Heidelberg University

Institute of Geography
Faculty of Chemistry and Earth Sciences

in cooperation with

HeiGIT gGmbH

Master's Thesis

Enhancing LULC Semantic Segmentation Models by Integrating Spatial Contextual Information from OpenStreetMap Road Networks

---

Author:
Nikolaos Kolaxidis

First Examiner:
apl. Prof. Dr. Sven Lautenbach

Second Examiner:
Dr. Christina Ludwig
associated with HeiGIT gGmbH

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Geography
Specialization in Geoinformatics

August 1, 2024

Abstract

Enhancing LULC Semantic Segmentation Models by Integrating Spatial Contextual Information from OpenStreetMap Road Networks

by Nikolaos Kolaxidis

Currently, two topics are prevalent in the geographic science world, namely deep learning and climate change. One key research field at the intersection of both fields is Land Use Land Cover (LULC) mapping, a geographical computer vision task requiring large amounts of remote sensing data and, therefore, suitable deep learning models for segmentation. By utilizing such models, LULC mapping can be done automatically and efficiently, aiding in environmental monitoring as well as sustainable urban planning and resource management. Deep learning models offer two ways of improvement: enhancing the performance or enhancing the efficiency. In regard to climate change, efficiency becomes more important, as the models should be as sustainable as possible without consuming unnecessary resources. However, the reduction of performance for the sake of efficiency is not a desirable trade-off, especially when the according results are used for further applications. Therefore, this study evaluates a concept of supporting the convergence of models for LULC mapping in order to reduce the resource consumption while achieving similar performance. Current semantic segmentation models are able to capture a large context area of pixels to classify them effectively with more information available. Hypothetically, for segmentation tasks like LULC mapping, spatial contextual information has the potential to aid the models in making more informed predictions. As they are an important part of the landscape and are built with the adjacent LULC areas in mind, roads seem suitable to provide such information. Utilizing the SegFormer architecture inside the LULC Utility, a cutting-edge semantic segmentation framework for LULC mapping, this study incorporates features from the OpenStreetMap road network to improve both model performance and resource efficiency. Key research questions focus on identifying relevant road attributes, suitable encoding methods for feature integration, and the impact of these integrations on model performance and resource consumption. The results demonstrate that incorporating road network data, particularly road types and speed limits, considerably enhances the model's performance, while also increasing the resource consumption. However, they also show that with fine-tuning hyperparameters and according optimization techniques, the method has the potential to achieve higher performance while being more efficient, accomplishing the aforementioned goal. The study confirms that the integration of contextual data can be a valuable addition to deep learning models in the field of computer vision, as it offers the potential to enhance both their performance and efficiency.

Zusammenfassung

Enhancing LULC Semantic Segmentation Models by Integrating Spatial Contextual Information from OpenStreetMap Road Networks

von Nikolaos Kolaxidis

In der geographischen Wissenschaftswelt sind heutzutage zwei Themen nicht wegzudenken: Deep Learning und der Klimawandel. Ein zentrales Forschungsfeld an der Schnittstelle beider Themen ist das Mapping von Landnutzung und Landbedeckung (LULC), eine geografische Computer Vision-Aufgabe, die vor allem wegen großer Mengen an Fernerkundungsdaten geeignete Deep Learning Modelle für die Segmentierung erfordert. Durch den Einsatz solcher Modelle kann das LULC-Mapping automatisch und effizient durchgeführt werden, was die Umweltüberwachung sowie die nachhaltige Stadtplanung und Ressourcenverwaltung unterstützt. Deep Learning-Modelle bieten zwei Möglichkeiten zur Verbesserung: die Erhöhung der Performance oder die Verbesserung der Effizienz. In Bezug zum Klimawandel gewinnt die Effizienz immer mehr an Bedeutung, da die Modelle so nachhaltig wie möglich sein sollten, ohne unnötige Ressourcen zu verbrauchen. Allerdings ist eine Reduzierung der Performance zugunsten der Effizienz kein wünschenswerter Kompromiss, insbesondere wenn die Ergebnisse für weitere Anwendungen genutzt werden. Diese Studie evaluiert daher ein Konzept zur Unterstützung der Konvergenz von Modellen für das LULC-Mapping, um den Ressourcenverbrauch zu reduzieren und gleichzeitig eine ähnliche Performance zu erzielen. Aktuelle Modelle zur semantischen Segmentierung sind in der Lage, einen großen Kontextbereich von Pixeln zu erfassen, um sie mit mehr verfügbaren Informationen effektiver zu klassifizieren. Hypothetisch haben räumliche Kontextinformationen für Segmentierungsaufgaben wie das LULC-Mapping das Potenzial, den Modellen zu helfen, fundiertere Vorhersagen zu treffen. Straßen, als wichtiger Teil der Landschaft und in Bezug auf die angrenzenden LULC-Gebiete gebaut, scheinen geeignet, solche Informationen bereitzustellen. Diese Studie nutzt das SegFormer Modell, eingebettet im LULC Utility, einem Framework zum LULC-Mapping, und integriert Features des OpenStreetMap-Straßennetzwerkes, um sowohl die Performance als auch die Ressourceneffizienz des Modells zu verbessern. Die Hauptforschungsfragen konzentrieren sich auf die Identifizierung relevanter Straßeneigenschaften, geeignete Kodierungsmethoden für die Integration von Features und die Auswirkungen der integrierten Kontextinformationen auf die Performance und den Ressourcenverbrauch. Die Ergebnisse zeigen, dass die Einbeziehung von Straßennetzdaten, insbesondere Straßentypen und Geschwindigkeitsbegrenzungen, die Performance des Modells erheblich verbessert, gleichzeitig aber auch den Ressourcenverbrauch erhöht. Sie zeigen allerdings auch, dass mit dem Fine-Tuning der Hyperparameter und entsprechenden Optimierungstechniken die Methode das Potenzial hat, höhere Performance bei höherer Effizienz zu erzielen, und somit das angestrebte Ziel der Kombination von beidem zu erreichen. Die Studie bestätigt, dass die Integration von Kontextdaten eine wertvolle Ergänzung für Deep Learning-Modelle im Bereich des Computer Vision darstellt, da sie das Potenzial bietet, sowohl deren Performance als auch Effizienz zu verbessern.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to *Dr. Christina Ludwig* for the opportunity of conducting this research, her continuous support, guidance, and insightful discussions that have tremendously enriched this work. Your dedication and knowledge have been invaluable to this study and my academic journey in general.

I extend my heartfelt thanks to *apl. Prof. Dr. Sven Lautenbach* for his valuable insights and constructive feedback throughout this research. Your expertise has greatly contributed to the development of this study, including the refinement of the research questions and the overall aim.

Special thanks go to my colleague, *Dr. Maciej Adamiak*, for his assistance, provision of the utilized framework, and inspiration that lay the foundation of this work. Without your limitless inventiveness, creativity, and passion, this thesis would have not been conducted so successfully. Thanks also for enhancing my understanding of the subject matter during the Innovation Weeks and our multiple meetings, this helped me greatly for this thesis.

I am grateful to the *Institute of Geography* and the *HeiGIT gGmbH* for providing the resources and environment necessary for the completion of this thesis.

I thank my *family* for continuous support and encouragement throughout my studies. You always had my back and spanned a boundless safety net, enabling me to focus on my studies. Thanks also go to my *friends* for multiple joyful and insightful discussions and experiences.

Thanks also to you, *Sara*, without you I would not have left my desk to enjoy the sun in the summer, take relaxing walks, and the office and us would miss a very welcome distraction.

Finally, I would like to express my deepest gratitude to *my better half* for unwavering and enduring support, patience, and the best food I could ask for during the long final sprint. Your love and belief in me have been a constant source of motivation and a large driver for my recent development in any aspect.

Thank you all for giving me the opportunity to grow, learn, and follow my dreams. Without you, this would not have been possible.

Statutory declaration

I, Nikolaos Kolaxidis, declare that this thesis titled, “Enhancing LULC Semantic Segmentation Models by Integrating Spatial Contextual Information from OpenStreetMap Road Networks” and the work presented in it are my own. I confirm that:

- I have authored this thesis independently.
- I have not used other than the declared sources / resources.
- I have explicitly marked all material which has been quoted either literally or by content from the used sources.
- this paper was not previously presented to another examination board and has not been published.

Dossenheim, August 1, 2024

Contents

List of Figures viii

List of Tables x

List of Acronyms and Abbreviations xi

1 Introduction 1

2 Theoretical Background 5

2.1 LULC Mapping . . . 5
2.2 Semantic Segmentation . . . 7
2.3 Machine & Deep Learning . . . 8
2.4 Artificial Neural Networks . . . 9
2.4.1 Perceptrons . . . 10
2.4.2 Multi-Layer Perceptrons . . . 12
2.5 Training (Deep) Neural Networks . . . 14
2.5.1 Tensors & Feature Space . . . 14
2.5.2 Loss Functions . . . 15
2.5.3 Gradient Descent . . . 15
2.5.4 Backpropagation . . . 18
2.5.5 Training Loop . . . 18
2.5.6 Regularization & Normalization . . . 20
2.5.7 Computational Demand . . . 22
2.6 Deep Neural Networks for Computer Vision . . . 22
2.6.1 Convolutional Neural Networks . . . 23
2.6.2 Encoder-Decoder Architectures . . . 27
2.6.3 Residual Networks . . . 28
2.7 Spatial Context & Attention in Computer Vision . . . 29
2.7.1 Self-Attention & Scaled Dot-Product Attention . . . 31
2.7.2 Multi-Head (Self-)Attention . . . 33
2.7.3 Positional Encoding . . . 34
2.8 Attention-Based Architectures for Computer Vision . . . 34
2.8.1 Vision Transformer . . . 34
2.8.2 Follow-Ups of the Vision Transformer . . . 35
2.8.3 SegFormer . . . 36
2.9 Summary . . . 40

3 Methodology **42**
3.1 Study Area . . . 42
3.2 Data . . . 44
3.2.1 OpenStreetMap . . . 44
3.2.2 Road Network . . . 46
3.2.3 LULC Classes . . . 50
3.2.4 Remote Sensing Data . . . 51
3.3 Explorative Relationship Analysis Between Roads & LULC . . . 53
3.4 LULC Utility . . . 55
3.4.1 Preparation Modules . . . 56
3.4.2 Training Module & Model Implementation . . . 56
3.4.3 Baseline & Model Parameters . . . 58
3.5 Integrate Road Network into Feature Space . . . 58
3.5.1 Extension 1 - Binary Road Network . . . 62
3.5.2 One-Hot Encoding . . . 62
3.5.3 Extension 2 - Encoded Road Types . . . 63
3.5.4 Extension 3 - Encoded Speed Limits . . . 65
3.5.5 Extension 4 - Ensemble of Extensions 2 and 3 . . . 67
3.6 Machine Configuration & Optimization . . . 67
3.7 Evaluation metrics . . . 69
3.7.1 Performance . . . 69
3.7.2 Resource Consumption & Efficiency . . . 72

4 Results **75**
4.1 Relationship Between Roads & LULC Classes . . . 75
4.1.1 Road Types vs. LULC Classes . . . 75
4.1.2 Speed Limit Classes vs. LULC Classes . . . 81
4.2 Model Evaluation . . . 85
4.2.1 Overview over Resource Consumption & Efficiency . . . 86
4.2.2 Performance . . . 89
4.2.3 Hardware Utilization . . . 92

5 Discussion **95**
5.1 Key Findings . . . 95
5.2 Road Attributes & LULC Classes . . . 96
5.3 Methodology . . . 99
5.4 Objective & Data Redundancy . . . 100

6 Conclusion **102**

Bibliography **104**

A Appendix **115**
A.1 Code Availability & Repositories . . . 115

A.2 Default and Country-Specific Speed Limits 115
A.3 Maps of Mannheim - Heidelberg - Rhein-Neckar-Kreis 116
A.4 Relationship Between Roads and LULC Classes 118
A.4.1 Visual Relationship 118
A.4.2 Buffered Road Types vs. LULC Classes 121
A.4.3 Buffered Speed Limit Classes vs. LULC Classes 125
A.5 Model Evaluation 129
A.5.1 Performance of All Model Runs 129
A.5.2 Confusion Matrices of All Model Runs 131
A.5.3 Training vs. Validation Overall Accuracy 133
A.5.4 Training vs. Validation Intersection over Union 134
A.5.5 Hardware Utilization of All Model Runs 135

List of Figures

2.1 LULC Mapping . . . 6
2.2 Types of Image Segmentation . . . 7
2.3 Perceptron Architecture . . . 10
2.4 Multi-Layer Perceptron Architecture . . . 13
2.5 Gradient Descent . . . 16
2.6 Concept of Over-, Under-, and Optimal Fitting of Neural Networks . . . 20
2.7 Convolutional Neural Network Architecture . . . 23
2.8 Convolution . . . 24
2.9 Padding . . . 25
2.10 Dilation . . . 25
2.11 Pooling Methods . . . 26
2.12 Encoder-Decoder Architecture . . . 27
2.13 Residual Learning Block . . . 29
2.14 Attention Mechanisms . . . 31
2.15 Self-Attention Block in Computer Vision . . . 33
2.16 Vision Transformer Architecture . . . 35
2.17 Effective Receptive Field Analysis of the SegFormer . . . 37
2.18 SegFormer Architecture . . . 38
2.19 Mix Transformer Block of the SegFormer . . . 39

3.1 Overview over the Area of Interest MA-HD-RNK . . . 43
3.2 Road Network of MA-HD-RNK . . . 47
3.3 Research Framework . . . 59
3.4 Pixel-Scale Sentinel-2 RGB Composite . . . 61
3.5 Averaged Dendrogram of Hierarchical Clustering Analysis of Road Types . . . 64
3.6 Concept of Confusion Matrices . . . 70

4.1 Visual Relationship between Select Road Types and LULC Classes . . . 76
4.2 Averaged Heatmaps of Road Types vs. LULC Classes . . . 78
4.3 Clustermap of Road Types vs. LULC Classes . . . 80
4.4 PPMC of LULC Classes Based on Road Types . . . 81
4.5 Averaged Heatmaps of Speed Limit Classes vs. LULC Classes . . . 82
4.6 Clustermap of Speed Limit Classes vs. LULC Classes . . . 84
4.7 PPMC of LULC Classes Based on Speed Limit Classes . . . 85
4.8 Loss Plots of Model Runs . . . 87
4.9 Regression Plots of GPU & CPU Utilization . . . 93
4.10 Regression Plots of RAM & VRAM Utilization . . . 94

