efficiency. However, due to the scope of the course for which this paper is written, which did not cover such modifications, the original architecture was retained, with the elevation layer duplicated. While this approach allows us to demonstrate the model's potential, it does not yield optimal results.

The inclusion of DEM data introduces an elevation aspect, providing crucial information about the topography and subtle height variations indicative of dike structures. Similarly, incorporating the distance to the nearest water source adds a contextual layer, offering insights into the likelihood of a structure being a dike based on its proximity to water bodies.

2.4.1 Data Normalization

A key aspect of utilizing the layers used in FCC images is the need to fit the virtual layer within the numerical range of 0 to 255, enabling them to be interpreted as color bands in a .png image. This requirement necessitates the compression or stretching

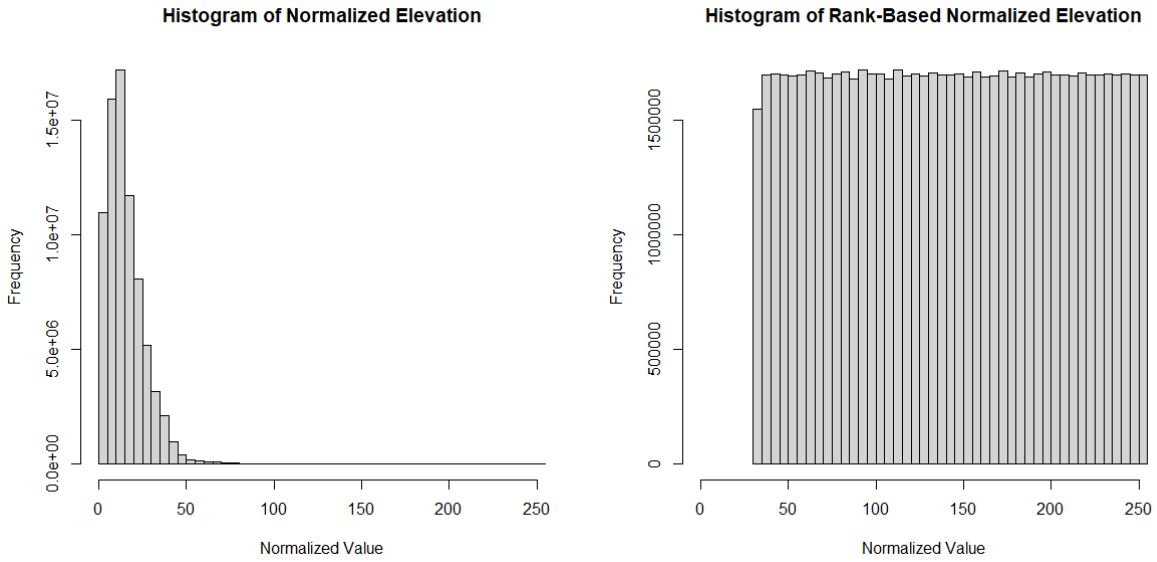

Figure 3: Distribution of elevation values across the range of 0 to 255, comparing two different normalization approaches. On the left, simple normalization is depicted; on the right, rank-based normalization. Values equaling zero, predominantly representing river pixels, have been omitted to enhance the readability of the plots.

of the values to ensure that the maximum value aligns with 255 and the minimum with 0. For normalization of the two layers, we initially applied the following formula:

$$\text{Normalized values} = \frac{\text{Specific values} - \text{Min value}}{\text{Max value} - \text{Min value}} \times 255 \quad (1)$$

However, this basic normalization method results in minimal contrast for datasets with a highly skewed distribution, consolidating numerous values into discrete integers within a range of 0 to 255. To further refine the visualization and enhance the contrast between different data values, rank-based normalization was employed. This technique involves converting the original data values into ranks and then scaling these ranks to the 0-255 range, thus ensuring a more uniform distribution of pixel values across the entire spectrum.

The rank-based normalization process can be mathematically represented as follows:

$$S(x_i) = \left(\frac{R(x_i) - 1}{n - 1} \right) \times 255 \quad (2)$$

Where $S(x_i)$ is the scaled value to be used in the image, $R(x_i)$ is the rank of the original data value x_i in the dataset, and n is the total number of observations (listing 2).

Figure 3 illustrate the effect of applying rank-based normalization to the data. The histogram on the left displays the distribution of the original normalized values, where it is evident that the data values are concentrated in a specific range, limiting the effective use of the available color spectrum. The right plot shows the histogram after rank-based normalization, showcasing a more uniform distribution of values and, consequently, a significant enhancement in the contrast and detail perceptible in the FCC images.

The adoption of rank-based normalization thus plays a critical role in the preprocessing of data for FCC imagery, ensuring that the full dynamic range is utilized

efficiently to produce images with greater detail.

2.4.2 Calculation of Relative Heights

Another challenge was the application of the U-Net model to broader regions, such as the Rhine area in Hessen, where the increased scope led to a wider range of DEM values. This expansion of the value range is a logical consequence of rivers always flowing downhill, affecting the distribution and variability of elevation data. This means that the longer the river chosen for dike identification, the larger the range of DEM values. When these values are scaled to fit within the binary range of 0 to 255, it results in significantly reduced contrast due to the merging of diverse elevation values into a single class within the 0-255 range, leading to information loss. This low contrast hindered the U-Net's ability to effectively identify dikes in bigger areas.

To address this issue, the relative height of every pixel to the nearest river was calculated, substituting the original DEM layer with this recalculated layer. This approach ensured that, no matter the length of the river, the elevation values stayed roughly within the same range. By normalizing the elevation data in relation to river height, we minimized the variation in DEM values across different geographic areas, thereby reducing the problem of information loss and maintaining contrast in the scaled images. The calculation of relative heights also allowed for setting all height values above 25 to 25, effectively eliminating possible mountains and high grounds adjacent to the river, to further reduce the range of values and increase contrast. Theses adaptations were crucial for preserving the detail needed for the U-Net to effectively identify dikes across varied landscapes (listing 1).

2.4.3 False Color Composite

In Figure 4, the distance to water is represented in the green channel and the relative elevation with respect to the nearest river pixel in the green and blue channel. The pixels distant to the river stand out in lush green, while the dikes are colored in a brighter magenta.

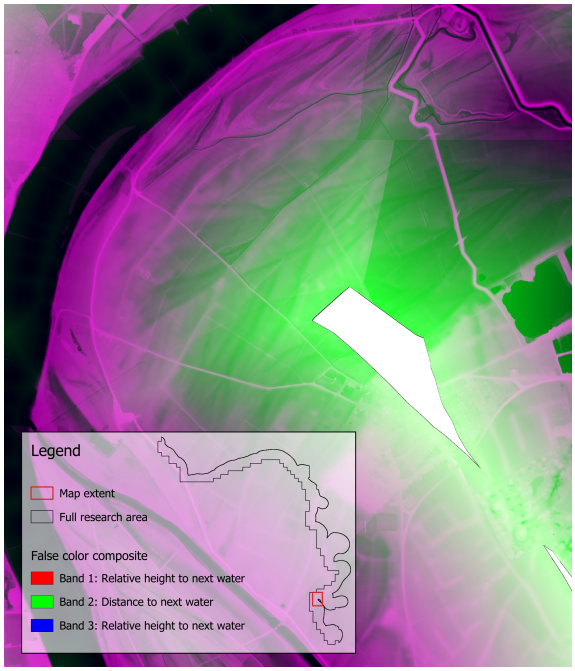

Figure 4: False color composite of a small extent of the training are of the U-Net.

2.5 U-Net implementation

First, vector data containing dike training data was converted into a raster format, creating a mask layer, as shown in listing 9. Subsequently, this mask layer alongside the FCC layer were segmented into 128 x 128 pixel images, matching the model's input dimensions, also shown in listing 9. Following this segmentation, an augmentation process was applied, which involved rotating both the images and the masks, and also randomly adjusting the saturation, brightness, and contrast levels within predefined ranges, as in listing 10. Upon completing the data preparation phase, 80% of the training data was randomly selected and

then used to train the model, also shown in listing 10, while the other 20% were used for the model's validation. The trained model was then employed to make predictions on a distinct test subset, which was segregated from the training dataset, equivalent to listing 11. The ultimate predictions were conducted across the entire Rhine area in Hessen (listing 12). The outputs generated by the U-Net model across all images were merged to form a comprehensive layer, comparable to a heatmap, indicating dike presence throughout the region.

3 Results

Upon evaluating the U-Net model across multiple epochs, the optimal performance was achieved with a batch size of 16 at the 17th epoch, marking it as the most effective iteration for dike identification. Throughout the iterative training process, the model demonstrated a consistent improvement in accuracy when applied to the validation dataset. An initial, pronounced decline in training and validation loss was observed in the first epoch, followed by a sustained, gradual reduction (Figure 5).

Both training and validation plots reveal high accuracy and low loss values.