A.1 LULC Classes of MA-HD-RNK 116
A.2 Feature Engineered Road Types of MA-HD-RNK 117
A.3 Feature Engineered Speed Limit Classes of MA-HD-RNK 118
A.4 Visual Relationship between Road Types and LULC Classes, Continuation I 119
A.5 Visual Relationship between Road Types and LULC Classes, Continuation II 120
A.6 Heatmaps of Road Types vs. LULC Classes, Vertical Summation 121
A.7 Heatmaps of Road Types vs. LULC Classes, Changes, Vertical Summation 121
A.8 Heatmaps of Road Types vs. LULC Classes, Horizontal Summation 122
A.9 Heatmaps of Road Types vs. LULC Classes, Changes, Horizontal Summation 122
A.11 PPMC of LULC Classes Based on Buffered Road Types 124
A.12 Heatmaps of Speed Limit Classes vs. LULC Classes, Vertical Summation 125
A.13 Heatmaps of Speed Limit Classes vs. LULC Classes, Changes, Vertical Summation . . . 125
A.14 Heatmaps of Speed Limit Classes vs. LULC Classes, Horizontal Summation 126
A.15 Heatmaps of Speed Limit Classes vs. LULC Classes, Changes, Horizontal Summation . 126
A.17 PPMC of LULC Classes Based on Buffered Speed Limit Classes 128
A.19 Confusion Matrices of All Model Runs 132
A.20 Overall Accuracy Plots of All Model Runs 133
A.21 Intersection over Union Plots of All Model Runs 134

List of Tables

2.1 3×3 Kernel 24

3.1 Road Lengths of MA-HD-RNK 49
3.2 Value Counts in the Tag *Maxspeed* for MA-HD-RNK 50
3.3 OSM Filters for the LULC Classes of MA-HD-RNK 51
3.4 Input Data for the Model 52
3.5 Tensor Channels of Extensions 59
3.6 Distribution of Speed Limit Classes for MA-HD-RNK 67
3.7 Machine Configuration - Hardware 68
3.8 Machine Configuration - Software 68
3.9 File Caching Format Comparison 69

4.1 Overview over General Model Run Metrics 88
4.2 Grouped General Model Run Metrics 89
4.3 Grouped Maximal Performance Metrics 90
4.4 Grouped Performance Metrics at Step 13 90
4.5 Grouped Class-Wise Accuracy 91
4.6 Grouped Hardware Utilization - GPU & CPU 92
4.7 Grouped Hardware Utilization - RAM & VRAM 93

A.1 Maximal Performance of All Model Runs 129
A.2 Performance of All Model Runs at Step 13 129
A.3 Class-Wise Accuracy of All Model Runs 130
A.4 Hardware Utilization - GPU & CPU of All Model Runs 135
A.5 Hardware Utilization - RAM & VRAM of All Model Runs 135

List of Acronyms and Abbreviations

AI Artificial Intelligence

ANN Artificial Neural Network

AOI Area of Interest

CA Class-Wise Accuracy

CLE Cross-Entropy Loss

CNN Convolutional Neural Network

CPU Central Processing Unit

CUDA Compute Unified Device Architecture

CV Computer Vision

DEM Digital Elevation Model

DL Deep Learning

DNN Deep Neural Network

ESA European Space Agency

FCN Fully Convolutional Network

FFN Feed-Forward Network

FN False Negative

FP False Positive

GELU Gaussian Error Linear Unit

GPU Graphics Processing Unit

HCA Hierarchical Clustering Analysis

HeiGIT Heidelberg Institute for Geoinformation Technology

IoU Intersection over Union

LCBB Lane-Count-Based Buffer

LULC Land Use Land Cover

LULC Utility Land Use Land Cover Utility

LULC-M Land Use Land Cover Mapping

MA-HD-RNK Mannheim - Heidelberg - Rhein-Neckar-Kreis

MHA Multi-Head (Self-)Attention

MiT Mix Transformer

ML Machine Learning

MLP Multi-Layer Perceptron

NaN Not a Number

NLP Natural Language Processing

NN Neural Network

OA Overall Accuracy

OHE One-Hot Encoding

OS Operating System

OSM OpenStreetMap

PPMC Pearson's Product Moment Correlation

PVT Pyramid Vision Transformer

RAM Random Access Memory

ReLU Rectified Linear Unit

ResNet Residual Network

RF RandomForest

SDG Sustainable Development Goal

SDPA Scaled Dot-Product Attention

SETR Segmentation Transformer

SGD Stochastic Gradient Descent

TN True Negative

TP True Positive

ViT Vision Transformer

VRAM Video Random Access Memory

WSL Windows Subsystem Linux

Chapter 1

Introduction

It's everywhere: in the news, in research and science, on the web, in our hands, used in our daily lives as a companion, its usage strongly debated in universities and schools, and its possibilities seemingly endless (Roumeliotis & Tselikas, 2023). Artificial Intelligence (AI) is currently one of the most discussed technological topics thanks to the development of large language models and the emergence of OpenAI's famous ChatGPT (Generative Pre-Trained Transformer, (OpenAI, 2022)) towards the end of 2022, functioning as a chatbot successfully mimicking human conversation (Abdullah et al., 2022; Alzubaidi et al., 2021). Despite its term being coined as far back as 1956 during the Dartmouth Conferences, it is the recent improvements in computational power, networks, and the advent of big data that have truly advanced its growth and application (Ongsulee, 2017). AI has already reinvented numerous sectors, from healthcare to finance, by automating tasks, improving decision-making processes, and providing new insights from data (Alzubaidi et al., 2021; Bini, 2018; Ongsulee, 2017; Roumeliotis & Tselikas, 2023).

Machine Learning (ML), a subfield of AI that enables data-driven pattern recognition and sits at the intersection of AI and data science, has emerged as a powerful tool for global sustainable development. One of the most critical issues in the 21st century, it aims at fulfilling current requirements of human development without compromising future generations (United Nations, 2023). ML provides innovative solutions for monitoring and managing natural resources, predicting environmental changes, and driving sustainable practices across various sectors (Getzner et al., 2023; Kar et al., 2022; Mehlin et al., 2023). Goal 13 "Climate Action" of the Sustainable Development Goals (SDGs), set by the United Nations to be achieved by 2030, explicitly addresses the need for urgent action to combat climate change and its impacts. Furthermore, goal 11 "Sustainable Cities and Communities" aims to make cities inclusive, safe, resilient, and sustainable, by addressing comprehensive urban issues such as infrastructure, management, and urban planning, a goal including the advancements in AI and, specifically, ML and its possibilities for sustainable development (United Nations, 2023).

An important geographical research field between AI and sustainability is the utilization of ML models for Land Use Land Cover Mapping (LULC-M), which gains a lot of attention from the need for sustainable development and urban planning (S. Zhao et al., 2023). The Intergovernmental Panel on Climate Change states that "global greenhouse gas emissions have continued to increase, with unequal historical and ongoing contributions arising from unsustainable energy use, land use and land use change, lifestyles and patterns of consumption and production across regions, between and within countries, and among individuals" (IPCC, 2023, p. 4). Land Use Land Cover (LULC) is named explicitly as one of the key sectors where more sustainable interaction methods are needed to mitigate climate change, showing its importance in the context of sustainable development.

LULC-M segments the landscape into patches of different land cover and land use types which gives valuable insights into their distribution and interaction at a given time (J. Li et al., 2024). Its largest potential lies in monitoring human-made and natural landscape changes which enables stakeholders to adapt the anthropogenic land uses in order to act more sustainable in and with the environment (Moharram & Sundaram, 2023). Moreover, it plays a crucial role in diverse sectors such as disaster management, monitoring carbon emissions, and agricultural production, which underlines its broad spectrum of applications (Alhassan et al., 2020; Rangel et al., 2024; S. Zhao et al., 2023).

Initially done manually or semi-automated, LULC-M has been revolutionized by the advent of ML and Deep Learning (DL) models, especially image segmentation models, which have significantly improved the accuracy and efficiency of the process (S. Zhao et al., 2023). These models are trained on large datasets of satellite imagery to predict the LULC classes of a given area, which can then be used for further analyses like monitoring changes over time and space (J. Li et al., 2024). Especially DL models building on the Transformer, serving as the foundation for ChatGPT, have shown great potential in this field due to their ability to capture long-range dependencies and contextual information in images by utilizing attention mechanisms, partially enlarging receptive fields beyond their initial fixed reach (Alhichri et al., 2021; L.-C. Chen et al., 2016; Xie et al., 2021; Zheng et al., 2021).

However, development and operation of ML models require substantial computational resources and energy, which adds negatively to their environmental benefits. This is in particular true for DL models as training increasingly complex models for improved accuracy results in greater energy use and carbon emissions (Desislavov et al., 2023; Mehlin et al., 2023; Strubell et al., 2019). It is estimated that training a large Transformer model with 213 Million parameters (like a GPT) emits as much carbon dioxide as an average car drive for nearly 400 Kilometers (Strubell et al., 2019; University of Michigan Center for Sustainable Systems, 2023). This shows that the environmental impact of ML models is not to be underestimated and that their sustainability has to be addressed alongside improving solely their performance. This ensures the technology's overall viability in the long term, also in respect to the named SDGs (Getzner et al., 2023; Thompson et al., 2022).

So, as the power of AI is harnessed for sustainable development, there are two main ways to enhance ML and DL models: either enhancing the performance by utilizing model improvements or reducing the resource consumption by utilizing sustainable methods for existing models (Getzner et al., 2023; Moharram & Sundaram, 2023; S. Zhao et al., 2023). This is a constant goal in the field of ML and DL (Ongsulee, 2017; Roumeliotis & Tselikas, 2023). In order to reduce the consumption of resources, a key strategy involves supporting model convergence with intelligent adaptations along with keeping the accuracies at a similar level, thereby shifting from a "faster-larger-better" to a "similar-but-more-efficient" approach (Lazzaro et al., 2023; Mehlin et al., 2023; Tao et al., 2022). But, such a strategy can also enhance the accuracy of the models, as the models can make more informed decisions about the data they are trained on (Mehlin et al., 2023).

Based on the possibilities brought by the attention mechanisms, larger receptive fields, and the idea of multiple studies, it seems that in the field of LULC-M, providing spatial contextual information to DL models is able to fulfill both roles. Y. Zhao et al. (2023) describe the main concept of spatial context using Tobler's First Law of Geography (Tobler, 1970): "The spatial distribution of geographical things

or attributes is interrelated with each other, and it appears with clustering, random, and regular characteristics. The relationship can be described by the spatial context. Hence, spatial context extraction from the surrounding environment of geographic objects can enrich their characteristics and is very useful for predicting their types" (Y. Zhao et al., 2023, p. 2). In their study, they used multiple spatial contexts like intersections, surrounding buildings, and points of interest to predict road types in satellite imagery. Additionally, they used tags from OpenStreetMap (OSM) which added information about traffic behavior and therefore the importance of roads, helping classifying the hierarchically structured road types. Y. Xu et al. (2022) and Zong et al. (2020) also utilized context inclusion, including points of interest and nighttime lights along with other data, to enhance LULC-M. They and others additionally used road networks from OSM to divide their Area of Interests (AOIs) into research units and urban segments. Forget et al. (2018) went even a step further and assumed a correlation between road network and adjacent urban blocks as well as building types in order to enhance the prediction of built-up areas, an assumption similarly established by Ahmadzai (2020). They all showed that including spatial contextual information greatly helps in prediction and can be used to enhance the performance of models, as proposed in its essence as early as in Tobler's First Law of Geography in 1970.

This raises the question: if a road network provides additional attributes like road types, can it be utilized to enhance the mapping of all the surrounding LULC areas, thus, LULC-M? This could be a valuable addition to the model, as roads are often built with adjacent LULC classes in mind and, therefore, should give hints towards them (Ahmadzai, 2020; Levinson et al., 2007; Zeng et al., 2020).

The main task of this study is the evaluation of a contextual inclusion approach which involves the integration of a road network into the feature space of a LULC-M model through feature engineering. The aim is to enrich the model with valuable spatial contextual information about the adjacent LULC classes, given their specific geographical positioning within the landscape and relationship to the road network. Hypothetically, by adding information about the spatial context of LULC classes using a road network, the model convergence should be supported and the overall resource consumption reduced alongside an increase of the performance, as the model can make more informed decisions about the LULC classes.

The DL framework selected for assessing this hypothesis is the Land Use Land Cover Utility (LULC Utility) (HeiGIT, 2024b), a tool developed by the Climate Action focus group of the Heidelberg Institute for Geoinformation Technology (HeiGIT) for LULC-M (HeiGIT, 2024a). The LULC Utility utilizes the SegFormer architecture proposed by Xie et al. (2021), a Transformer-based semantic segmentation model originally for RGB images, with adaptations to support multi-spectral satellite bands and a flexible framework to allow for easy integration of additional data sources.

The research questions of this study outline the aim of integrating a road network into the feature space of a suitable DL model and assessing the impact of integrated spatial contextual information on the model's performance with:

RQ 1: *Which attributes of the OSM road network can provide the model spatial contextual information regarding LULC classes, and what is their relationship to adjacent LULC classes derived from OSM?*

RQ 2: *What is a suitable encoding method to integrate them into the feature space of the model?*

RQ 3: *How much does the integrated road network and its attributes impact the model's performance and resource consumption?*

The further structure of this thesis is organized specifically to address these research questions. First, LULC-M and the segmentation technique are explained, followed by an introduction to DLs for a basic understanding of the training process of neural networks. Later, the background of the SegFormer is explained with a deep dive into attention mechanisms. The methodology outlines the research design used to answer the research questions, including a description of OSM and utilized datasets, the explorative relationship analysis between select road attributes and LULC classes, and the model used as baseline. It further answers RQ 2 by showing four different approaches to integrate the road network into the feature space of the LULC Utility. The results of the experiments are presented and discussed critically in regard to the research questions and similar studies afterwards. The thesis is concluded with an outlook on future research and a summary of the key findings.

Chapter 2

Theoretical Background

This chapter provides the theoretical background for the study. It first gives an introduction to LULC-M and semantic segmentation. Afterwards, basics of Neural Networks (NNs) and Deep Neural Networks (DNNs) are explained, including training concepts and some milestone model architectures for Computer Vision (CV). A deeper dive into the attention mechanism and the SegFormer architecture follows, which forms the primary theoretical emphasis of the study. This enables the integration of contextual data into the feature space, which is approached in the next chapter.

2.1 LULC Mapping

Being a classical remote sensing task and having largely profited from advancements in ML, LULC-M is the process of segmenting satellite images into LULC classes as seen in figure 2.1 (J. Li et al., 2024). They represent natural and anthropogenic features on the Earth's surface, whereas "land use" is the human activity that takes place on the land, such as agriculture, forestry, or urban development, and "land cover" refers to the physical characteristics of the land, such as vegetation, water, or soil (C. Zhang & Li, 2022). LULC are critical components of the Earth's ecosystems and essential for monitoring and managing the environment (S. Zhao et al., 2023). Especially in the context of climate change and the quoted SDGs, LULC-M is a key factor for understanding the impact of human activities on the environment and for developing strategies to mitigate these impacts (Moharram & Sundaram, 2023).

Y. Wang et al. (2023) present a plethora of different LULC products of different spatial scale, ranging from local to global products in both low and high spatial resolution. The literature on LULC products is vast, with many studies focusing on the development of automated algorithms for creating better LULC maps with higher resolutions on greater scales. These studies have used a variety of methods, including ML, DL, and traditional image processing techniques, to classify land cover and land use types, primarily from remote sensing data, but also by utilizing data fusion methods (Alhassan et al., 2020; Comber & Wulder, 2019; Rangel et al., 2024; Y. Wang et al., 2023; Yu et al., 2014).

Traditionally, LULC-M was done manually or semi-automated using remote sensing data and domain knowledge. LULC areas could be identified by their spatial distribution, phenotype, and morphology. Remote sensing data, especially satellite imagery, has been crucial for LULC-M since the Landsat-1 satellite in 1972, which carried the first multi-spectral camera system. Through this, LULC areas could additionally be identified by their different spectral properties (Ustin & Middleton, 2021; C. Zhang & Li, 2022). Satellite imagery has some unique characteristics, which is why it is used for LULC-M. It offers different spatial and temporal scales, can carry different payloads targeted at specific tasks like multi- and hyper-spectral imagery, and generates huge amounts of data in short time, both an advantage and a

challenge (Rolf et al., 2024). At this time, despite LULC-M being highly accurate due to the expertise of the interpreter, it was time-consuming, spatially inefficient, and prone to discrepancies due to different interpretations (Rangel et al., 2024; Tzepkenlis et al., 2023).

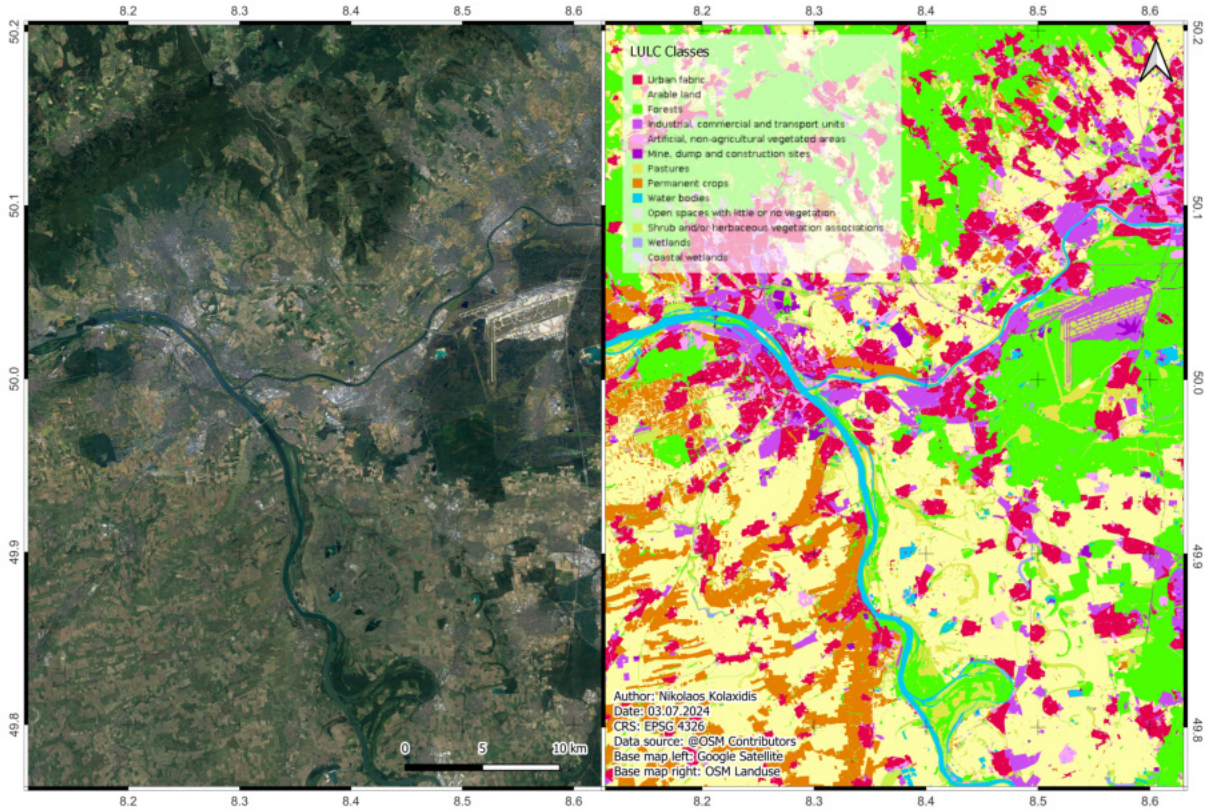


FIGURE 2.1: Exemplary depiction of the LULC mapping process from an RGB satellite image (left) to a segmented map of different LULC classes (right), somewhere in southwestern Germany. Utilizing sophisticated DL methods, more LULC classes can be identified which otherwise could be not distinguished with the human eye. The segmentation method utilized different data than the satellite image depicted.

Over the years, advances in remote sensing technology have led to satellites with better resolutions also in multi-spectral wavelengths, thus improving the quality of satellite images and LULC-M simultaneously (Alhassan et al., 2020; R. Cao et al., 2018; Kussul et al., 2017; C. Zhang & Li, 2022). But, their size and complexity have also increased, making manual interpretation even more challenging. With the recent progress in AI and ML, especially in the domain of CV, LULC-M can be done more efficiently, both regarding time and spatial resolution (Comber & Wulder, 2019; S. Zhao et al., 2023). This progress highlights the shift from broad, general maps to highly detailed ones, offering precise information about land patterns and uses (Comber & Wulder, 2019; Ongsulee, 2017; C. Zhang & Li, 2022). Although many algorithms for LULC-M exist and they already perform quite well, it continues to be a challenging task because of the high abstraction level from single pixels to objects to whole landscapes and the complexity of landscapes with various LULC types (Qi et al., 2015; S. Zhao et al., 2023).

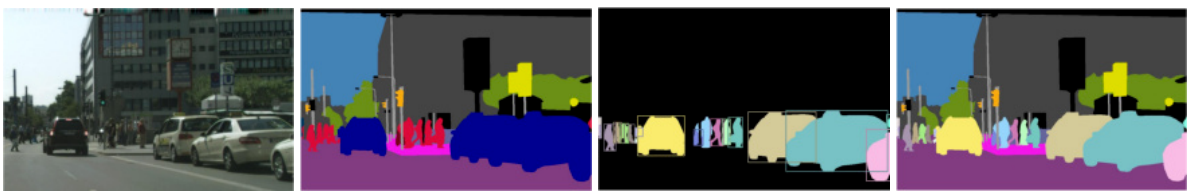
Basheer et al. (2022) and Rangel et al. (2024) name a plethora of different ML models which have been successfully utilized for LULC-M, including RandomForests (RFs), Artificial Neural Networks (ANNs), as well as ensemble learning methods. (J. Li et al., 2024; Moharram & Sundaram, 2023; Rangel et al.,

2024). However, the most promising results are dominated by DL models like Convolutional Neural Networks (CNNs), Autoencoders, Transformers, and their follow-ups, published only recently (J. Li et al., 2024; Rangel et al., 2024; Tzepkenlis et al., 2023). This is also shown by Boston et al. (2022) and Kussul et al. (2017). They compared the performance of CNNs and RFs for LULC-M and found that the CNNs outperformed the RFs in terms of accuracy and computational efficiency. A similar conclusion was made by Digra et al. (2022), Moharram and Sundaram (2023), and Tzepkenlis et al. (2023), who value the DL approaches more promising for LULC-M than other ML models, emphasizing their high potential for innovative advancements in the future.

2.2 Semantic Segmentation

There are two main types of DL based CV methods that have proven their suitability for LULC-M, particularly image classification and image segmentation (Boston et al., 2022; Digra et al., 2022; Kussul et al., 2017). While classification utilizes object detection methods to label whole images based on the detected primary object in them, segmentation assigns each pixel to a class based on its spectral properties and spatial relationships, allowing for a more detailed analysis of the image (Alhassan et al., 2020; J. Li et al., 2024; Rangel et al., 2024; Szeliski, 2022). It is important to differentiate between classification and segmentation, as these terms are occasionally used interchangeably by authors (R. Cao et al., 2018; Onim et al., 2020). This confusion arises because segmentation involves categorizing segments, essentially classifying them into different groups, therefore, incorporating classification (Alhassan et al., 2020).

Because image segmentation considers all pixels in an image and not only the whole image, it is more suitable for detailed LULC-M mapping (J. Li et al., 2024). Moreover, it distinguishes itself from image classification and object detection by its necessity to segment even unknown objects. To be more specific, it has to acquire new knowledge about a class without prior knowledge about it (Guo et al., 2018). This proves an additional challenge on the one hand, but a strong advantage on the other hand. Furthermore, Y. Chen et al. (2023) and Guo et al. (2018) state that semantic segmentation requires less preprocessing of the data and is more efficient in predicting end-to-end ground features, because it processes multi-spectral images in full rather than patch- or feature-based. That makes image segmentation all in all very suitable for tasks like LULC-M.



(A) Original Image. (B) Semantic Segmentation. (C) Instance Segmentation. (D) Panoptic Segmentation.

FIGURE 2.2: The three different image segmentation types compared to the original image (A), semantic (B), instance (C), and panoptic segmentation (D). Adapted from Kirillov et al. (2019) and Szeliski (2022).

In image segmentation, there are three main types: instance segmentation, semantic segmentation, and panoptic segmentation, which are exemplarily shown in figure 2.2. Instance segmentation, strongly

building on object detection methods, identifies every significant object within an image and creates precise pixel-level outlines for their visible areas. In semantic segmentation, each pixel of the whole image is assigned to a class without distinguishing between different objects within the same class. Panoptic segmentation is a combination of these two, identifying and segmenting significant objects, but also assigning everything else in the image to classes (J. Li et al., 2024; Minaee et al., 2022; Szeliski, 2022).

In the context of LULC-M, semantic segmentation is advantageous for its missing focus on specific objects, but classes distributed over the whole image. Semantic in this context means that similar features inherit similar characteristics, they are "semantically" related (Shanmugamani, 2018). Contrary to the other two image segmentation types, semantic segmentation treats all instances of a class as a single entity. This means that if an image contains multiple objects of the same class, such as several cars, semantic segmentation does not distinguish between different cars but labels all pixels belonging to any car as "car". The focus is on understanding the image at a class level, identifying and categorizing the entire area of the image into meaningful classes based on the spectral properties of the pixels and function of the regions, without concern for individual object identities. Thus, LULC-M is preferably done by utilizing semantic segmentation, although other methods also are suitable (J. Li et al., 2024; Minaee et al., 2022; Shanmugamani, 2018; Szeliski, 2022).

2.3 Machine & Deep Learning

In order to understand the process of LULC-M using semantic segmentation and why DL is so computationally demanding, it is necessary to have a basic understanding of the fundamentals of ML, DL, and the training process of ANNs. DL is a subfield of ML, a field that emphasizes data-driven pattern recognition. Unlike traditional programming, where tasks are solved by explicitly programming rules, ML algorithms learn these rules from the data on their own. This "black box" approach, where the input is data and the output is an applicable model rather than a result, has revolutionized many fields (Bernard, 2021; Sarker, 2021; Shinde & Shah, 2018). ML algorithms are adept at recognizing complex patterns in data. Once these patterns are learned, they can be applied to new problems for prediction. This has led to their widespread use in diverse fields such as CV and Natural Language Processing (NLP). They are suitable for a plethora of tasks, such as classification, regression, transcription, translation, anomaly detection, and missing value imputation, along with many others. The choice of model depends on the task at hand, the data available, and the desired outcome (Goodfellow et al., 2016).

The main difference between ML and DL is the complexity of the models. ML models are usually simpler and have fewer parameters, making them easier to interpret and faster to train. However, they need elaborate manual preprocessing of data, feature extraction, and feature selection to perform well. DL models, on the other hand, are more complex and have more parameters, making them harder to interpret and slower to train. But, they can learn complex patterns from raw data, eliminating the need for manual feature extraction and selection. This makes them more suitable for tasks where the data is complex and the patterns are not easily discernible. Furthermore, DL models are highly scalable, adapting to the problem at hand, and offer a high degree of generalization, where the same model architectures can be used with different data for multiple tasks (Alzubaidi et al., 2021).

In DL, models mimic the complex NN of the human brain to analyze and interpret data. Multi-layered (deep) ANNs are employed for feature extraction, utilizing these features for predictive and decision-making purposes. The advantages of DL include the capability to process complex and voluminous data, achieve high accuracy, and autonomously identify data features, further enhancing its suitability for named fields (Bernard, 2021; Szeliski, 2022). Nonetheless, it faces challenges such as the need for extensive datasets for training, significant computational resources, and specific data quality demands. Moreover, the models are difficult to interpret due to their "black box" approach and high complexity. Still, DL incorporates a range of architectures of cutting-edge models, which played a crucial role in the progression of the field, offering solutions to previously challenging problems and paving the way for new research and practical applications (G. Cao, 2022; Reichstein et al., 2019; Sarker, 2021).