Figure 6 illustrates the performance of the U-Net model in dike prediction within the testing area, where the model assigned higher confidence scores to the upper and lower parts of the dike structures while displaying moderate certainty for the central segments. Achieving a loss of 0.0105 and an accuracy of 0.9958, the model demonstrated effectiveness in dike identification, with these metrics closely aligning with the similarly high accuracy and low loss observed in both the training and validation phases.

Additionally, the figure reveals instances of erroneous classification west of the Rhine River, where certain roadways were mistakenly identified as dikes.

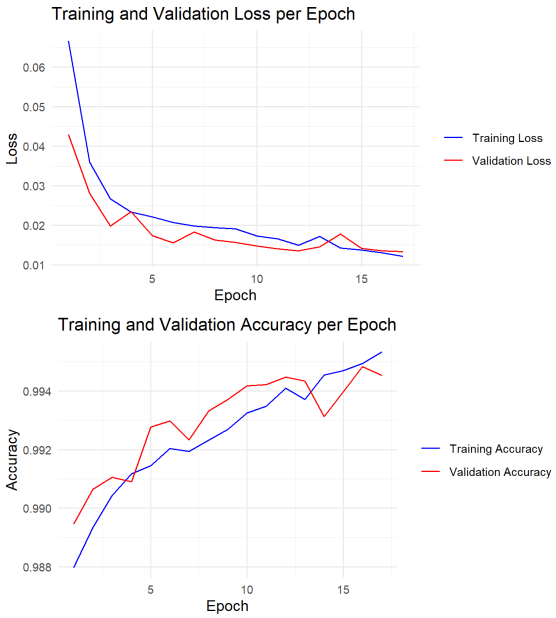

Figure 5: Top plot shows the loss and bottom plot the accuracy of the training's- and validation area.

Figure 7 presents the model's predictive analysis for dike presence across the entirety of the Rhine area in Hesse. Notably, in the undiked, hilly northern regions, the model exhibited high precision, with only a sparse scattering of pixels incorrectly identified as dikes. Progressing southward, the model accurately mapped an extensive stretch of dike infrastructure spanning from the border of Ginsheim to Leeheim and onwards to the boarder of Erfelden. A similar proficiency was observed in identifying the dike section extending from Groß-Rohrheim to the start of the Lampertheim Altrhein, where high likelihood values were associated with true dike features.

Conversely, the urban areas of Ginsheim, Biebesheim, Stockstadt, and Erfelden, as well as the northern part of the dike at Gustavsheim, revealed the model's vulnerabilities, with a noticeable decline in classification accuracy. Moreover, a consistent pattern of miss-classification was detected along the west of the Rhine River, with the model erroneously interpreting roads as dikes over the entire length of the river.

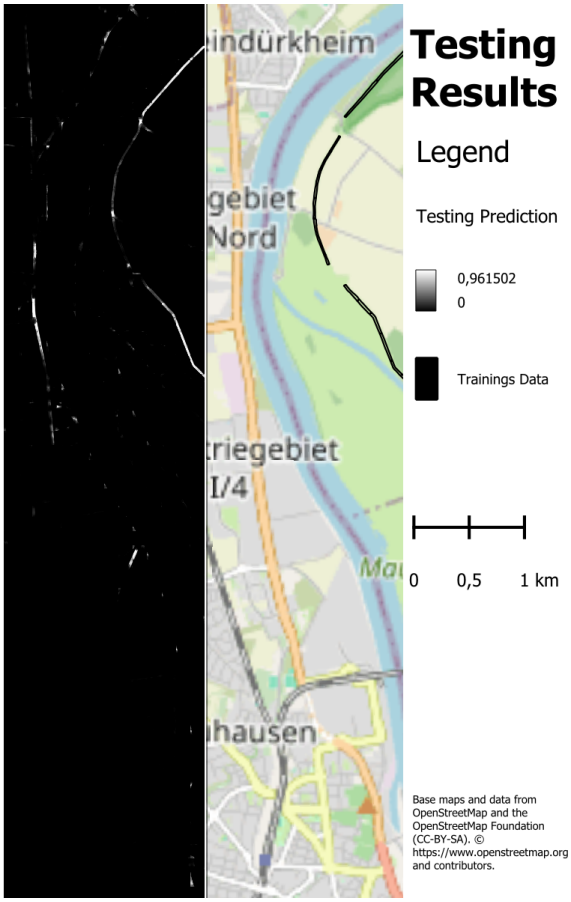

Figure 6: Testing Results containing likelihood of a pixel being a dike. Comparison of test results and real dike data.

4 Discussion

The core objective of this study was to configure a U-Net model capable of accurately identifying dike structures, enabling the development of a national or international dike dataset to enhance flood defense strategies. An initial assessment of model performance is usually based on the evaluation of accuracy and loss metrics, which provide an initial indication of the models quality and are therefore presented at the outset.

Observations from the model's validation phase indicated a consistent improvement in accuracy for both the training and validation data, though the latter displayed somewhat less stability. The irregularities can primarily be attributed to the characteristics of the validation area. While being smaller in size than the training area, the validation area is effected stronger by single false classifications and therefore shows bigger variances in loss and accuracy over the different epochs. The high accuracy and low loss values, which supposedly indicate an effective model, should be treated with caution. Due to the low proportion of dike pixels relative to non-dike pixels, the accuracy metric may be artificially inflated; the model could achieve high accuracy simply by correctly identifying the majority class – in this case, non-dike areas, even if all dikes are classified wrong. This skew in the class distribution suggests that alternative metrics, such as the kappa index, would be more informative as they account for different class sizes and offer a more balanced view of the model's performance. However this metric is not an standard output of the training phase in tensorflow.

The application of the model to the testing area yielded results of comparable quality to those observed during the validation and training phases. In the testing area, the model effectively identified the upper and lower sections of the dike, while the central part received only moderate likelihood values for being a dike. This discrepancy may be attributed to the central section's thinner structure compared to those in the training dataset. This outcome highlights the importance of training the model on a variety of dike structures to enhance its ability to accurately identify dikes with varying thicknesses. Additionally, the testing showed instances where streets west of the Rhine River were incorrectly classified as dikes, likely due to their proximity to the river and their structural similarities. However, the predicted values for these missclassifications were neither as high nor as continuous as those assigned to actual dike structures, indicating a lower level of confidence in these predictions.

The application of the model to the whole Rhine area in Hesse demonstrated proficiency in identifying isolated dikes situated

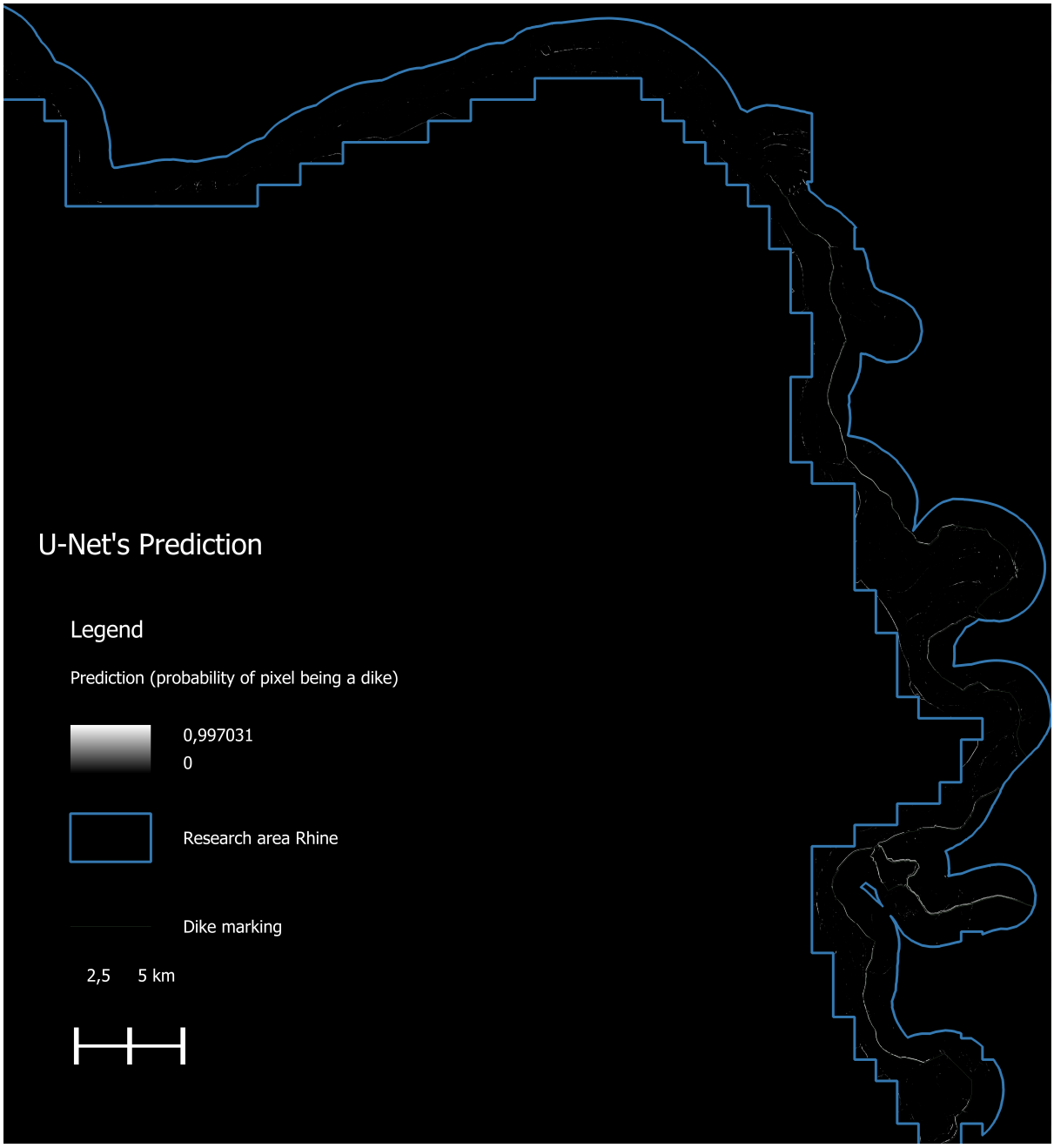

Figure 7: Results of dike prediction of the whole Rhine river region in Hesse, dike marking shows the coarse location of the dike, the prediction shows the more precise location (Zoom in for detailed view).