ML and DL algorithms can be split into two main paradigms: supervised and unsupervised. In supervised learning, the algorithm learns from labeled data, where the input data is paired with the correct output. The algorithm learns to map the input data to the output data, making predictions based on this mapping. In unsupervised learning, the algorithm learns from unlabeled data, where the input data is not paired with the correct output. The algorithm learns to find patterns in the data without "knowing" what the data is, clustering similar data points together or reducing the dimensionality of the data (Goodfellow et al., 2016; Shinde & Shah, 2018; Szeliski, 2022; A. Zhang et al., 2023). There are also subclasses like semi- and self-supervised, but these are not as relevant at this point.

Semantic segmentation can be both, supervised and unsupervised (Guo et al., 2018; Hamilton et al., 2022; S. Zhao et al., 2023). But, for LULC-M, as it requires labeled data to learn the mapping between the input data and the output data, it usually is supervised, although much research is focused on semi- and self-supervised learning methods (Guo et al., 2018; J. Li et al., 2024). Nevertheless, the basic architectures and training concepts are the same for both paradigms.

2.4 Artificial Neural Networks

In their essence, ANNs, called NNs from this point on, are constructs of multiple multiplication and addition operations, which are performed on the input data to produce an output. They are composed of multiple layers of interconnected nodes, or neurons, that process data in a hierarchical manner. Each layer of an NN performs a specific transformation on the input data, and the output of one layer serves as the input to the next layer. The final layer of the network produces the output, which is used to make predictions or decisions (Goodfellow et al., 2016). These operations are carried out in the form of matrix multiplications and additions, where the features' weights are learned during the training process. This involves adjusting the weights of the model to minimize the difference between the predicted output and the actual output, using a process called backpropagation. Different kinds of layers, such as convolutional, pooling, and fully connected, as well as activation functions, regularization techniques, and optimization algorithms, are used to build the different architectures of NNs and DNNs (Goodfellow et al., 2016; Y. LeCun et al., 2015; Szeliski, 2022; A. Zhang et al., 2023).

Some of these terms are explained in more detail in the following sections with the focus on classification tasks, as they are the basis for semantic segmentation and the development of the SegFormer. ML and especially DL are vast fields with much current research going on. This study only provides

an overview of the basics to understand the SegFormer architecture and the framework of the utilized LULC Utility. For further information, especially for a deeper explanation of the mathematical background, the works of Bernard (2021), Bishop (2006), Goodfellow et al. (2016), Nielsen (2015), and A. Zhang et al. (2023) are recommended, among others.

2.4.1 Perceptrons

In ML, the simplest NN is the Perceptron, an algorithm for supervised binary classification, capable of predicting whether an input belongs to a given class (Rosenblatt, 1957). Consisting solely of one artificial neuron (called simply neuron from this point on), the perceptron is inspired by the biological neuron in the brain's nervous system, capable of classifying linearly separable data (Rosenblatt, 1957; Szeliski, 2022). The following detailed explanation of the Perceptron is summarized from Bernard (2021), Bishop (2006), Goodfellow et al. (2016), Y. LeCun et al. (2015), Nielsen (2015), Szeliski (2022), and A. Zhang et al. (2023), where more details can be found.

Architecture & Forward Propagation

Technically not a network, the perceptron, visualized in figure 2.3, utilizes a single neuron to make binary classifications. It consists of three main components: input features, weights, and bias.

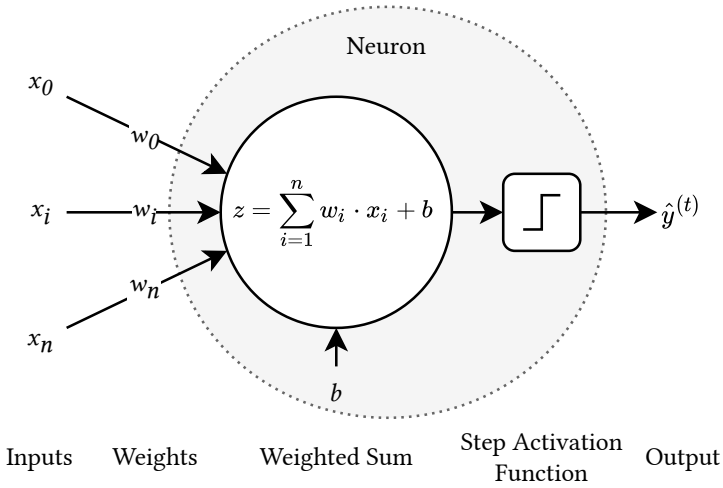


FIGURE 2.3: Schematic visualization of the Perceptron's architecture. In gray the artificial neuron. Conceptualized from Nielsen (2015) and Sarker (2021).

The input features are the characteristics or properties of the data item being classified, represented as a vector $\mathbf{x} = [x_1, x_2, \ldots, x_n]$, where n is the number of features. Each x_i is a numerical value indicating the presence or magnitude of a specific characteristic of the input. Each feature x_i is associated with a weight w_i, which are adjustable parameters that determine the importance of each feature in the classification process. Initially, these weights can be set to small random values or zeros.

In addition to weights, the Perceptron includes a bias b, an additional parameter that helps adjust the output independently of the input features. The bias allows the decision boundary to shift to a specific value range, increasing the model's flexibility for particular tasks.

The Perceptron computes a weighted sum (Nielsen, 2015; Szeliski, 2022) of the input features plus the bias with

$$z = \sum_{i=1}^{n} w_i x_i + b \tag{2.1}$$

as shown inside the neuron in figure 2.3. This sum, z, is also known as the activation value, which is fed into a so called activation function. For binary classification, the Perceptron uses a step activation function, which outputs one class label if z is above a certain threshold (typically zero) and another class label if it is below. Therefore, this function transforms the linear combination of inputs into a binary decision. The process of feeding input vectors into the neuron, which calculates the weights sum and applies the activation function, is called forward propagation, the first step in the training process of the Perceptron.

Training

Training the Perceptron involves adjusting the weights and bias to minimize classification errors on a training dataset. This process is iterative and involves a straightforward algorithm consisting of forward and backward passes through the model, called steps. Initially, weights w_i and bias b are set to small random values or zeros. For each training example $[\mathbf{x}^{(t)}, y^{(t)}]$, where $y^{(t)}$ is the true label for a specific training sample, the weighted sum z is computed. The activation function is applied to get the predicted output $\hat{y}^{(t)}$. Then, the error $e^{(t)}$ (Bishop, 2006) is calculated as

$$e^{(t)} = y^{(t)} - \hat{y}^{(t)} \tag{2.2}$$

and the weights and bias are updated based on the error. The updates are done using the learning rate η, which controls how much the weights are adjusted (A. Zhang et al., 2023). The weight update is proportional to the product of the error, the input feature, and the learning rate, while the bias update is proportional to the product of the error and the learning rate, specifically,

$$w_i \leftarrow w_i + \eta \cdot e^{(t)} \cdot x_i^{(t)} \quad \text{and} \quad b \leftarrow b + \eta \cdot e^{(t)}. \tag{2.3}$$

Minimizing the error by updating weights and biases is done for multiple steps, until the algorithm reaches a state where further training does not significantly alter both components. At this point it has reached convergence, indicating that it has effectively learned from the training data.

Inference

Once the Perceptron is trained, having adjusted its weights and biases to minimize classification errors on the training dataset, it can then be used to classify new, unseen data. The process for classifying new data points involves calculating the weighted sum using the learned weights and biases, applying the activation function, and outputting the predicted class label, all in only one forward pass. This process is known as inference, where the model applies the learned patterns to new data to make predictions.

In some cases, after initial deployment, the Perceptron's performance might be evaluated further. If the classification accuracy on new data is not satisfactory, additional adjustments or retraining with more data might be necessary.

Activation Functions

While the Perceptron is simple and intuitive, it has a defining limitation. It can only solve problems that are linearly separable and might struggle converging with other data. To overcome this limitation, other activation functions h like the very popular Rectified Linear Unit (ReLU) (Szeliski, 2022) can be utilized. This allows non-linear decision boundaries with

$$h(z) = max(0, z), \tag{2.4}$$

whereas z is the aforementioned weighted sum, put into a different mathematical context. This effectively forms a new type of neuron which responds differently to the data. Other popular activation functions include sigmoid, tanh, and adaptations of ReLUs, among others. They all are aimed at introducing directed non-linearity into the model, allowing it to learn more complex patterns in the data tailored at specific tasks and laying the foundation for more complex NN architectures.

2.4.2 Multi-Layer Perceptrons

Using Perceptrons as the basis, more complex NN architectures can be constructed by organizing neurons into multiple consecutive layers. Such architectures are similar to Perceptrons, as they also consist of inputs, neurons, and outputs, with the difference that they introduce layers of these three components, effectively forming a neur(on)al network with multiple Perceptrons. NNs usually have three groups of layers, an input layer, an output layer, and at least one hidden layer in between. Figure 2.4 visualizes a more complex NN. This architecture is called a Feed-Forward Network (FFN), for information is fed forward through the network without loops (Nielsen, 2015; Sarker, 2021; Szeliski, 2022). The hidden layers consist of multiple neurons which perform the weighted summation and application of the activation function, while the output layer produces the final prediction.

In the case of the depicted network, the layers are fully connected, meaning that each neuron in one layer is connected to every neuron in the next layer, feeding their activated weighted sum into the next layer of neurons. A network that consists only of fully connected layers is called Multi-Layer Perceptron (MLP). Despite its name, it usually uses non-linear activation functions and therefore other types of neurons than Perceptrons. This architecture, alongside using non-linear activation functions and different types of connections between the layers, allows models to learn complex patterns in the data, forming the foundation for the majority of subsequent DNN (Goodfellow et al., 2016; Szeliski, 2022; A. Zhang et al., 2023).

Increase in Complexity

MLPs can be increased in complexity and, thus, training capabilities. By adding more neurons to a hidden layer, the network gets wider, enhancing its capacity to capture diverse information from the input data. However, this can lead to a higher tendency for overfitting, where the model learns the

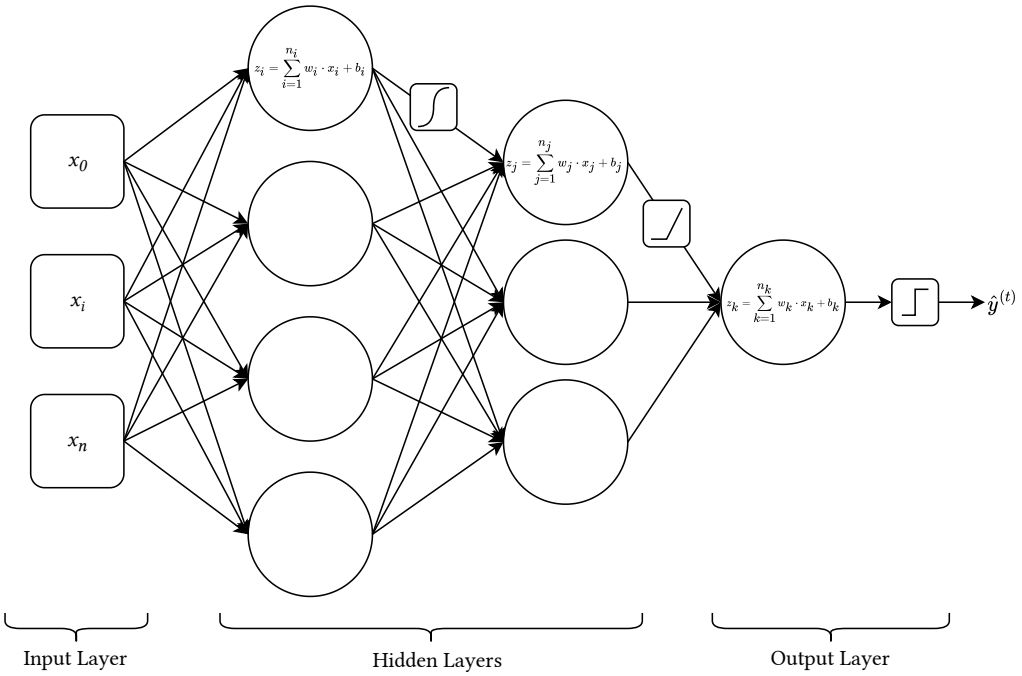


FIGURE 2.4: Detailed schematic visualization of the MLP architecture with two hidden layers and one output, all fully connected. The nodes represent neurons, weights are implicitly included in the arrows. Weight summation and activation functions are added representatively to the top neurons. Different activation functions are used solely for illustration. Conceptualized from Bernard (2021), Nielsen (2015), and Szeliski (2022).

training data too well to generalize effectively to new data. On the other hand, adding more hidden layers results in a deeper network, capable of learning more complex patterns. This depth, though, demands more computational resources and a larger dataset for effective training. A network with just one hidden layer is termed a shallow NN, whereas those with multiple hidden layers are referred to as DNNs (Bernard, 2021; Nielsen, 2015; A. Zhang et al., 2023). The MLP in figure 2.4, for example, is a DNN with two hidden layers and a (maximal) width of four neurons, for illustration purposes.

Softmax Function

Along with offering more neurons and layers for a deeper understanding of patterns in the data, MLPs offer another enhancement. When using them for multi-class classification, the output layer can consist of multiple neurons corresponding to the different classes rather than resulting in only one output as shown in both the aforementioned NNs. Depending on applied weights and biases, the raw output scores (logits) can be scaled differently, making their comparison and fusion difficult. For this purpose, a so called softmax function (called simply softmax from this point on) is introduced, where the logits are passed through. The softmax produces a vector with normalized logits, converting them into probabilities. These probabilities indicate the likelihood of the input data belonging to each class, allowing the model to make a decision based on the highest probability (Goodfellow et al., 2016; Szeliski, 2022).

The formula for the softmax for class i over all classes j is given by

$$\hat{y}_i = \frac{\exp(o_i)}{\sum_j \exp(o_j)} \quad \text{resulting to} \quad \hat{\mathbf{y}} = \text{softmax}(\mathbf{o}), \tag{2.5}$$

whereas o_i is the logit and $\hat{y}_i$ the predicted probability for class i, respectively. Note that the largest value of $\mathbf{o}$ corresponds to the class with the highest probability according to $\hat{\mathbf{y}}$ (Goodfellow et al., 2016; Szeliski, 2022; A. Zhang et al., 2023).