in close proximity to rivers, where the dike structures are clearly represented in the digital elevation model (DEM), like the dike stretch spanning from the border of Ginsheim to the boarder of Erfelden. This success is attributed to the distinct topographical signature that dikes imprint on the landscape, which is captured in the DEM. However, in urban environments, specifically in the proximity of other buildings where dike structures blend into the surrounding architecture, the model struggled to distinguish dikes effectively. This can be attributed to the training, where dike structures not looking typically dike like were omitted to avoid false classifications in the identification process. The fusion of dike features with adjacent urban structures results in reduced visibility of the dike in the DEM, which the model cannot reliably interpret, without falsely interpreting other urban areas as dikes. Differentiating roads from dikes, adjacent to the river, presented another significant challenge as both share similar structural characteristics and elevation profiles. This led to false classification of roads along the west of the river.

The layers utilized in this study significantly influenced the model's performance. The DEM layer proved indispensable for identifying dikes, as it clearly delineated the structural features of dikes. This layer was crucial for recognizing the elevated contours that typify dike constructions against the surrounding landscape. Conversely, the layer measuring the distance to the nearest water body had a dual impact. While it helped reduce false classifications of dike-like streets that are further from any water body, it simultaneously increased the likelihood of falsely classifying streets, which are close to water, as dikes. This layer also presented challenges when dikes were situated further away from the river than typical, demonstrating its limitations in such scenarios. When the research area is focused solely on areas adjacent to water bodies, the utility of the distance to water layer diminishes, suggesting that it could be replaced by other layers that provide valuable information and are widely available across regions. These considerations underscore the need for a careful selection of layers that not only enhance the model's accuracy but are also universally accessible to ensure the method's applicability in diverse geographical settings.

The methodology employed in this study, while demonstrating potential, also manifests limitations that must be acknowledged. The necessity for intricate preprocessing, particularly the height normalization of data relative to the nearest water body, is a time-consuming process that constrains the scalability and rapid deployment of the model. This limitation reflects back on the initial objectives outlined in the introduction, where the aim was to develop a fast and efficient method for dike identification using minimal data. The results suggest that while the model holds promise, further refinement is needed to meet these goals without the extensive preprocessing currently required. Notably, these time-consuming preprocessing steps could potentially be bypassed by adapting the U-Net's architecture, as mentioned in the methods section. The transition from RGB images to GeoTIFF's would make the normalization of the data to 0 and 255 superfluous and thus considerably speed up pre-processing. Furthermore reducing the input layers to two is likely to reduce training time of the model. By embracing this architectural adjustment, the model could directly process the more informative and unaltered data, thus expediting the preparation phase and enhancing the overall efficiency of the dike identification process.

5 Conclusion

Despite the challenges described, the U-Net model has shown it's potential to be a valuable tool in the field of dike identifi-

cation. It is important, however, to refine the model to mitigate the identified limitations, which includes enhancing its ability to differentiate between similar structures and adjusting the U-Nets's architecture to use GeoTIFF's as input data to streamline the preprocessing. Achieving these improvements would significantly enhance the model's utility in real-world scenarios, enabling its application across extensive geographical regions.

In conclusion, this study showcases the U-Net model as a conceptually sound approach with significant potential for dike identification, especially important in the context of escalating flood risks. However, the practical application of this method is hindered by extensive data preprocessing demands and the model's sensitivity to environmental complexities. The application of the U-Net model could markedly advance disaster management by improving flood preparedness, aligning with the growing need for robust environmental management strategies. Although the study underlines the model's potential, it also highlights the necessity of further developments to mitigate these challenges. Progressing beyond the current limitations is essential to achieve a rapid and data-efficient solution for national or international dike dataset development, which was the original vision of this research. The potential benefits suggest that despite the hurdles, pursuing this technological avenue could yield substantial rewards in disaster management and environmental protection.

References

[1] Paul Becker et al. *Die Entwicklung von Starkniederschlägen in Deutschland*. July 2016. URL: https://www.dwd.de/DE/leistungen/besondereereignisse/niederschlag/20160719_entwicklung_starkniederschlag_deutschland.pdf (visited on 12/30/2023).

[2] Patrice Mériaux and Paul Royet. *Surveillance, maintenance and diagnosis of flood protection dikes*. éditions Quae, 2007. URL: https://www.quae-open.com/produit/111/9782759200375/surveillance-maintenance-and-diagnosis-of-flood-protection-dikes.

[3] Sergiy Vorogushyn, Bruno Merz, and Heiko Apel. "Development of dike fragility curves for piping and micro-instability breach mechanisms". In: *Natural Hazards and Earth System Sciences* 9.4 (2009), pp. 1383–1401.

[4] Lukas Schmocker and Willi H. Hager. "Overtopping and breaching of dikes – Breach profile and breach flow". eng. In: *River Flow 2010*. Ed. by Bundesanstalt fÜr Wasserbau et al. Karlsruhe: Bundesanstalt fÜr Wasserbau, 2010, p. 515. URL: https://hdl.handle.net/20.500.11970/99684.

[5] Bruno Merz. *Hochwasserrisiken*. Stuttgart, Germany: Schweizerbart Science Publishers, Apr. 2006. ISBN: 9783510652204. URL: http://www.schweizerbart.de//publications/detail/isbn/9783510652204/Merz%5C_Hochwasserrisiken.

[6] Kristian Pilarczyk. *Dikes and revetments: design, maintenance and safety assessment*. Routledge, 2017.

[7] Ministry for Environment, Agriculture, Nature Conservation and Consumer Protection of the State of North Rhine-Westphalia. *Leitfaden Hochwassergefahrenkarten NRW*. 2003. URL: https://www.yumpu.com/de/document/read/21192206/leitfaden-hochwassergefahrenkarten-nrw (visited on 12/30/2023).

[8] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *arXiv preprint arXiv:1505.04597* (2015). URL: https://arxiv.org/pdf/1505.04597.pdf.

[9] Regierungspräsidium Darmstadt. *Plan Rheindeichsystem*. Apr. 2022. URL: https://rp-darmstadt.hessen.de/sites/rp-darmstadt.hessen.de/files/2022-04/plan_rheindeichsystem.pdf (visited on 04/02/2024).

[10] Hessian Administration for Soil Management and Geoinformation. *Geodatenzentrum*. URL: https://gds.hessen.de/INTERSHOP/web/WFS/HLBG-Geodaten-Site/de_DE/-/EUR/ViewDownloadcenter-Start (visited on 12/30/2023).

[11] Environmental Informatics Lab @ Marburg University. *envimaR*. 2021. URL: https://github.com/envima/envimaR (visited on 10/10/2023).

[12] Robert J. Hijmans. *Terra: Spatial Data Analysis*. R package version 1.7-55. 2023. URL: https://CRAN.R-project.org/package=terra.

[13] Edzer Pebesma. *Simple Features for R*. R package version 1.0-14. 2023. URL: https://CRAN.R-project.org/package=sf.

[14] J.J. Allaire et al. *Interface to 'Python'*. R package version 1.34.0. 2023. URL: https://CRAN.R-project.org/package=reticulate.

[15] J.J. Allaire and François Chollet. *R Interface to 'Keras'*. R package version 2.13.0. 2023. URL: https://CRAN.R-project.org/package=keras.

[16] J.J. Allaire and Yuan Tang. *TensorFlow for R*. R package version 2.14.0. 2023. URL: https://CRAN.R-project.org/package=tensorflow.

[17] Charles R. Harris et al. *NumPy: A fundamental package for scientific computing with Python*. 2023. URL: https://numpy.org/.

[18] Sean Gillies et al. *Rasterio: Access to geospatial raster data*. 2023. URL: https://rasterio.readthedocs.io/en/latest/.

[19] Pauli Virtanen et al. *SciPy: Open source scientific tools for Python*. Version 1.5.2. 2023. URL: https://www.scipy.org/.

[20] Stefan van der Walt et al. *scikit-image: Image processing in Python*. 2023. URL: https://scikit-image.org/.

[21] Caspar Lant et al. *tqdm: A Fast, Extensible Progress Bar for Python and CLI*. 2023. URL: https://tqdm.github.io/.

[22] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. *CUDA, release: 10.2.89*. 2020. URL: https://developer.nvidia.com/cuda-toolkit.

[23] Reza Azad et al. *Medical Image Segmentation Review: The success of U-Net*. 2022. arXiv: 2211.14830 [eess.IV].

[24] Bipul Neupane, Teerayut Horanont, and Jagannath Aryal. "Deep Learning-Based Semantic Segmentation of Urban Features in Satellite Images: A Review and Meta-Analysis". In: *Remote Sensing* 13.4 (2021). ISSN: 2072-4292. DOI: 10.3390/rs13040808. URL: https://www.mdpi.com/2072-4292/13/4/808.

[25] Lazhar Khelifi and Max Mignotte. "Deep Learning for Change Detection in Remote Sensing Images: Comprehensive Review and Meta-Analysis". In: *IEEE Access* 8 (2020), pp. 126385–126400. DOI: 10.1109/ACCESS.2020.3008036.

[26] Geoffrey A. Fricker et al. “A Convolutional Neural Network Classifier Identifies Tree Species in Mixed-Conifer Forest from Hyperspectral Imagery”. In: *Remote Sensing* 11.19 (2019). ISSN: 2072-4292. DOI: 10.3390/rs11192326. URL: https://www.mdpi.com/2072-4292/11/19/2326.

Appendices

Python Code

Calculation of relative heights

```
import numpy as np
import rasterio
from scipy.spatial import cKDTree
from rasterio.windows import Window
from tqdm import tqdm
from skimage.morphology import binary_dilation

def create_mask_from_dem(dem_data):
    """Create a binary mask where True/1 represents valid data and False/0 represents NA."""
    return np.isfinite(dem_data)

def find_valid_data_stretches_in_row(row_mask):
    """Find start and end indices of contiguous True regions in the row."""
    stretches = []
    start = None
    for i, value in enumerate(row_mask):
        if value and start is None:
            start = i
        elif not value and start is not None:
            stretches.append((start, i))
            start = None
    if start is not None:  # Handle case where row ends with a stretch of valid data
        stretches.append((start, len(row_mask)))
    return stretches

def process_chunk(dem_data, window, affine_transform, tree, river_dem_values, output_raster):
    """Extract and process each window from the DEM, calculating relative heights."""
    # Adjust for window's relative position within the entire DEM
    if dem_data.ndim > 2:
        dem_data = dem_data.squeeze()  # Ensure dem_data is 2D

    rows, cols = np.indices((window.height, window.width))
    rows_flat, cols_flat = rows.ravel(), cols.ravel()

    # Adjust row and column indices based on the window's position
    rows_flat += window.row_off
    cols_flat += window.col_off

    # Convert to spatial coordinates
    x, y = affine_transform * (cols_flat, rows_flat)

    # Query nearest river pixel for each DEM pixel
    _, indices = tree.query(np.column_stack((x, y)), k=1)
    nearest_river_dem_values = river_dem_values[indices]

    # Calculate and write relative heights
    relative_heights = dem_data[window.toslices()] -
```

```
    nearest_river_dem_values
48      output_raster.write(relative_heights.astype(np.float32), 1, window=
        window)
49
50  # Setup paths and open raster files
51  dem_raster_path = 'masked_dem.tif'
52  river_raster_path = 'Rhein_Hessen_waternowater.tif'
53  output_file_path = 'relativ_heights_test.tif'
54
55  with rasterio.open(dem_raster_path) as dem_raster, rasterio.open(
        river_raster_path) as river_raster:
56      dem_data = dem_raster.read(1)
57      mask = create_mask_from_dem(dem_data)
58      dilated_mask = binary_dilation(mask, footprint=np.ones((3, 3)))
59
60      river_indices = np.where(river_raster.read(1) == 1)
61      river_points = np.column_stack((river_indices[0], river_indices[1]))
62      tree = cKDTree(river_points)
63      river_dem_values = dem_data[river_indices]
64
65      out_meta = dem_raster.meta.copy()
66      out_meta.update(dtype=rasterio.float32, count=1, nodata=np.nan)
67
68      with rasterio.open(output_file_path, 'w', **out_meta) as output_raster:
69          # Iterate with tqdm for progress bar
70          for row_index in tqdm(range(dem_data.shape[0]), desc='Processing
        Rows'):
71              stretches = find_valid_data_stretches_in_row(dilated_mask[
        row_index, :])
72              for start_col, end_col in stretches:
73                  window = Window.from_slices((row_index, row_index+1), (
        start_col, end_col))
74                  affine_transform = dem_raster.window_transform(window)
75                  process_chunk(dem_data, window, affine_transform, tree,
        river_dem_values, output_raster)
```

Listing 1: Python code for calculating relative heights

Rank based normalization

```
1   import rasterio
2   import numpy as np
3   from scipy.stats import rankdata
4
5   # Load raster
6   with rasterio.open('C:/Users/koengel/Documents/dikes/data/data_level0/
        Rhein_Hessen/divided_dem/relativ_heights.tif') as src:
7       raster = src.read(1)  # Read the first band into a 2D array
8       raster = np.clip(raster, 0, 25)  # Apply clipping
9
10  # Handle NA values by creating a mask
11  mask = np.isnan(raster)
12
13  # Perform rank transformation on non-NA values
14  non_na_raster = raster[~mask]
15  ranked = rankdata(non_na_raster, method='average')  # Rank non-NA values
```

```python
ranked_norm = (ranked - 1) / (len(ranked) - 1)  # Normalize ranks between 0 and 1

# Scale to 0-255
scaled = (ranked_norm * 255)

# Initialize normalized_raster with NaN values
normalized_raster = np.full(raster.shape, np.nan, dtype=np.float32)  # Use float32 to support NaN
normalized_raster[~mask] = scaled

# Directly use normalized_raster for saving, maintaining its float dtype
# Adjusted to use float32 for saving to support NaNs properly
with rasterio.open(
    'C:/Users/koengel/Documents/dikes/data/data_level0/Rhein_Hessen/relative_heights_norm.tif',
    'w',
    driver='GTiff',
    height=raster.shape[0],
    width=raster.shape[1],
    count=1,
    dtype=rasterio.float32,  # Use float32 to save, supporting NaNs
    crs=src.crs,
    transform=src.transform,
) as dst:
    dst.write_band(1, normalized_raster.astype(rasterio.float32))
```

Listing 2: Python code for rank based normalization

R Code for Data Manipulation and Model Training

Functions

Function to subset raster into tiles for U-net

```r
subset_ds <- function(
    input_raster,
    model_input_shape,
    path,
    targetname = "") {
  targetSizeX <- model_input_shape[1]
  targetSizeY <- model_input_shape[2]
  inputX <- terra::ncol(input_raster)
  inputY <- terra::nrow(input_raster)

  # difference of input and target size
  diffX <- inputX %% targetSizeX
  diffY <- inputY %% targetSizeY

  # determine new dimensions of raster and crop,
  # cutting evenly on all sides if possible
  newXmin <- terra::ext(input_raster)[1] + ceiling(diffX / 2) * terra::res(input_raster)[1]
  newXmax <- terra::ext(input_raster)[2] - floor(diffX / 2) * terra::res(input_raster)[1]
  newYmin <- terra::ext(input_raster)[3] + ceiling(diffY / 2) * terra::res(input_raster)[2]
```

```
  newYmax <- terra::ext(input_raster)[4] - floor(diffY / 2) * terra::res(
    input_raster)[2]
  rst_cropped <- crop(
    input_raster,
    terra::ext(newXmin, newXmax, newYmin, newYmax)
  )

  # grid for splitting
  agg <- terra::aggregate(
    rst_cropped[[1]],
    c(targetSizeX, targetSizeY)
  )
  agg[] <- 1:ncell(agg)
  agg_poly <- terra::as.polygons(agg)
  names(agg_poly) <- "polis"

  lapply(seq_along(agg_poly), FUN = function(i) {
    subs <- local({
      e1 <- terra::ext(agg_poly[agg_poly$polis == i, ])
      subs <- terra::crop(rst_cropped, e1)
    })
    min_max <- minmax(subs)
    all_na_min <- all(is.na(min_max['min',]))
    all_na_max <- all(is.na(min_max['max',]))
    if ((all_na_min && all_na_max)) { # If not all values are NA (i.e.,
    there's at least one non-NA value)
    }else{
      writeRaster(subs, filename=file.path(path, paste0(targetname, i, ".
    png")), filetype="PNG", overwrite=TRUE)
    }
  })

  return(rst_cropped)
}
```