The softmax applies the exponential function to each logit, magnifying the differences among them. This means higher logits get much larger probabilities than lower ones. Additionally, since all logits are made positive through exponentiation and normalized by dividing by the total of all exponentiated logits, the output values are constrained between 0 and 1, summing up to 1, effectively making them interpretable as probabilities (Nielsen, 2015; Szeliski, 2022; A. Zhang et al., 2023).

2.5 Training (Deep) Neural Networks

As seen in section 2.4.1, the perceptron is trained by calculating one error term $e^{(t)}$ and updating the one bias and the according weights. In contrast to that, training NNs is more complex due to their multi-layer structure, where every neuron inherits and produces weights and biases. It involves minimizing a loss function through an iterative optimization process known as gradient descent. This process is enabled by backpropagation, an algorithm that efficiently computes gradients for adapting each parameter in the network. In this section, these concepts are explained in more detail to provide a better understanding of the complex training process of NNs, which also applies to DNNs, in order to show why DL is so computationally demanding. Additionally, these concepts come into effect in the LULC Utility, especially optimization techniques, to increase the training efficiency and make the framework adaptable and scalable.

2.5.1 Tensors & Feature Space

The first thing to mention is the data which is used in ML and DL, specifically focusing on tensors and the feature space, which are fundamentally interconnected. Tensors, which can be scalars (0D), vectors (1D), matrices (2D), or higher-dimensional arrays (3D, 4D, etc.), represent numerical data within the feature space. They can be arithmetically altered, providing a flexible datatype for efficiently handling data, and are used to represent input data, weights, biases, and intermediate results during training and inference. They are the fundamental data structure in DL, allowing for efficient computation and storage of large amounts of data (Goodfellow et al., 2016; A. Zhang et al., 2023)

The shape of a tensor directly reflects the structure of the feature space. For example, an image with a resolution of 100×100 pixels and three color channels (RGB) can be represented as a 3D tensor with the shape $(100, 100, 3)$, where the first two dimensions represent the spatial dimensions of the image, height and width, and the third represents the color channels, red, green, and blue. The channels represent the value vectors of the features, in this case pixels, so all the information values the pixels can have on multiple dimensions (red, green, and blue) in the according image spaces. Therefore, feature space primarily means the channels of the tensors. More channels mean larger feature space, as features

can have more information values describing the features/pixels (Bernard, 2021; Bishop, 2006; Courtial et al., 2022; Szeliski, 2022; A. Zhang et al., 2023).

"Input data" from this point on means input tensor, feature space means dimensionality or channels of the according tensor.

2.5.2 Loss Functions

The loss function is a measure of the model's performance, indicating how well the model's predictions match the true labels, similar to the error when training perceptrons. Loss functions are used to aggregate the error over all instances, including softmax vectors, and map it to a non-negative value to guide the learning process. Again, the goal is to reach convergence by minimizing this aggregated loss, thereby improving the model's predictions. For different tasks and different datasets, specific loss functions can be used to and enhance the adaptability and flexibility of NNs (Bernard, 2021; Goodfellow et al., 2016; Janocha & Czarnecki, 2017).

Common loss functions include the Mean Squared Error for regression tasks and the Cross-Entropy Loss (CLE) for classification tasks (Goodfellow et al., 2016; Janocha & Czarnecki, 2017; Szeliski, 2022). CLE, also called the log loss, measures the performance of a classification model whose output is a probability value between 0 and 1, indicating the belonging to a class (softmax). The loss L is defined as

$$L_{CLE}(y, \hat{y}) = -\sum_{i} y_i \log(\hat{y}_i), \tag{2.6}$$

whereas y_i is the true label (0 or 1) and $\hat{y}_i$ is the predicted probability for class i, respectively (A. Zhang et al., 2023). Minimizing this loss function ensures that the predicted probabilities are as close as possible to the true labels.

An example: for a three-class classification problem, the softmax vector is $[0.1, 0.7, 0.2]$. Suppose the true label is class 2, so classes 1 and 3 are 0/false and class 2 is 1/true. The CLE is calculated as

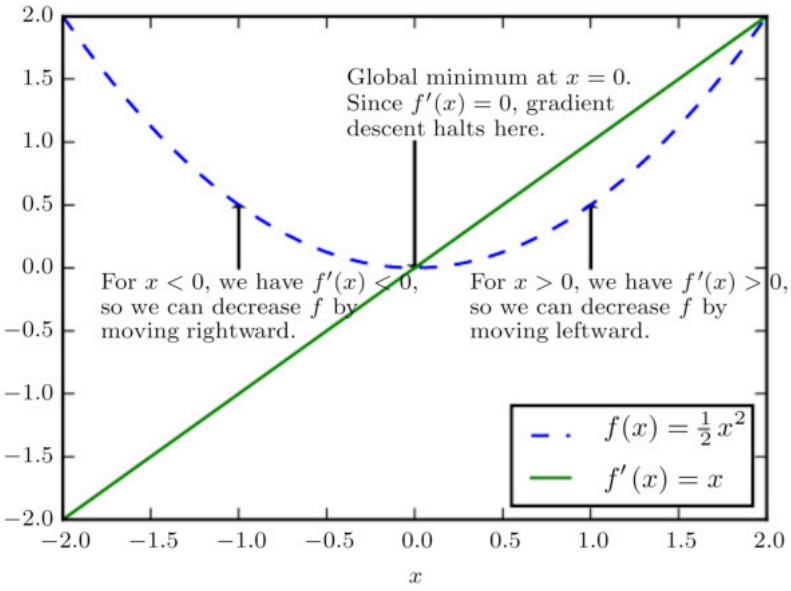
$$L_{CLE} = -(0 \cdot \log(0.1) + 1 \cdot \log(0.7) + 0 \cdot \log(0.2)) = -\log(0.7) \approx 0.155 \tag{2.7}$$

while for a true label of class 3, the loss would be $-\log(0.2) \approx 0.7$. This simple example shows that the loss function penalizes the model more for incorrect predictions. For a perfect probability of 1 for the true class, the loss would be 0, indicating a perfect prediction (Szeliski, 2022; A. Zhang et al., 2023).

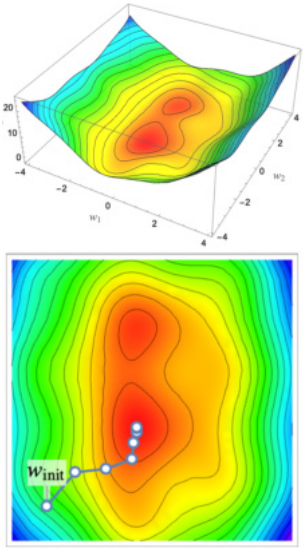
2.5.3 Gradient Descent

In order to minimize the loss, the weights and biases of all neurons have to be adjusted. This is where gradient descent comes into play, an optimization algorithm originally proposed by Cauchy in 1847 (Netrapalli, 2019). Gradient descent is used to minimize the loss function by iteratively adjusting the model's parameters in the opposite direction of the gradient of the loss function with respect to the parameters. Figure 2.5a visualizes this concept of gradient descent. Goodfellow et al. (2016) explain it this way: "Suppose we have a function $y = f(x)$, where both x and y are real numbers. The derivative of this function is denoted as $f'(x)$ or as $\frac{dy}{dx}$. The derivative $f'(x)$ gives the slope of $f(x)$ at the point x.

In other words, it specifies how to scale a small change in the input to obtain the corresponding change in the output" (Goodfellow et al., 2016, p. 81). So, the gradient is the derivative of the loss function, mathematically representing the slope of the loss function at a specific point (Nielsen, 2015; Szeliski, 2022). This is why all calculations in NNs aimed at training the model, especially the loss function, have to be differentiable, so the gradient can be calculated. This restricts the usage of some activation functions and loss functions.



(A) Mathematical Gradient Descent (Goodfellow et al., 2016).



(B) Gradient Descent visualized as a landscape. Adapted from Bernard (2021).

FIGURE 2.5: Illustrating how the gradient descent algorithm uses the derivatives of a function to follow the function downhill to a minimum.

In visual language, gradient descent can be described as moving downhill in a mountainous landscape to reach the lowest point. The landscape represents the feature space, where each position on the surface represents a particular set of parameters (weights and biases) of the model. Figure 2.5b visualizes this further, showing the landscape for two parameters w_1 and w_2 and the current point w_{init}, representing the current step's loss gradient. The steepness and direction of the slope at the current position represent the gradient of the loss function. Gradient descent aims to find the direction of which the network should shift its parameters to further minimize errors along the steepest path possible. This is done iterative, adjusting the parameters based on the gradient for each step (Bernard, 2021; Goodfellow et al., 2016; Nielsen, 2015; Szeliski, 2022).

Global & Local Minima

In DNNs, the landscape is much more complex with multiple dimensions and many parameters, usually millions, often even exceeding the number of training samples. In that high-dimensional feature space, this over-parametrization creates multiple local minima along the global minimum. The gradient descent follows the steepest path towards the next minimum, so it often aims towards the local minimum rather than the global minimum, which would represent the best model performance with

the lowest loss. This is why the gradient descent is called a local optimization method (Bernard, 2021; Szeliski, 2022).

Yet, the high-dimensionality is the problem and the solution at the same time. Because of so many parameters, there are multiple options to achieve a good model performance, even if the global minimum is not reached. "Also, the dimension of the cost-function space is so high that there is often a way out: even when the network seems to get stuck in a minima, one of the many directions might allow for escape. Finally, the gradient descent procedure is likely to end up in a local minima that has a large basin of attraction" (Bernard, 2021, p. 289), which would be an acceptable minimum (Bernard, 2021; Goodfellow et al., 2016; Szeliski, 2022).

Stochastic Gradient Descent

For large DNNs, calculating the gradient descent is computationally inefficient, because it would need to sum the loss function over all training samples. Instead, the Stochastic Gradient Descent (SGD) is utilized, which only uses a single training sample n to calculate the gradient of the according loss function $L_n(w)$. But, because only using a single sample results in a noisy and inaccurate estimate of a suitable descent direction, the losses and gradients of a small subset of the training data are summed with

$$L_{\mathcal{B}}(\mathbf{w}) = \sum_{n \in \mathcal{B}} L_n(\mathbf{w}) \tag{2.8}$$

for the vector of weights $\mathbf{w}$ consisting of the according weights of the minibatch $\mathcal{B}$, a small subset of the training data (Szeliski, 2022). Using this method, a similar estimate of the gradient descent direction can be calculated without needing the entire training dataset. This speeds up the training process and allows for more frequent updates of the model's parameters (Bernard, 2021; Goyal et al., 2018; Netrapalli, 2019; Szeliski, 2022; A. Zhang et al., 2023).

After calculating the SGD, the weights can be updated by taking a step in the gradient direction with a temporal dependency by using

$$\mathbf{w}^{(t+i)} \leftarrow \mathbf{w}^{(t)} - \eta^{(t)} \cdot \mathbf{g}^{(t)} \tag{2.9}$$

with the summed gradient $\mathbf{g}$ denoted as

$$\mathbf{g} = \nabla_{\mathbf{w}} L_{\mathcal{B}}, \tag{2.10}$$

whereas i are the steps ahead of the current step t and $\eta^{(t)}$ the current learning rate, a hyperparameter that controls the size of the step taken in the gradient direction (Szeliski, 2022; A. Zhang et al., 2023).

Learning Rate & Optimizers

The learning rate is crucial for the training process, as it determines how quickly the model learns and how well it generalizes to new data. If the learning rate is too small, the convergence will be too slow, and if it is too large, the model might not even converge and training infinitely until terminated. The choice of learning rate depends on the specific task, the dataset, and the model architecture, and it is usually tuned through hyperparameter optimization (Bernard, 2021; Szeliski, 2022).

To enhance this procedure, optimizers can be utilized, of which the the Adam optimizer (Kingma & Ba, 2015) is the most popular one. Most of these optimizers, also called adaptive learning rate methods, follow a simple principle to optimize the learning rate: when successive gradients are in the same direction, the learning rate is increased, are the gradients in the opposite directions, it is decreased. They often implement further optimization techniques to optimize their optimization method, like decaying averages of gradients, but this will not be elaborated further at this point. The automatic adaptation of the learning rate allows for optimized convergence, increasing the model's efficiency and achieving good loss minimization (Bernard, 2021; Kingma & Ba, 2015; Szeliski, 2022; A. Zhang et al., 2023).

2.5.4 Backpropagation

Efficiently computing the SGD and updating the according weights is mandatory for training DNNs. This is done via "backward propagation of errors", backpropagation for short (Szeliski, 2022).

During forward propagation (explained in section 2.4.1), the input data is passed through the network, layer by layer, to produce an output. Each layer applies its weights and activation functions to transform the data. The output is then compared to the true label using a loss function, which quantifies the error of the network's prediction. In order to effectively apply the SGD, it has to be calculated how much each parameter in the network contributed to the overall error. For this, a backward pass is performed (Goodfellow et al., 2016; Szeliski, 2022; A. Zhang et al., 2023).

The first step in backpropagation is to propagate the error backward through the network. For each neuron, the error is determined by the errors of the neurons in the subsequent layer that it connects to. This is computed similarly to forward propagation but in reverse. After propagating the error backward, the gradient of the loss function with respect to each weight and bias is calculated. This is done by multiplying the propagated error by the gradient of the activation function of the current neuron. The chain rule of calculus is utilized here, which allows the calculation of the gradient of a composed function. Szeliski (2022) explains this intuitively: "this backpropagation rule has a very intuitive explanation. The error [...] for a given unit depends on the errors of the units that it feeds multiplied by the weights that couple them together. This is a simple application of the chain rule" (Szeliski, 2022, p. 286). Once the gradients are computed, they are used to update the network's parameters using the gradient descent rule shown above in equation 2.9 (Goodfellow et al., 2016; Y. LeCun et al., 2015; Y. A. LeCun et al., 2012; Szeliski, 2022; A. Zhang et al., 2023).

In summary, backpropagation involves a forward pass to calculate the output and error, followed by a backward pass to propagate the error and compute gradients. These gradients are then used to adjust the weights and biases of the network, thereby minimizing the loss function iteratively. This process enables the network to learn from the data and improve its performance over time. Backpropagation is critical for training DNNs effectively and is one of the foundational algorithms in the fields of ML and especially DL (Goodfellow et al., 2016; Y. LeCun et al., 2015; Szeliski, 2022; A. Zhang et al., 2023).