Listing 3: Function to subset raster into tiles for U-net

Function for dataset preparation and spectral augmentation

```
spectral_augmentation <- function(img) {
  img <- tf$image$random_brightness(img, max_delta = 0.1)
  img <- tf$image$random_contrast(img, lower = 0.9, upper = 1.1)
  img <- tf$image$random_saturation(img, lower = 0.9, upper = 1.1)

  # make sure we still are between 0 and 1
  img <- tf$clip_by_value(img, 0, 1)
}

prepare_ds <- function(files = NULL, # Pfade zu den Masken und DOPS's
                       train, # trainingsdaten oder nicht
                       predict = FALSE, # fuer prediction der testdaten
                       subsets_path = NULL,
                       batch_size = batch_size) { # wie viele Bilder sollen
    auf einmal verarbeitet werden
  if (!predict) {
    # create a tf_dataset from the input data.frame
    # right now still containing only paths to images
    dataset <- tfdatasets::tensor_slices_dataset(files)

```

```
    # use tfdatasets::dataset_map to apply function on each record of the
    dataset
    # (each record being a list with two items: img and mask), the
    # function is purrr::list_modify, which modifies the list items
    # 'img' and 'mask' by using the results of applying decode_png on the
    img and the mask
    # -> i.e. pngs are loaded and placed where the paths to the files were
    (for each record in dataset)
    dataset <-
      tfdatasets::dataset_map(dataset, function(.x) {
        purrr::list_modify(
          .x,
          img = tf$image$decode_png(tf$io$read_file(.x$img)),
          mask = tf$image$decode_png(tf$io$read_file(.x$mask))
        )
      })

    # convert to float32:
    # for each record in dataset, both its list items are modified
    # by the result of applying convert_image_dtype to them
    dataset <-
      tfdatasets::dataset_map(dataset, function(.x) {
        purrr::list_modify(
          .x,
          img = tf$image$convert_image_dtype(.x$img, dtype = tf$float32),
          mask = tf$image$convert_image_dtype(.x$mask, dtype = tf$float32)
        )
      })

    # data augmentation performed on training set only
    if (train) {
      # augmentation 1: flip left right, including random change of
      # saturation, brightness and contrast

      # for each record in dataset, only the img item is modified by the
    result
      # of applying spectral_augmentation to it
      augmentation <-
        tfdatasets::dataset_map(dataset, function(.x) {
          purrr::list_modify(.x, img = spectral_augmentation(.x$img))
        })

      # ...as opposed to this, flipping is applied to img and mask of each
    record
      augmentation <-
        tfdatasets::dataset_map(augmentation, function(.x) {
          purrr::list_modify(
            .x,
            img = tf$image$flip_left_right(.x$img),
            mask = tf$image$flip_left_right(.x$mask)
          )
        })

      dataset_augmented <-
        tfdatasets::dataset_concatenate(augmentation, dataset)

      # augmentation 2: flip up down,
      # including random change of saturation, brightness and contrast
```

```
    augmentation <-
      tfdatasets::dataset_map(dataset, function(.x) {
        purrr::list_modify(.x, img = spectral_augmentation(.x$img))
      })

    augmentation <-
      tfdatasets::dataset_map(augmentation, function(.x) {
        purrr::list_modify(
          .x,
          img = tf$image$flip_up_down(.x$img),
          mask = tf$image$flip_up_down(.x$mask)
        )
      })

    dataset_augmented <-
      tfdatasets::dataset_concatenate(augmentation, dataset_augmented)

    # augmentation 3: flip left right AND up down,
    # including random change of saturation, brightness and contrast

    augmentation <-
      tfdatasets::dataset_map(dataset, function(.x) {
        purrr::list_modify(.x, img = spectral_augmentation(.x$img))
      })

    augmentation <-
      tfdatasets::dataset_map(augmentation, function(.x) {
        purrr::list_modify(
          .x,
          img = tf$image$flip_left_right(.x$img),
          mask = tf$image$flip_left_right(.x$mask)
        )
      })

    augmentation <-
      tfdatasets::dataset_map(augmentation, function(.x) {
        purrr::list_modify(
          .x,
          img = tf$image$flip_up_down(.x$img),
          mask = tf$image$flip_up_down(.x$mask)
        )
      })

    dataset_augmented <-
      tfdatasets::dataset_concatenate(augmentation, dataset_augmented)

    # shuffling
    dataset <-
      tfdatasets::dataset_shuffle(dataset_augmented, buffer_size =
  batch_size * 256)
  }

  # train in batches; batch size might need to be adapted depending on
  # available memory
  dataset <- tfdatasets::dataset_batch(dataset, batch_size)

  # output needs to be unnamed
  dataset <- tfdatasets::dataset_map(dataset, unname)
```

```
  } else {
    # make sure subsets are read in in correct order
    # so that they can later be reassembled correctly
    # needs files to be named accordingly (only number)
    o <-
      order(as.numeric(tools::file_path_sans_ext(basename(
        list.files(subsets_path)
      ))))
    subset_list <- list.files(subsets_path, full.names = T)[o]

    dataset <- tfdatasets::tensor_slices_dataset(subset_list)

    dataset <-
      tfdatasets::dataset_map(dataset, function(.x) {
        tf$image$decode_png(tf$io$read_file(.x))
      })

    dataset <-
      tfdatasets::dataset_map(dataset, function(.x) {
        tf$image$convert_image_dtype(.x, dtype = tf$float32)
      })

    dataset <- tfdatasets::dataset_batch(dataset, batch_size)
    dataset <- tfdatasets::dataset_map(dataset, unname)
  }
}
```

Listing 4: Function for dataset preparation and spectral augmentation

Function to build U-Net

```
# U-Net
# function to build a U-Net
# of course it is possible to change the input_shape
# definiert Architektur des U-net
get_unet_128 <- function(input_shape = c(128, 128, 3),
                         num_classes = 1) {
  inputs <- keras::layer_input(shape = input_shape)
  # 128

  down1 <- inputs %>%
    layer_conv_2d(
      filters = 64,
      kernel_size = c(3, 3),
      padding = "same"
    ) %>%
    layer_activation("relu") %>%
    layer_conv_2d(
      filters = 64,
      kernel_size = c(3, 3),
      padding = "same"
    ) %>%
    layer_activation("relu")
  down1_pool <- down1 %>%
    layer_max_pooling_2d(pool_size = c(2, 2), strides = c(2, 2))
  # 64

  down2 <- down1_pool %>%
    layer_conv_2d(
```

```
      filters = 128,
      kernel_size = c(3, 3),
      padding = "same"
    ) %>%
    layer_activation("relu") %>%
    layer_conv_2d(
      filters = 128,
      kernel_size = c(3, 3),
      padding = "same"
    ) %>%
    layer_activation("relu")
  down2_pool <- down2 %>%
    layer_max_pooling_2d(pool_size = c(2, 2), strides = c(2, 2))
  # 32

  down3 <- down2_pool %>%
    layer_conv_2d(
      filters = 256,
      kernel_size = c(3, 3),
      padding = "same"
    ) %>%
    layer_activation("relu") %>%
    layer_conv_2d(
      filters = 256,
      kernel_size = c(3, 3),
      padding = "same"
    ) %>%
    layer_activation("relu")
  down3_pool <- down3 %>%
    layer_max_pooling_2d(pool_size = c(2, 2), strides = c(2, 2))
  # 16

  down4 <- down3_pool %>%
    layer_conv_2d(
      filters = 512,
      kernel_size = c(3, 3),
      padding = "same"
    ) %>%
    layer_activation("relu") %>%
    layer_conv_2d(
      filters = 512,
      kernel_size = c(3, 3),
      padding = "same"
    ) %>%
    layer_activation("relu")
  down4_pool <- down4 %>%
    layer_max_pooling_2d(pool_size = c(2, 2), strides = c(2, 2))
  # 8

  center <- down4_pool %>%
    layer_conv_2d(
      filters = 1024,
      kernel_size = c(3, 3),
      padding = "same"
    ) %>%
    layer_activation("relu") %>%
    layer_conv_2d(
      filters = 1024,
```

```
      kernel_size = c(3, 3),
      padding = "same"
    ) %>%
    layer_activation("relu")
  # center

  up4 <- center %>%
    layer_upsampling_2d(size = c(2, 2)) %>%
    {
      layer_concatenate(inputs = list(down4, .), axis = 3)
    } %>%
    layer_conv_2d(
      filters = 512,
      kernel_size = c(3, 3),
      padding = "same"
    ) %>%
    layer_activation("relu") %>%
    layer_conv_2d(
      filters = 512,
      kernel_size = c(3, 3),
      padding = "same"
    ) %>%
    layer_activation("relu") %>%
    layer_conv_2d(
      filters = 512,
      kernel_size = c(3, 3),
      padding = "same"
    ) %>%
    layer_activation("relu")
  # 16

  up3 <- up4 %>%
    layer_upsampling_2d(size = c(2, 2)) %>%
    {
      layer_concatenate(inputs = list(down3, .), axis = 3)
    } %>%
    layer_conv_2d(
      filters = 256,
      kernel_size = c(3, 3),
      padding = "same"
    ) %>%
    layer_activation("relu") %>%
    layer_conv_2d(
      filters = 256,
      kernel_size = c(3, 3),
      padding = "same"
    ) %>%
    layer_activation("relu") %>%
    layer_conv_2d(
      filters = 256,
      kernel_size = c(3, 3),
      padding = "same"
    ) %>%
    layer_activation("relu")
  # 32

  up2 <- up3 %>%
    layer_upsampling_2d(size = c(2, 2)) %>%
```

```
    {
      layer_concatenate(inputs = list(down2, .), axis = 3)
    } %>%
    layer_conv_2d(
      filters = 128,
      kernel_size = c(3, 3),
      padding = "same"
    ) %>%
    layer_activation("relu") %>%
    layer_conv_2d(
      filters = 128,
      kernel_size = c(3, 3),
      padding = "same"
    ) %>%
    layer_activation("relu") %>%
    layer_conv_2d(
      filters = 128,
      kernel_size = c(3, 3),
      padding = "same"
    ) %>%
    layer_activation("relu")
  # 64

  up1 <- up2 %>%
    layer_upsampling_2d(size = c(2, 2)) %>%
    {
      layer_concatenate(inputs = list(down1, .), axis = 3)
    } %>%
    layer_conv_2d(
      filters = 64,
      kernel_size = c(3, 3),
      padding = "same"
    ) %>%
    layer_activation("relu") %>%
    layer_conv_2d(
      filters = 64,
      kernel_size = c(3, 3),
      padding = "same"
    ) %>%
    layer_activation("relu") %>%
    layer_conv_2d(
      filters = 64,
      kernel_size = c(3, 3),
      padding = "same"
    ) %>%
    layer_activation("relu")
  # 128

  classify <- layer_conv_2d(
    up1,
    filters = num_classes,
    kernel_size = c(1, 1),
    activation = "sigmoid"
  )

  model <- keras_model(
    inputs = inputs,
    outputs = classify
```

```
  )

  return(model)
}
```

Listing 5: Function to build U-Net

Function to rebuild raster out of tiles

```
# Define a function to rebuild a raster image after processing in a U-Net model
rebuild_img <- function(pred_subsets, out_path, target_rst, model_name, dop_all_tiles) {
  # Calculate the number of columns in each subset (tile)
  subset_pixels_x <- ncol(pred_subsets[1, , ,])
  # Calculate the number of rows in each subset (tile)
  subset_pixels_y <- nrow(pred_subsets[1, , ,])
  # Determine the number of tile rows in the original raster
  tiles_rows <- as.integer(terra::nrow(target_rst) / subset_pixels_y)
  # Determine the number of tile columns in the original raster
  tiles_cols <- as.integer(terra::ncol(target_rst) / subset_pixels_x)

  # Convert the original raster to a 'stars' object for dimension information
  target_stars <- stars::st_as_stars(target_rst, proxy = F)

  # Create a folder for storing the output, using the model name
  result_folder <- file.path(out_path, model_name)
  # If the folder already exists, delete it
  if (dir.exists(result_folder)) {
    unlink(result_folder, recursive = T)
  }
  # Create the output folder
  dir.create(path = result_folder)

  # Loop over each tile, reconstruct and save as a 'stars' object in TIFF format
  for (crow in 1:tiles_rows) {
    for (ccol in 1:tiles_cols) {
      # Calculate the index of the current tile
      i <- (crow - 1) * tiles_cols + (ccol - 1) + 1
      print(i)
      if(i %in% dop_all_tiles){
        # Define the x-dimensions of the current tile
        dimx <- c(((ccol - 1) * subset_pixels_x + 1), (ccol * subset_pixels_x))
        # Define the y-dimensions of the current tile
        dimy <- c(((crow - 1) * subset_pixels_y + 1), (crow * subset_pixels_y))
        # Convert the subset to a 'stars' object and adjust the delta attribute
        cstars <- stars::st_as_stars(t(pred_subsets[which(dop_all_tiles %in% i) , , , 1]))
        attr(cstars, "dimensions")[[2]]$delta <- -1

        # Assign dimensions to the 'stars' object based on the original raster
        st_dimensions(cstars) <- stars::st_dimensions(target_stars[, dimx[1]:dimx[2], dimy[1]:dimy[2]])[1:2]
```

```
41
42          # Write the current tile as a TIFF file
43          stars::write_stars(cstars, dsn = paste0(result_folder, "/_out_", i,
        ".tif"))
44        }
45      }
46    }
47
48    # List all TIFF files in the output folder
49    starstiles <- as.vector(list.files(result_folder, full.names = T), mode =
        "character")
50
51    # Use GDAL to build a VRT (Virtual Raster Tile) from the individual TIFF
        files
52    sf::gdal_utils(util = "buildvrt", source = starstiles, destination = file
        .path(result_folder, "mosaic.vrt"))
53
54    # Use GDAL to warp the VRT into a single mosaic TIFF file
55    sf::gdal_utils(util = "warp", source = file.path(result_folder, "mosaic.
        vrt"), destination = file.path(result_folder, "mosaic.tif"))
56  }
```

Listing 6: Function to rebuild raster out of tiles

Setting up working environment

```
1  library(envimaR)
2  #new_library_path <- "C:/Users/dogbert/Documents/R/win-library/4.1"
3  #.libPaths(c(new_library_path, .libPaths()))
4  packagesToLoad <- c(
5    "terra",
6    "png",
7    "tensorflow",
8    "keras",
9    "reticulate",
10   "sf",
11   "rsample",
12   "tfdatasets",
13   "purrr",
14   "stars",
15   "magick",
16   "mapview",
17   "leafsync",
18   "viridis",
19   "proj4",
20   "raster",
21   "imager",
22   "parallel",
23   "dismo",
24   "doParallel",
25   "raster"
26 )
27
28 # define a project root folder
29 rootDir <- "D:/edu/dikes"
30
31 # some new paths
```

```
projectDirList <- c(
  "data/",
  "data/data_level0/",
  "data/data_level1/",
  "data/modelling/",
  "data/modelling/model_training_data/",
  "data/modelling/model_training_data/dop/",
  "data/modelling/model_training_data/bui/",
  "data/modelling/models/",
  "data/modelling/prediction/",
  "data/modelling/validation/",
  "data/modelling/model_testing_data/",
  "data/modelling/model_testing_data/dop/",
  "data/modelling/model_testing_data/bui/",
  "data/small/",
  "data/small/modelling/model_training_data/",
  "data/small/modelling/model_testing_data/",
  "data/small/modelling/model_training_data/dop/",
  "data/small/modelling/model_training_data/bui/",
  "data/small/modelling/model_testing_data/dop/",
  "data/small/modelling/model_testing_data/bui/",
  "data/small/modelling/models/",
  "data/small/modelling/prediction/",
  "data/small/modelling/full_prediction/",
  "data/small/modelling/validation/",
  "data/dem_stretch/",
  "data/dem_stretch/modelling/model_training_data/",
  "data/dem_stretch/modelling/model_testing_data/",
  "data/dem_stretch/modelling/model_training_data/dop/",
  "data/dem_stretch/modelling/model_training_data/bui/",
  "data/dem_stretch/modelling/model_testing_data/dop/",
  "data/dem_stretch/modelling/model_testing_data/bui/",
  "data/dem_stretch/modelling/models/",
  "data/dem_stretch/modelling/prediction/",
  "data/dem_stretch/modelling/validation/",
  "data/demWater/",
  "data/demWater/modelling/model_training_data/",
  "data/demWater/modelling/model_testing_data/",
  "data/demWater/modelling/model_training_data/dop/",
  "data/demWater/modelling/model_training_data/bui/",
  "data/demWater/modelling/model_testing_data/dop/",
  "data/demWater/modelling/model_testing_data/bui/",
  "data/demWater/modelling/models/",
  "data/demWater/modelling/prediction/",
  "data/demWater/modelling/validation/",
  "data/double/",
  "data/double/modelling/model_training_data/",
  "data/double/modelling/model_testing_data/",
  "data/double/modelling/model_training_data/dop/",
  "data/double/modelling/model_training_data/bui/",
  "data/double/modelling/model_testing_data/dop/",
  "data/double/modelling/model_testing_data/bui/",
  "data/double/modelling/models/",
  "data/double/modelling/prediction/",
  "data/double/modelling/full_prediction/",
  "data/double/modelling/validation/",
  "data/final/",
  "data/final/modelling/model_training_data/",
```

```
  "data/final/modelling/model_testing_data/",
  "data/final/modelling/model_training_data/dop/",
  "data/final/modelling/model_training_data/bui/",
  "data/final/modelling/model_testing_data/dop/",
  "data/final/modelling/model_testing_data/bui/",
  "data/final/modelling/models/",
  "data/final/modelling/prediction/",
  "data/final/modelling/full_prediction/",
  "data/final/modelling/validation/",
  "data/epoch_10/",
  "data/epoch_10/modelling/model_training_data/",
  "data/epoch_10/modelling/model_testing_data/",
  "data/epoch_10/modelling/model_training_data/dop/",
  "data/epoch_10/modelling/model_training_data/bui/",
  "data/epoch_10/modelling/model_testing_data/dop/",
  "data/epoch_10/modelling/model_testing_data/bui/",
  "data/epoch_10/modelling/models/",
  "data/epoch_10/modelling/prediction/",
  "data/epoch_10/modelling/full_prediction/",
  "data/epoch_10/modelling/validation/",
  "data/epoch_12/",
  "data/epoch_12/modelling/model_training_data/",
  "data/epoch_12/modelling/model_testing_data/",
  "data/epoch_12/modelling/model_training_data/dop/",
  "data/epoch_12/modelling/model_training_data/bui/",
  "data/epoch_12/modelling/model_testing_data/dop/",
  "data/epoch_12/modelling/model_testing_data/bui/",
  "data/epoch_12/modelling/models/",
  "data/epoch_12/modelling/prediction/",
  "data/epoch_12/modelling/full_prediction/",
  "data/epoch_12/modelling/validation/",
  "data/demWaterHQ100/",
  "data/demWaterHQ100/modelling/model_training_data/",
  "data/demWaterHQ100/modelling/model_testing_data/",
  "data/demWaterHQ100/modelling/model_training_data/dop/",
  "data/demWaterHQ100/modelling/model_training_data/bui/",
  "data/demWaterHQ100/modelling/model_testing_data/dop/",
  "data/demWaterHQ100/modelling/model_testing_data/bui/",
  "data/demWaterHQ100/modelling/models/",
  "data/demWaterHQ100/modelling/prediction/",
  "data/demWaterHQ100/modelling/full_prediction/",
  "data/demWaterHQ100/modelling/validation/",
  "docs/",
  "run/",
  "tmp",
  "src/",
  "src/functions/"
)

# Now set automatically root direcory, folder structure and load libraries
envrmt <- envimaR::createEnvi(
  root_folder = rootDir,
  folders = projectDirList,
  path_prefix = "path_",
  libs = packagesToLoad,
  alt_env_id = "COMPUTERNAME",
  alt_env_value = "PCRZP",
  alt_env_root_folder = "F:/BEN/edu"
```

```
)

## set terra temp path
terra::terraOptions(tempdir = envrmt$path_tmp)
```