2.5.5 Training Loop

In summary, despite being similar to training Perceptrons, training DNNs requires much more additional assumptions, optimizations, and calculations. Together, SGD and backpropagation enable the NN

to learn effectively. Gradient descent acts as the guide, directing the network towards minimizing the loss by adjusting the parameters in the right direction. Backpropagation ensures that the adjustments are computed efficiently, considering how each parameter influences the loss through the network's layers. This combination allows the NN to learn from data, improve its predictions over time, and ultimately achieve better performance on the given task (Y. A. LeCun et al., 2012; Szeliski, 2022).

During the training process, the network undergoes forward and backward propagation multiple times, corresponding to each step of gradient descent until convergence is reached. This iterative process is known as iteration, often used interchangeably with step. The training loop involves passing the training dataset multiple times through the network (multiple steps) in smaller groups called batches. This enables parallel computing and higher training efficiency. In each step, the network processes these batches, makes predictions for each data point, and adjusts its parameters during backpropagation based on the error of these predictions. Through this iterative process, the network's weights are fine-tuned, enabling the network to learn complex patterns and make accurate predictions. If the entire dataset is fed through the network forward and backward once, an epoch is completed. For the NN to learn effectively, usually multiple epochs are used, which explains why NN are computationally demanding (Bernard, 2021; Y. A. LeCun et al., 2012; Szeliski, 2022; A. Zhang et al., 2023).

Batch Size

The size of batches usually can be controlled via a batch size hyperparameter, parameters not part of the model, but tuning the network. Of all the training samples in the entire training dataset, a random selection of the size of this hyperparameter is extracted and passed through the network. A small batch size can lead to faster convergence, but it can also introduce noise into the training process. A large batch size can provide a more stable estimate of the gradient, but it can also slow down the training process. The choice of batch size depends on the specific task, the dataset, and the computational resources available (Bernard, 2021; Szeliski, 2022).

In semantic segmentation, for example, where samples are single images with multiple bands, the batch size depends on the size of the images, the number of bands, and the available memory. A batch size that is too large can lead to memory issues, while a batch size that is too small maybe is not enough to find enough distant relationships between pixels, reducing the accuracy of distributed classes. By tuning the batch size, the training process can be optimized for efficiency and performance, ensuring that the network learns effectively and generalizes well to new data (Szeliski, 2022; A. Zhang et al., 2023).

Dataset Split

Furthermore, the training data is usually split in training and test sets in order to train the model and evaluate its performance in one go. The training set is used to train the model, while the test set is used to evaluate the model's performance on new, unseen data, part of the initial dataset, but strictly not used during training. When working with tunable hyperparameters, the training dataset is further split into a training and a validation set, which is used to track the loss of the model and use it to optimize hyperparameters, also with unseen data. The difference between validation and test sets is

that validation sets are used during training to optimize the model, but test sets are only used once after the training to "test" the model's inference capability (Alzubaidi et al., 2021; A. Zhang et al., 2023).

The largest part of the dataset is usually the training set, to have a large enough sample size to learn from, usually followed by the validation set and finally, the smallest, the test set (Szeliski, 2022).

Model Performance

Successful training of a NN is indicated by the model's performance on the test set. The model's performance is evaluated using metrics such as accuracy, precision, recall, and F_1 score, depending on the specific task (more on this topic in section 3.7). These metrics provide insights into the model's performance, indicating how well it generalizes to new data and how effectively it makes predictions. By evaluating the model's performance on the test set, the effectiveness of the model can be assessed, and further adjustments can be made to improve its performance (Bernard, 2021; Szeliski, 2022; A. Zhang et al., 2023).

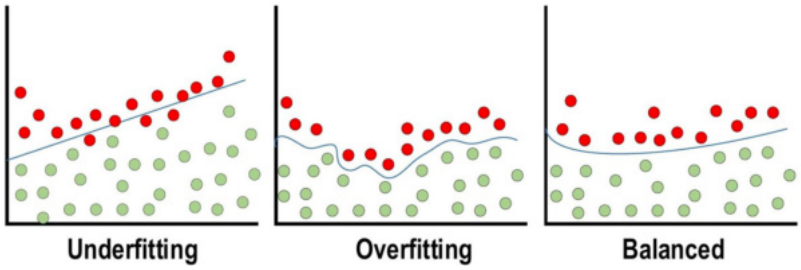


FIGURE 2.6: Evaluating the performance of NN results to three possible outcomes, of which balanced (or optimal) fitting is desired (Alzubaidi et al., 2021).

There are three possible fittings when evaluating the model's performance during training: underfitting, overfitting, and optimal fitting, as visualized in figure 2.6. Underfitting occurs when the model is too simple to capture the patterns in the data, resulting in poor performance on both the training and validation/test sets. Overfitting occurs when the model learns the training data too well, including noise and irrelevant patterns, leading to poor generalization on the validation set or new data, respectively. Optimal (or balanced) fitting occurs when the model learns the underlying patterns in the data and generalizes well to new data, achieving high performance on both the training and validation/test sets. In the best case, the loss decreases for both the training and validation sets and the accuracy increases. By evaluating the model's performance on the different data sets, the model's fitting can be assessed, and further adjustments can be made to improve its performance (Alzubaidi et al., 2021; Bernard, 2021; Szeliski, 2022; A. Zhang et al., 2023).

2.5.6 Regularization & Normalization

When training such networks, it can happen that the model overfits, a common issue in ML (Szeliski, 2022). This is where regularization come into play. Regularization techniques are designed to prevent overfitting by constraining the model's complexity and ensuring it generalizes well to new data. Techniques, such as dropout, batch regularization, and early stopping, help prevent overfitting by constraining the model's complexity and ensuring it generalizes well to new data. (Alzubaidi et al., 2021; Bernard, 2021; Nielsen, 2015; Szeliski, 2022).

Early stopping mechanisms, such as a maximum number of epochs or a minimum error threshold, both, upon reaching, terminate the training to reduce overfitting. One very common early stopping mechanism is tracking the error on the validation set and terminating the training if the error does not decrease by a certain (small) amount for a certain number of steps. This is called a patience criterion. Early stopping mechanism can also be optimized, especially when restricting the number of epochs the model is trained on. If the number of epochs is too low, the model may not have enough time to learn the patterns in the data, and the performance may be poor. Contrary, if the number of epochs is too high, the model may overfit the training data, and the performance may be poor on new data. By setting specific restrictive mechanisms, the model can be trained much more efficient, sometimes saving a lot of time and costs (Goodfellow et al., 2016; A. Zhang et al., 2023).

Another technique widely used in NNs to improve generalization by reducing overfitting is Dropout, first introduced by N. Srivastava et al. (2014). Dropout randomly sets a fraction of the connections between neurons to zero at each step, which helps to prevent co-adaptation of neuronal information. In other words, for every step, it randomly breaks up the connection between some of the neurons. This technique is especially useful for DNNs, where overfitting is a common issue due to the high number of parameters and the complexity of the model. By randomly dropping connections, Dropout forces the network to learn more robust features and prevents it from relying too heavily on specific neurons, improving generalization and reducing overfitting (Goodfellow et al., 2016; N. Srivastava et al., 2014; Szeliski, 2022).

Input-wise, batch normalization is used to normalize the input data to the network, ensuring that the data has normal distribution. This helps to stabilize the training process and accelerates convergence by ensuring that the input data is within a specific range. By normalizing the input data, batch normalization helps to prevent the network from becoming too sensitive to the input data's scale and distribution, improving its generalization performance and training efficiency (Bernard, 2021; Bjorck et al., 2018; Szeliski, 2022; A. Zhang et al., 2023). However, batch normalization has some drawbacks, such as the requirement of large batch sizes to accurately estimate the statistics, which can be computationally expensive and memory-intensive. It also introduces additional computation during training and may not perform well in online learning scenarios or when the batch size is small (Ba et al., 2016; Szeliski, 2022).

To tackle the drawbacks of batch normalization, layer normalization was introduced by Ba et al. (2016). This technique normalizes the activations of each layer across the feature dimension for each individual data point, rather than across the batch. This means that for each data point, the normalization is performed on the features within that single data point, ensuring that the mean and variance are calculated for each individual sample's features. By doing so, layer normalization stabilizes the training process and improves the model's generalization performance. It is particularly effective for scenarios with variable batch sizes, as it does not rely on batch statistics. This independence from batch size makes it more stable when batch sizes are small or when using sequential models. By ensuring consistency in the distribution of activations, layer normalization helps the model learn more robust features, reduces the risk of overfitting, accelerates the training process, and improves convergence, making it an essential technique for training DNNs effectively (Ba et al., 2016; Bernard, 2021; Szeliski, 2022).

2.5.7 Computational Demand

So all in all, training ANNs and DNNs is a complex process and involves a lot of different calculations. That is the reason why especially DL needs a lot of computational power. As a general rule, the deeper and more complex networks get, the more computational resources are needed to train the model effectively. The computational demand of training DNNs is a significant challenge in the field of DL, as it limits the scalability and accessibility of these models to researchers and practitioners with limited resources. However, advancements in hardware and software, such as distributed training, parallelization, and cloud computing, have helped to address these challenges and make DL more accessible to a wider audience. Furthermore, every model architecture incorporates a lot of optimization techniques, which not only increase the model's performance, but also its efficiency.

Still, the focus often lies on the model's performance rather than its computational efficiency, which can lead to resource-intensive models with great performance that are not practical for repeated real-world applications (Getzner et al., 2023; Lazzaro et al., 2023; Mehlin et al., 2023; Thompson et al., 2022). Therefore, it is essential to consider the computational demand of training DNNs and optimize the model architecture and training process even further to ensure that the model is efficient and scalable. This should be a key consideration when designing and training DNNs for real-world applications, especially in the field of remote sensing, where large datasets and complex models are common.

2.6 Deep Neural Networks for Computer Vision

The MLP architecture and its further developments have been presented as powerful tools for learning complex patterns in data, making them popular for a wide range of ML tasks. However, MLPs have limitations, especially when dealing with high-dimensional data like images, audio, and text, where spatial relationships are crucial. They treat input values, whether directly adjacent or on opposite sides of an image or a sentence, as equally related since every value is connected to every neuron in a fully connected layer. This approach overlooks the spatial relationships between these input values, which are critical for understanding images and text, leading to the loss of important information (Nielsen, 2015).

To address these limitations, more advanced DNN architectures have been developed. DNNs enhance ANNs by incorporating multiple successive layers of transformations, effectively making them "deep", thus the term for such networks (Goodfellow et al., 2016; Y. LeCun et al., 2015). Additionally, they incorporate new types of layers, tuned activation functions, and optimization techniques. These enhancements, when combined, can significantly improve the performance and capability of DNNs, making them powerful tools for an even wider range of complex ML and DL tasks (Bernard, 2021; Nielsen, 2015; Szeliski, 2022; A. Zhang et al., 2023).

In this section, some of the most important types of DNNs for CV are introduced, particularly CNNs and encoder-decoder architectures, along with residual networks. While there are many DNNs designed for various tasks, this study focuses on LULC-M, therefore, these are described as they address specific challenges in CV, especially object detection and semantic segmentation, with images as inputs.

2.6.1 Convolutional Neural Networks

Based on the MLP and improving it by introducing two new types of hidden layers, CNNs are specialized DNNs for CV tasks. They are designed to process multi-dimensional grid-like data, particularly images, by exploiting the spatial relationships between pixels. This makes them particularly well-suited for tasks like image classification, object detection, and image segmentation, where the input data has spatial structures that need to be preserved and learned (Y. LeCun et al., 2010; Szeliski, 2022).

In comparison to MLPs, the inputs to CNNs are not single pixels, but rather whole images. In such grid-like inputs, which are seen as 2nd-order tensors (images with one band) to simplify the explanations, every pixel in these images has a specific pixel value. In MLPs, every pixel would be fully connected to the next layer of neurons to classify each pixel. This would result in a huge number of parameters, which would be computationally expensive and inefficient. Furthermore, as mentioned above, the spatial relationships between the pixels would be omitted, disregarding critical information of the image. CNNs solve this problem by using convolutional layers, which apply a set of learnable filters to the input image. These filters slide (convolve) in small focus areas over the input, performing element-wise multiplications and summing up the results to produce feature maps. This process is known as convolution, and it helps in detecting local patterns, such as edges, textures, and shapes, irrespective of their position in the image. These are crucial for recognizing more complex structures in higher layers. As the network progresses, the depth of the convolutional layer increases and the layers get more coarse, enabling it to learn more abstract features like objects (Alzubaidi et al., 2021; Dumoulin & Visin, 2018; Nielsen, 2015; Szeliski, 2022; A. Zhang et al., 2023).

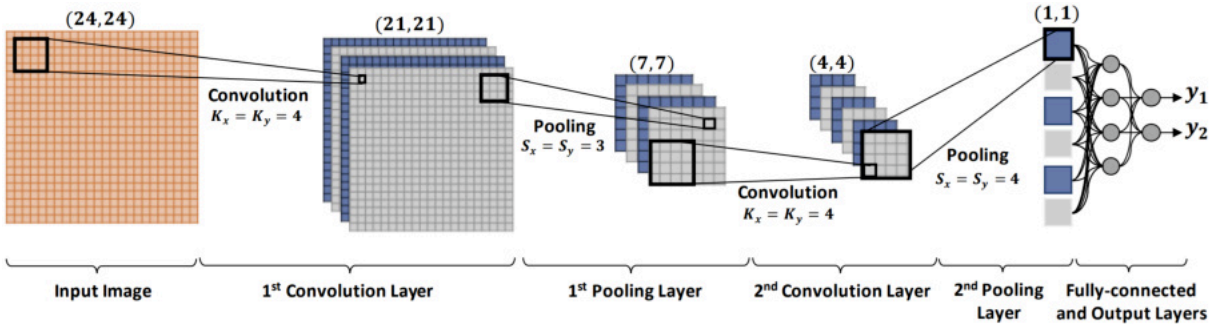


FIGURE 2.7: Graphical representation of the general architecture of CNNs (Moharram & Sundaram, 2023).

A typical CNN architecture consists of several key layers, including convolutional, pooling, and fully connected layers, often in multiple repetitions, as seen in figure 2.7. To clarify what CNNs are, Goodfellow et al. (2016) put it simply like this: "convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers" (Goodfellow et al., 2016, p. 326). The convolution operation is done in convolutional layers, therefore, they are the fundamental building blocks of CNNs.