Listing 7: Setting up working environment

Layer preparation

```
source("D:/edu/dikes/src/dikes_setup.R")

# Merging TIFF files into one DEM
tiff_files <- list.files(path = "path_to_tiff_folder", pattern = "\\.tif$",
    full.names = TRUE)
dem <- merge(rast(tiff_files))

# Assuming you have a shapefile for the river
river <- vect("path_to_river_shapefile.shp")

# Buffering the river by 200 meters
river_buffer <- buffer(river, width = 1500)

# Mask DEM outside the 200m river buffer
dem_masked <- mask(dem, river_buffer)

# Create a distance layer to the nearest river
distance_to_river <- distance(dem, river, filename="distance_to_river.tif",
    overwrite=TRUE)

# Save the processed DEM
writeRaster(dem_masked, "merged_dem.tif", format="GTiff", overwrite=TRUE)
writeRaster(distance_to_river, "distance_to_river.tif", format="GTiff",
    overwrite=TRUE)
```

Listing 8: Input layer prepartion

Create masks and tiles

```
source("D:/edu/dikes/src/dikes_setup.R")

ras <- terra::rast(file.path(envrmt$path_data_level0, "/Rhein_Hessen/
    Rhein_stack_final_new.tif"))
ras[[3]] <- ras[[1]]

# download OSM building data
dikes <- vect(file.path(envrmt$path_data_level0, "trainingsdata.gpkg"))

polygon <- vect(file.path(envrmt$path_data_level0, "full_research_area/
    research_area.shp"))
ras <- crop(ras, ext(polygon))
#ras[[1]][!is.na(ras[[2]])] <- 0
# crop OSM building data to the extent of the raster data
dikes <- crop(dikes, ext(ras))

# rasterize the buildings
```

```r
rasterized_vector <- terra::rasterize(dikes, ras[[1]], field=255,
    background = 0)
rasterized_vector[is.na(ras[[2]])] <- NA
plot(rasterized_vector)
# save
terra::writeRaster(rasterized_vector,
                   file.path(envrmt$path_data_level1, "mask.tif"),
                   overwrite = T
)

# Assuming 'ras' is your raster object
ext_ras <- terra::ext(ras)  # Store raster extent once
res_ras_x <- res(ras)[1]  # Assuming uniform resolution, get x resolution

# Calculate xmax for testing extent based on 20% of raster width from left
xmax_test <- ext_ras[1] + round(ncol(ras) * 0.2) * res_ras_x

# Define testing extent (left 20%)
e_test <- terra::ext(ext_ras[1], xmax_test, ext_ras[3], ext_ras[4])

# Define training extent (remaining 80%)
e_train <- terra::ext(xmax_test, ext_ras[2], ext_ras[3], ext_ras[4])


# crop files
biblis_mask_train <- terra::crop(rasterized_vector, e_train)
biblis_dop_train <- terra::crop(ras, e_train)
biblis_mask_test <- terra::crop(rasterized_vector, e_test)
biblis_dop_test <- terra::crop(ras, e_test)

# save files
terra::writeRaster(
  biblis_mask_test,
  file.path(envrmt$path_epoch_10_modelling_model_testing_data, "mask_test.
    tif"),
  overwrite = T
)
terra::writeRaster(
  biblis_dop_test,
  file.path(envrmt$path_epoch_10_modelling_model_testing_data, "dop_test.
    tif"),
  overwrite = T
)
terra::writeRaster(
  biblis_mask_train,
  file.path(envrmt$path_epoch_10_modelling_model_training_data, "mask_train
    .tif"),
  overwrite = T
)
terra::writeRaster(
  biblis_dop_train,
  file.path(envrmt$path_epoch_10_modelling_model_training_data, "dop_train.
    tif"),
  overwrite = T
)

source(paste0(envrmt$path_functions, "/subset_ds.R"))

```

```r
# read training data
biblis_mask_train <-
  terra::rast(file.path(envrmt$path_epoch_10_modelling_model_training_data,
    "mask_train.tif"))
biblis_dop_train <-
  terra::rast(file.path(envrmt$path_epoch_10_modelling_model_training_data,
    "dop_train.tif"))

# set the size of each image
model_input_shape <- c(128,128)


# subsets for the dop
subset_ds(
  input_raster = biblis_dop_train,
  model_input_shape = model_input_shape,
  path = envrmt$path_epoch_10_modelling_model_training_data_dop
)

# subsets for the mask
subset_ds(
  input_raster = biblis_mask_train,
  model_input_shape = model_input_shape,
  path = envrmt$path_epoch_10_modelling_model_training_data_bui
)
```

Listing 9: Create masks and and subset into tiles

Spectral augmentation and U-Net training

```r
source("D:/edu/dikes/src/dikes_setup.R")


gpu_options <- tf$compat$v1$GPUOptions(
  per_process_gpu_memory_fraction = 0.95,  # Limits the GPU usage to 75% of
    total GPU memory
  allow_growth = TRUE                      # Allows dynamic GPU memory
   allocation
)

config <- tf$compat$v1$ConfigProto(gpu_options = gpu_options)

# Create and set the session with these GPU options
sess <- tf$compat$v1$Session(config = config)
tf$compat$v1$keras$backend$set_session(sess)


files <- data.frame(
  img = list.files(
    envrmt$path_epoch_10_modelling_model_training_data_dop,
    full.names = TRUE,
    pattern = "*.png"
  ),
  mask = list.files(
    envrmt$path_epoch_10_modelling_model_training_data_bui,
    full.names = TRUE,
    pattern = "*.png"
```

```
  )
)
# split randomly into training and validation (not testing!!) data sets
set.seed(7)
data <- rsample::initial_split(files, prop = 0.8)

source(paste0(envrmt$path_functions, "/prepare_ds_spectral_augmentation.R")
    )

batch_size <- 16

# prepare data for training
training_dataset <- prepare_ds(
  training(data),
  train = TRUE,
  predict = FALSE,
  batch_size = batch_size
)

# also prepare validation data
validation_dataset <- prepare_ds(
  testing(data),
  train = FALSE,
  predict = FALSE,
  batch_size = batch_size
)

source(paste0(envrmt$path_functions, "/get_unet_128.R"))
unet_model <- get_unet_128()

# compile the model
unet_model %>% compile(
  optimizer = optimizer_adam(learning_rate = 0.0001),
  loss = "binary_crossentropy",
  metrics = "accuracy"
)

# train the model
hist <- unet_model %>% fit(
  training_dataset,
  validation_data = validation_dataset,
  epochs = 17,
  verbose = 1
)

# save the model
unet_model %>% save_model_hdf5(file.path(
    envrmt$path_epoch_10_modelling_models, "unet_dikes.hdf5"))

# plot training history
#plot(hist)

# save training history
path <- file.path(envrmt$path_epoch_12_modelling_models, "hist_biblis.RDS")
saveRDS(hist, file = path)
test <- readRDS(path)
metrics <- test[[2]]

```

```r
# Creating a dataframe from the metrics list
metrics_df <- data.frame(
  loss = metrics$loss,
  accuracy = metrics$accuracy,
  val_loss = metrics$val_loss,
  val_accuracy = metrics$val_accuracy
)


library(ggplot2)
library(gridExtra)

# Plot for Loss (Training and Validation)
plot_loss <- ggplot(metrics_df, aes(x = 1:nrow(metrics_df))) +
  geom_line(aes(y = loss, colour = "Training Loss")) +
  geom_line(aes(y = val_loss, colour = "Validation Loss")) +
  labs(title = "Training and Validation Loss per Epoch",
       x = "Epoch",
       y = "Loss") +
  theme_minimal() +
  scale_colour_manual(values = c("Training Loss" = "blue", "Validation Loss" = "red"))+
  guides(colour = guide_legend(title = NULL))  # Remove legend title


# Plot for Accuracy (Training and Validation)
plot_accuracy <- ggplot(metrics_df, aes(x = 1:nrow(metrics_df))) +
  geom_line(aes(y = accuracy, colour = "Training Accuracy")) +
  geom_line(aes(y = val_accuracy, colour = "Validation Accuracy")) +
  labs(title = "Training and Validation Accuracy per Epoch",
       x = "Epoch",
       y = "Accuracy") +
  theme_minimal() +
  scale_colour_manual(values = c("Training Accuracy" = "blue", "Validation Accuracy" = "red"))+
  guides(colour = guide_legend(title = NULL))  # Remove legend title


# Arrange plots vertically
grid.arrange(plot_loss, plot_accuracy, ncol = 1)
```

Listing 10: Spectral augmentation and U-Net training

Model testing

```r
source("D:/edu/dikes/src/dikes_setup.R")
source(paste0(envrmt$path_functions, "/subset_ds.R"))
source(paste0(envrmt$path_functions, "/prepare_ds_spectral_augmentation.R"))

# load the test data
biblis_mask_test <-
  terra::rast(file.path(envrmt$path_epoch_10_modelling_model_testing_data, "mask_test.tif"))
biblis_dop_test <-
  terra::rast(file.path(envrmt$path_epoch_10_modelling_model_testing_data, "dop_test.tif"))
```

```
10
11 model_input_shape <- c(128, 128)
12 batch_size <- 16
13
14 target_rst <- subset_ds(
15   input_raster = biblis_mask_test,
16   model_input_shape = model_input_shape,
17   path = envrmt$path_epoch_10_modelling_model_testing_data_bui
18 )
19
20 subset_ds(
21   input_raster = biblis_dop_test,
22   model_input_shape = model_input_shape,
23   path = envrmt$path_epoch_10_modelling_model_testing_data_dop
24 )
25
26
27 list_dops_names <-
28   list.files(envrmt$path_epoch_10_modelling_model_testing_data_dop,
29              full.names = FALSE,
30              pattern = "*.png"
31   )
32
33 list_masks_names <-
34   list.files(envrmt$path_epoch_10_modelling_model_testing_data_bui,
35              full.names = FALSE,
36              pattern = "*.png"
37   )
38
39 # write the target_rst to later rebuild your image
40 terra::writeRaster(
41   target_rst,
42   file.path(envrmt$path_epoch_10_modelling_model_testing_data, "
     mask_test_target.tif"),
43   overwrite = T
44 )
45
46 test_file <- data.frame(
47   img = list.files(
48     envrmt$path_epoch_10_modelling_model_testing_data_dop,
49     full.names = T,
50     pattern = "*.png"
51   ),
52   mask = list.files(
53     envrmt$path_epoch_10_modelling_model_testing_data_bui,
54     full.names = T,
55     pattern = "*.png"
56   )
57 )
58
59 testing_dataset <- prepare_ds(
60   test_file,
61   train = FALSE,
62   predict = FALSE,
63   batch_size = batch_size
64 )
65
66 # load a U-Net
```

```
unet_model <- keras::load_model_hdf5(
  file.path(envrmt$path_epoch_10_modelling_models, "unet_dikes.hdf5"),
  compile = TRUE
)

# evaluate the model with test set
ev <- unet_model$evaluate(testing_dataset)

# prepare data for prediction
prediction_dataset <- prepare_ds(
  predict = TRUE,
  subsets_path = envrmt$path_epoch_10_modelling_model_testing_data_dop,
  batch_size = batch_size
)

# get sample of data from testing data
#t_sample <- floor(runif(n = 5, min = 1, max = nrow(test_file)))

# simple visual comparison of mask, image and prediction
#for (i in t_sample) {
#  png_path <- test_file
#  png_path <- png_path[i, ]
#
#  img <- magick::image_read(png_path[, 1])
#  mask <- magick::image_read(png_path[, 2])
#  pred <- magick::image_read(
#    terra::as.raster(predict(object = unet_model, testing_dataset)[i, , , ])
#  )
#
#  out <- magick::image_append(c(
#    magick::image_annotate(
#      mask,
#      "Mask",
#      size = 10,
#      color = "black",
#      boxcolor = "white"
#    ),
#    magick::image_annotate(
#      img,
#      "Original Image",
#      size = 10,
#      color = "black",
#      boxcolor = "white"
#    ),
#    magick::image_annotate(
#      pred,
#      "Prediction",
#      size = 10,
#      color = "black",
#      boxcolor = "white"
#    )
#  ))
#
#  plot(out)
#}

target_rst <- terra::rast(file.path(
```

```
    envrmt$path_epoch_10_modelling_model_testing_data, "mask_test_target.tif
    "))

# make the actual prediction
pred_subsets <- predict(object = unet_model, x = prediction_dataset)

model_name <- "unet_abc"

# rebuild .tif from each patch
files <- list.files(envrmt$path_epoch_10_modelling_model_testing_data_dop)
dop_all_tiles <- sort(as.numeric(gsub("[^0-9]", "", files)))
source(paste0(envrmt$path_functions, "/rebuild_img.R"))
rebuild_img(
  pred_subsets = pred_subsets,
  out_path = envrmt$path_epoch_10_modelling_prediction,
  target_rst = target_rst,
  model_name = model_name,
  dop_all_tiles = dop_all_tiles
)
```

Listing 11: Model testing

Model on whole Rhine

```
ource("D:/edu/dikes/src/dikes_setup.R")
source(paste0(envrmt$path_functions, "/subset_ds_mod.R"))
source(paste0(envrmt$path_functions, "/prepare_ds_spectral_augmentation.R")
    )

model_input_shape <- c(128, 128)
batch_size <-16

# Explicitly set TensorFlow to use no GPU
tf$config$set_visible_devices(list(), 'GPU')

# Reinitialize session to apply changes
keras::k_clear_session()
use_session_with_seed(42)  # Optional: Reset with a seed for
    reproducibility

# Load model and proceed with CPU only
unet_model <- keras::load_model_hdf5(
  file.path(envrmt$path_epoch_10_modelling_models, "unet_dikes.hdf5"),
  compile = TRUE
)

# Verify the device configuration
tf$config$experimental$list_physical_devices()

dop_test <-
  terra::rast(file.path(envrmt$path_data_level0, "/Rhein_Hessen/
    Rhein_stack_final_new.tif"))
dop_test[[3]] <- dop_test[[1]]

files <- list.files(envrmt$path_small_modelling_full_prediction)
dop_all_tiles <- sort(as.numeric(gsub("[^0-9]", "", grep("\\.png$", files,
    value = TRUE))))
```

```
30
31 subset_ds(
32   input_raster = dop_test,
33   model_input_shape = model_input_shape,
34   path = envrmt$path_final_modelling_full_prediction,
35   dop_all_tiles = dop_all_tiles
36 )
37
38 target_rst <- subset_ds(
39   input_raster = dop_test,
40   model_input_shape = model_input_shape,
41   path = envrmt$path_final_modelling_full_prediction,
42   dop_all_tiles = dop_all_tiles
43 )
44
45 target_rst <- terra::rast(file.path(envrmt$path_data_level0, "/Rhein_Hessen
     /Rhein_stack_final_new.tif"))
46
47 prediction_dataset <- prepare_ds(
48   predict = TRUE,
49   subsets_path = envrmt$path_small_modelling_full_prediction,
50   batch_size = batch_size
51 )
52
53 pred_subsets <- predict(object = unet_model, x = prediction_dataset)
54
55 target_rst <- target_rst[[1]]
56 model_name <- "unet_abc"
57
58 source(paste0(envrmt$path_functions, "/rebuild_img.R"))
59 # rebuild .tif from each patch
60 rebuild_img(
61   pred_subsets = pred_subsets,
62   out_path = envrmt$path_epoch_10_modelling_full_prediction,
63   target_rst = target_rst,
64   model_name = model_name,
65   dop_all_tiles = dop_all_tiles
66 )
```

Listing 12: Model on whole Rhine area