Convolutional Layers

Convolution is best described with a graphical representation of the operation, as seen in figure 2.8. Using a specified and learnable filter, called kernel, the pixels of the image are grouped into patches of

the size of the kernel, in this case a 3×3 grid, as shown exemplary in the figure-like table 2.1. These groups function as the input to the subsequent hidden neurons, which usually also are pixels on a so called output feature map. To generalize, these neurons are called activations. As mentioned above, convolutional layers are not fully connected layers, so not every pixel is connected to every activation in the next layer. Instead, the kernel is applied to each of these groups, multiplying each pixel value with the corresponding kernel value. These processed pixel values are then connected to a activation of the subsequent layer by applying the kernel-weighted sum, shown as the dark colored values of the smaller grids in figure 2.8, effectively reducing the input values to this activation. These groups, which resemble the inputs to the activations, are called (local) receptive fields of the according activation (Alzubaidi et al., 2021; Dumoulin & Visin, 2018; Szeliski, 2022).

TABLE 2.1: The 3×3 kernel used for convolution in figure 2.8.

0	1	2
2	2	0
0	1	2

This process of applying the kernel-weighted sum is repeated for each pixel group in the image by sliding over the image horizontally and vertically, as depicted in figure 2.8. Different kernels can be used to form multiple output feature maps, as seen in figure 2.7 (Alzubaidi et al., 2021; Dumoulin & Visin, 2018; Szeliski, 2022). The input feature map with a spatial dimension of $24 \times 24 \times 1$ pixels (height, width, depth), for example, is convoluted four times with a kernel size of 4×4 each time, resulting in an output feature map of the spatial dimension of $21 \times 21 \times 4$ pixels. Using multiple kernels, which is equivalent to making the network wider, can help the network learn different features.

The behavior of a convolution layer can further be specified by additional parameters, including stride, padding, dilation, and grouping (Dumoulin & Visin, 2018; Szeliski, 2022).

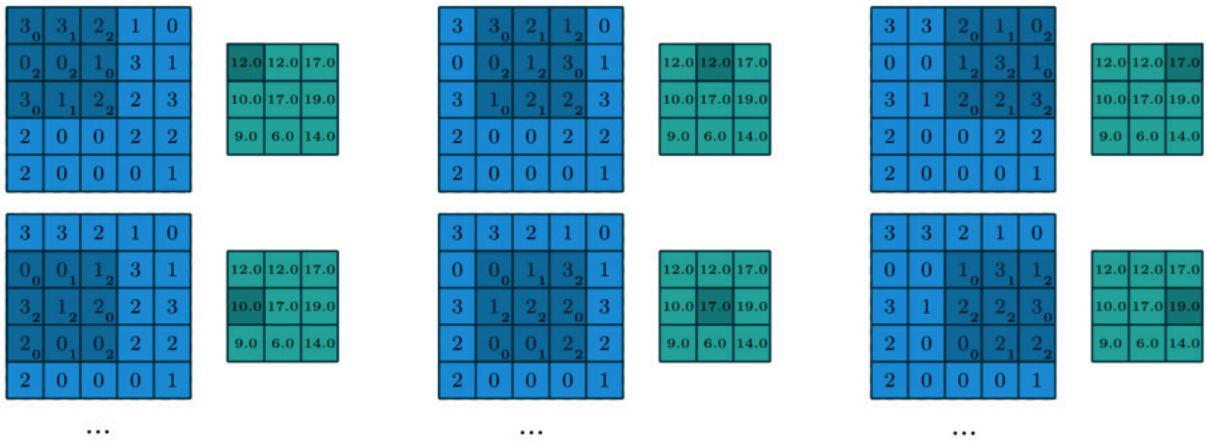


FIGURE 2.8: Graphical representation of the convolution operation. The input values of size 3×3 are multiplied by the kernel, denoted as subscript numbers in the input cells and shown in table 2.1. The result is a kernel-weighted sum of the inputs, resulting in an output feature map. Adapted from Dumoulin and Visin (2018).

Stride is the parameter that specifies the sliding movement over the image. It determines the step size of the kernel as it slides over the input feature map. A stride of 1 means that the kernel moves one pixel at a time, while a stride of 2 means that the kernel moves two pixels at a time. Stride affects the spatial dimensions of the output feature map, as it determines how many times the kernel is applied to the input feature map. It also determines how detailed the output feature map is by including selected information.

Padding is applied in two cases: first, to ensure that the output feature map has the same spatial dimensions as the input feature map, and second, to prevent the loss of information at the edges of the input feature map. Padding involves adding values around the input feature map, which allows the kernel to slide over the input without losing information at the edges, as shown in figure 2.9. The amount of padding added to the input feature map is determined by the size of the kernel and the desired spatial dimension of the output feature map. A usual choice is to add zeros around the input feature map, which is called zero padding.

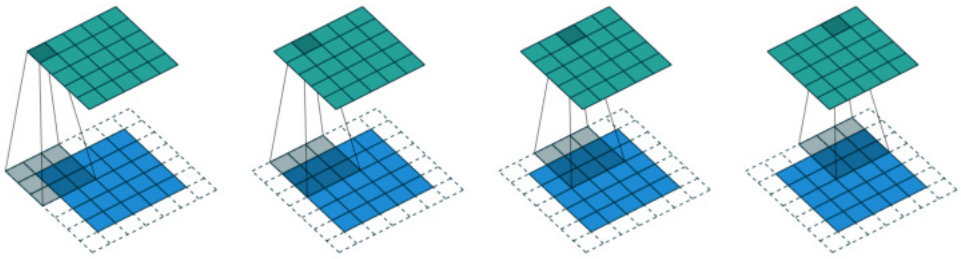


FIGURE 2.9: Padding operation during convolution. Around the edges of the input feature map, zeros are added to enlarge the output feature map and to include all information of the edges (Dumoulin & Visin, 2018).

Dilation enlarges the receptive fields by skipping columns and rows of the input feature map, as shown in figure 2.10. This can be effective when the images are large and include large details, as dilation needs less slides over the image to fully capture it without losing too much information.

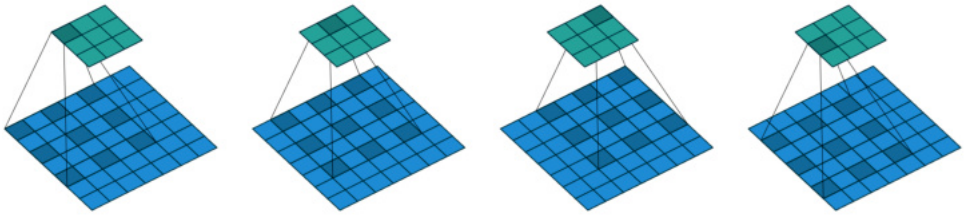


FIGURE 2.10: Dilation operation during convolution. The receptive field is enlarged by adding space in between the input pixels (Dumoulin & Visin, 2018).

Especially when the images have multiple channels, grouping can be used to reduce the computational load. Instead of applying a single kernel to all input channels, the channels are divided into groups, and each group is convolved with a separate set of kernels. This approach reduces the number of parameters and the amount of computation required, making the network more efficient.

Convolutional layers can be stacked to form deeper NNs, with each layer learning different features at different levels of abstraction. The output of the convolutional layers is typically passed through an activation function, such as ReLU (see section 2.4.1), to introduce non-linearity into the network. This

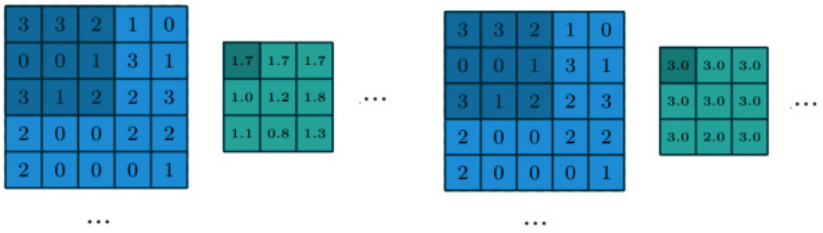
allows the network to learn complex patterns and relationships in the data, making it more powerful and expressive.

The convolution explained above is an example of a 2D convolution, but the process can be generalized to nD convolutions. For instance, in a 3D convolution, the kernels would be cuboids and would slide across the height, width, and depth of the input feature maps. The resulting output feature maps would also be cuboids, but could be flattened to a 2D matrix for further processing, depending on the requirements of the task at hand. This makes CNNs very flexible and scalable (Dumoulin & Visin, 2018).

Pooling Layers

The second important building block of CNNs are pooling layers, often following convolutional layers as shown in figure 2.7. Pooling operations reduce the spatial dimensions of feature maps in order to reduce the computational load and the number of parameters. Also, they help in reducing the sensitivity to noise and irrelevant details in the input data. Furthermore, by summarizing the presence of features in a particular region of the input, they effectively downsample the input representation, making the network less sensitive to small translations and distortions in the input data. This provides a form of translation invariance, meaning that small shifts in the input image will not significantly affect the pooled output. This helps in recognizing objects in varying positions within the image. Overall, pooling is an essential operation in CNNs additionally to convolutions, contributing to the model's ability to generalize well to new data by focusing on the most important features and reducing the effect of noise and minor variations. Pooling helps in making the network more efficient and robust (Gholamalinezhad & Khosravi, 2020; Y. LeCun et al., 2015).

Pooling works similar to convolution, only that it uses a different function with the aim to reduce the spatial dimensions of feature maps rather than "read" the input data. There are two main types of pooling, average and max pooling, although multiple other enhancements can also be added. In average pooling, shown in figure 2.11a, the average value of each patch is calculated. On the other hand, in max pooling, shown in figure 2.11b, the maximum value of each patch is selected (Dumoulin & Visin, 2018; Gholamalinezhad & Khosravi, 2020; Y. LeCun et al., 2015). The sizes of patches are adjustable as in convolutional layers, and the sliding parameters can be applied here as well.



(A) Average Pooling. The average of the patch is calculated.

(B) Max Pooling. Of the patch, the maximal value is selected.

FIGURE 2.11: Average (A) and max pooling (B) operations to reduce the spatial dimensions of feature maps. Adapted from Dumoulin and Visin (2018).

For instance, in figure 2.7, a pooling layer is applied after the convolutional layer, reducing the spatial dimensions of the feature maps from $21 \times 21 \times 4$ to $7 \times 7 \times 4$ by using a filter of size 3×3. After the

second convolution layer, another pooling layer is applied, further reducing the spatial dimension of the feature maps to $1 \times 1 \times 1$, which then can be fed to a fully connected layer.

Fully Connected Layers

Usually at the end of CNNs, the downsampled feature maps are flattened to a 1D vector and passed through fully connected layers, as shown in figure 2.7. These layers are similar to the hidden layers in MLPs, connecting every neuron in one layer to every neuron in the next layer, leading to a final output layer for classification or regression tasks, as explained for MLPs in section 2.4.2 (Alzubaidi et al., 2021).

CNNs for Segmentation

Based on the CNN, the Fully Convolutional Network (FCN) was developed by Long et al. (2015) in order to extend the CNN architecture to perform pixel-wise classification. The FCN uses convolutional layers to extract features from the input image and upsampling layers to generate a dense prediction map. This allows the network to produce pixel-wise class labels for the entire image, making it suitable for segmentation. FCNs has been widely used in various CV tasks, including object detection, image segmentation, and image classification, and has been the foundation for many other advanced architectures (Long et al., 2015; Minaee et al., 2022; Noh et al., 2015).

2.6.2 Encoder-Decoder Architectures

Even though FCNs are suitable for image segmentation and have shown great performance, they have two main limitations. First, the networks have a predefined fixed-size receptive field based on the chosen kernel size, which can lead to fragmented or mislabeled objects that are substantially larger or smaller than the receptive field. Second, the detailed structures of objects are often lost or smoothed because of sparse label maps and the simplified upsampling procedure (Badrinarayanan et al., 2017; Y. Liu et al., 2018; Noh et al., 2015).

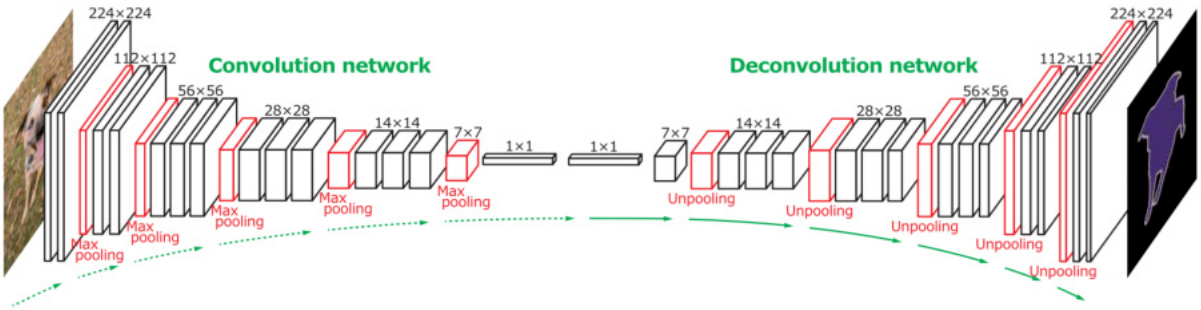


FIGURE 2.12: Graphical representation of the Deep Deconvolution Network architecture, an Encoder-Decoder for image segmentation proposed by Noh et al. (2015).

To address these limitations, encoder-decoder architectures have been developed, which consist of two main parts: an encoder and a decoder, as shown in figure 2.12. The encoder part is typically a CNN that processes the input image and extracts features at different levels of abstraction. The decoder part of the architecture is usually a series of upsampling and deconvolutional layers, reverting the feature extraction process to generate an output segmentation mask with the same spatial dimensions as the

input image. This architecture is particularly well-suited for semantic segmentation tasks, where the goal is to assign a class label to each pixel in the image (Minaee et al., 2022; Noh et al., 2015).

The key innovation, the decoder, utilizes reverting operations including unpooling and deconvolution in order to reconstruct to initial spatial relationships between pixels. In unpooling, stored locations of maximum activations selected during the max pooling operation in the encoder are employed to place each activation back to its original pooled location. This unpooling strategy is particularly useful to reconstruct the structure of input object. After that, the feature map is enlarged, but sparse, as zeros have been used to fill in the feature maps. Deconvolution densifies feature maps by applying convolutions as well. However, contrary to convolutional layers, which connect multiple input activations within the receptive field to a single activation, deconvolutional layers associate a single input activation with multiple outputs. The learned filters act as bases for reconstructing an input object's shape (Badrinarayanan et al., 2017; L.-C. Chen et al., 2015; Noh et al., 2015).

Hierarchically arranged, these layers capture varying levels of detail, with lower (deeper) layers focusing on the object's overall shape and higher layers encoding class-specific fine details. This structure allows the network to incorporate class-specific shape information directly into semantic segmentation, a feature often overlooked in purely convolutional approaches (L.-C. Chen et al., 2015; Noh et al., 2015).

2.6.3 Residual Networks

CNNs have been the backbone of many advancements in CV tasks, designed to learn hierarchical representations of data through a series of convolutional layers, which capture spatial hierarchies by applying filters that detect features such as edges, textures, and more complex patterns. However, as deeper and more powerful networks were created, a significant challenge was encountered: the vanishing gradient problem, where gradients become too small to effectively update the parameters in the early layers. This leads to slow or stalled training and can prevent the network from learning effectively (He et al., 2016; A. Zhang et al., 2023).

To overcome this limitation, Residual Networks (ResNets) were introduced by He et al. (2016). They address the vanishing gradient problem by incorporating shortcut connections, also known as skip or residual connections, which allow the network to bypass one or more layers. This innovative architecture enables the creation of much deeper networks by ensuring that gradients can flow more easily through the network during backpropagation.

To enable this, the inputs to activations are added after the activations to the weighted sums before being fed into the activation functions, as depicted in figure 2.13. Using this, the parameters that need to be learned and adapted are reduced. "In a regular block ([figure 2.13] left), the portion within the dotted-line box must directly learn the mapping $f(x)$. In a residual block ([figure 2.13] right), the portion within the dotted-line box needs to learn the residual mapping $g(x) = f(x) - x$, making the identity mapping $f(x) = x$ easier to learn" (A. Zhang et al., 2023, p. 313).

In summary, while CNNs laid the groundwork for understanding and processing visual data, the introduction of ResNets marked a significant advancement by enabling the training of much deeper networks. This innovation has not only improved performance on a wide range of CV tasks, but also

inspired further developments in the field of DL (Goodfellow et al., 2016; Szeliski, 2022; A. Zhang et al., 2023).

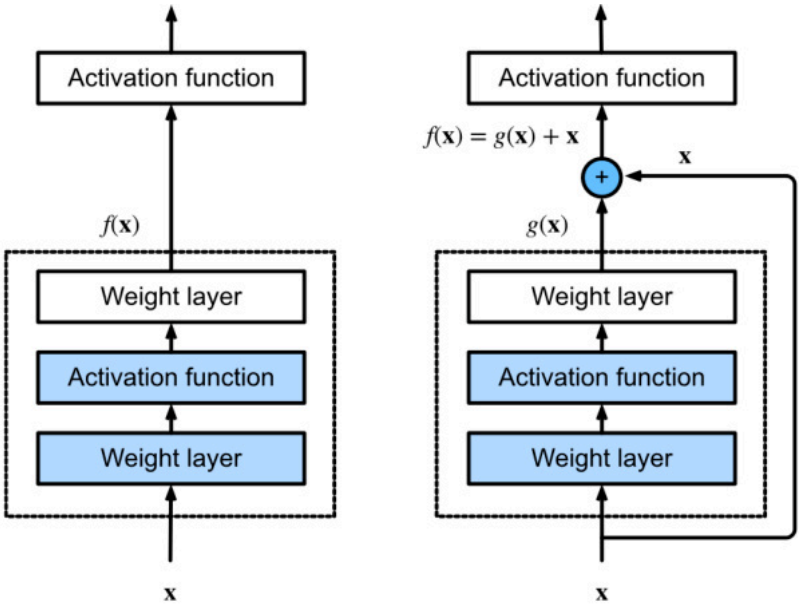


FIGURE 2.13: Residual (right) vs. regular block (left) in a NN. "In a regular block (left), the portion within the dotted-line box must directly learn the mapping $f(x)$. In a residual block (right), the portion within the dotted-line box needs to learn the residual mapping $g(x) = f(x) - x$, making the identity mapping $f(x) = x$ easier to learn" (A. Zhang et al., 2023, p. 313).

2.7 Spatial Context & Attention in Computer Vision

In the section above, some of the most important DL architectures and techniques for CV have been introduced. Regarding the topic of this study, the inclusion of contextual information into a model, the key topic of context and its suitable implementation have to be highlighted as well. Therefore, the following section will introduce the concepts of context and attention in DL and how they can be used to improve the performance of DNNs for CV tasks. This functions as a background to the SegFormer, the model utilized by the study's experiment framework, the LULC Utility.

In DL, context refers to the additional information that surrounds a particular piece of data, assisting to interpret and provide a fuller understanding of that data. In CV and semantic segmentation, the spatial context of features is crucial for accurately interpreting visual scenes. For instance, in an image of a city street, the context includes not just the individual pixels or segments but also their relationships to each other. Recognizing a car involves understanding the context provided by the road, nearby buildings, pedestrians, and other cars. This contextual information helps to distinguish between objects and improve the accuracy of segmentation (Ding et al., 2020; J. Zhang et al., 2022).

Traditional CNNs capture context through the aforementioned receptive fields. Convolution assumes that nearby pixels are more important than far away pixels, weighting close locality more. As the network grows deeper, the receptive fields grow. This expansion means that activations in deeper layers have a wider view of the input image, allowing the network to capture and integrate information from

larger spatial regions. Thus, more complex patterns or features are recognized. But, it comes at a cost of low spatial resolution for the segmentation result, in which blurry class boundaries and the loss of object details become a challenge. Also, receptive fields are limited by their fixed size and locality in each layer, depending on the specification by the model architecture. Therefore, the relationship between distant parts of the image might still be poorly captured, especially in complex scenes requiring long-range dependencies. Dilation enlarges the receptive field more efficiently than other techniques, but still comes with drawbacks like lower resolution or omission of pixels due to the inflation of the kernel with zeros (Cordonnier et al., 2020; de Santana Correia & Colombini, 2022; Y. Liu et al., 2018; Luo et al., 2017; Ma et al., 2019; Szeliski, 2022; Xie et al., 2021).

Tackling these challenges initially in language models and subsequently in CV tasks, attention mechanisms were introduced. Attention is a concept deriving from cognitive psychology, summarized by de Santana Correia and Colombini (2022) as: "Attention is a behavioral and cognitive process of focusing selectively on a discrete aspect of information, whether subjective or objective, while ignoring other perceptible information [...], playing an essential role in human cognition and the survival of living beings in general" (de Santana Correia & Colombini, 2022, p. 6038).

Its development and implementation in DL models is a major research topic because it enables models to focus on specific parts of the input data that are most relevant to the task at hand, effectively weighting the importance of different input elements and their context. It allows the model to dynamically adjust the focus and available processing resources on the most relevant parts of the input data, regardless of their distance from each other. This enhances the model's ability to capture dependencies and relationships, especially in sequences or complex scenes, effectively understanding the relationship between features and their important context (Alhichri et al., 2021; de Santana Correia & Colombini, 2022).

For this study and the spatial contextual information inclusion approach, attention is a key concept and critical for success. The SegFormer model, which is used in the experiment framework, the LULC Utility, is based on the Transformer architecture, which is known for its attention mechanisms. The following section will introduce attention mechanisms in DL and how they can be used to improve the performance of DNNs for CV tasks.

Most current state-of-the-art models in DL for NLP or CV use some form of attention. To understand why attention mechanisms are so effective in DL, it is essential to recognize that NNs fundamentally serve as function approximators. The ability of NNs to approximate various functions depends significantly on their architecture. Traditional NNs function through sequences of matrix multiplications and element-wise non-linearities, with interactions between input or feature vector elements limited to addition. Attention mechanisms revolutionize this process by generating a mask that multiplies the features. This seemingly simple adjustment substantially expands the spectrum of functions that NNs can approximate accurately. As a result, it enables entirely new applications and greatly enhances performance on complex tasks (Alhichri et al., 2021; Cordonnier et al., 2020).

In image segmentation, precise delineation of object boundaries is essential. Attention mechanisms can refine these boundaries by focusing on the relevant context around edges, improving the segmentation quality. To do this, they allow the model to consider the entire image when determining the

importance of each pixel or region. This global perspective ensures that even distant parts of the image can influence the segmentation decision, enhancing accuracy. Unlike fixed receptive fields, attention mechanisms can adjust focus based on the input data. For instance, if a scene contains a small but crucial detail (like a traffic light in the background), attention can highlight this detail and incorporate it into the segmentation process (L.-C. Chen et al., 2016; de Santana Correia & Colombini, 2022).

Attention is often described by using the analogy of dictionaries, a key data structure in multiple programming languages. They consist of key-value pairs, denoted as k_i, v_i, and a query q, which can be used to return the value v_i associated with a specific key k_i. In NNs, these three instances are vectors. The query vector q represents the specific input feature that the model should measure its importance of. The key vectors k_i represent all relevant inputs in the sequence that the model should consider when processing the query, so the relational data which shows how important the query is. The value vectors v_i are the actual information associated with the keys that the model will use to form its final output. By using the query to search the keys, the model can retrieve the most relevant information from the key-value pairs, allowing it to focus on the most important parts of the input data (Brauwers & Frasincar, 2023; Han et al., 2023; Szeliski, 2022; Vaswani et al., 2017; A. Zhang et al., 2023).

2.7.1 Self-Attention & Scaled Dot-Product Attention

Scaled Dot-Product Attention (SDPA) and self-attention are often used interchangeably, although self-attention is the broader concept, which incorporates SDPA. Self-attention is the idea of each feature in a dataset attending to all other features in the same dataset and estimating the relevance of one feature to other features. When implemented, self-attention allows each feature to focus on other features in the dataset to build a richer representation of the features' context.

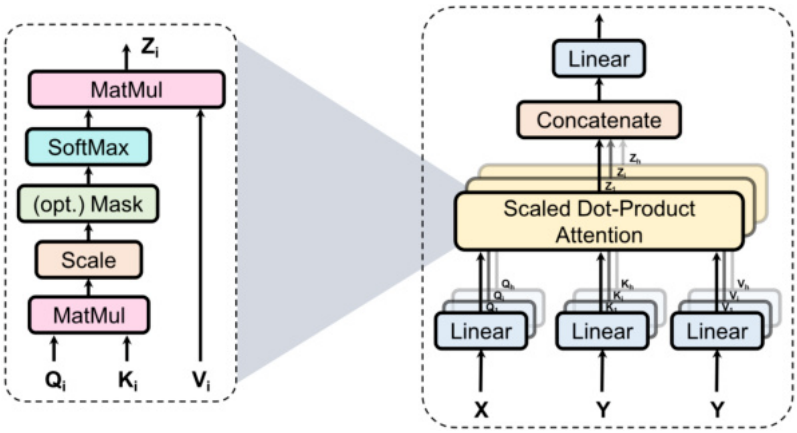


FIGURE 2.14: General attention mechanisms. Left: Scaled Dot-Product Attention. Right: Multi-Head (Self-)Attention. Multi-Head (Self-)Attention consists of several attention layers running in parallel (Y. Liu et al., 2024; Vaswani et al., 2017).

Figure 2.14 shows two common implementations of self-attention, whereas the left one, SDPA, is part of the right one. As shown in the figure, the inputs to the attention layer are not vectors, but rather multi-dimensional tensors consisting of these vectors, denoted as Q, K, and V, according to query, key, and value tensor, respectively. However, the attention works channel-wise, so the explanations will focus on the channels of the tensors, which are considered matrices.

The first step of an attention layer is to calculate the attention scores S, which measure how relevant each key is to the query. This is typically done using a dot product between the query matrix Q and the key matrix K with

$$S(Q, K) = Q \cdot K^T, \tag{2.11}$$

also called matrix multiplication, denoted in the figure as "MatMul", whereas K^T is the transposed key matrix. Transposing is necessary to generate a dot product of Q and K, assuming Q is a $n \cdot d_k$ matrix and K a $m \cdot d_k$ matrix, respectively, resulting in a $n \times m$ matrix when multiplying Q with the transposed matrix K^T.

The output of the attention function is a weighted sum of the value matrix V, based on Q and K. This dot product results in scores that can become very large when the dimensionality of the key vectors, d_k, is high. To prevent this and ensure gradient stability, the scores are normalized by dividing them by the square root of d_k, as in

$$S_{norm}(Q, K) = \frac{S(Q, K)}{\sqrt{d_k}}, \tag{2.12}$$

denoted in the figure as "Scale". After this step, an optional mask can be applied to restrict the attention only to past and current tokens, important for sequential inputs like in NLP. Nonetheless, as explained similarly in section 2.4.2, these normalized scores are then passed through a softmax function ("SoftMax") to obtain the probabilities P with

$$P(Q, K) = \text{softmax}(S_{norm}(Q, K)) = \frac{\exp(S_{norm}(Q, K)_i)}{\sum_j \exp(S_{norm}(Q, K)_j)}. \tag{2.13}$$

Finally, the value matrix is multiplied by the softmax score to obtain the weighted sum Z of the value matrix, again denoted in the figure as "MatMul", whose result is the output of the attention layer. This can be expressed as

$$Z(Q, K, V) = P(Q, K) \cdot V, \tag{2.14}$$

where vectors with larger probabilities receive additional attention from the subsequent layers as they have higher weights, meaning higher importance for the query.

This four-step process can be unified into a single function (Vaswani et al., 2017) as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \cdot V. \tag{2.15}$$

This process in the attention layer is called SDPA, as it scales the dot product of the query and key vectors by the square root of the key vector dimensionality. This ensures that the gradients remain stable during training and that the model can effectively learn the attention weights. Figure 2.15 shows the attention layer for multi-dimensional feature maps, as used in CV tasks. Note that 1×1 convolutions are used, which are reducing dimensionalities by applying a kernel of size $1 \times 1 \times d_k$ to the input feature maps (Cordonnier et al., 2020; Pan et al., 2022). Because only one head is used, this approach is called single-head (self-)attention.

This section was summarized from Bahdanau et al. (2014), Brauwers and Frasincar (2023), Han et al.

(2023), Khan et al. (2021), Y. Liu et al. (2024), Szeliski (2022), Vaswani et al. (2017), and A. Zhang et al. (2023).

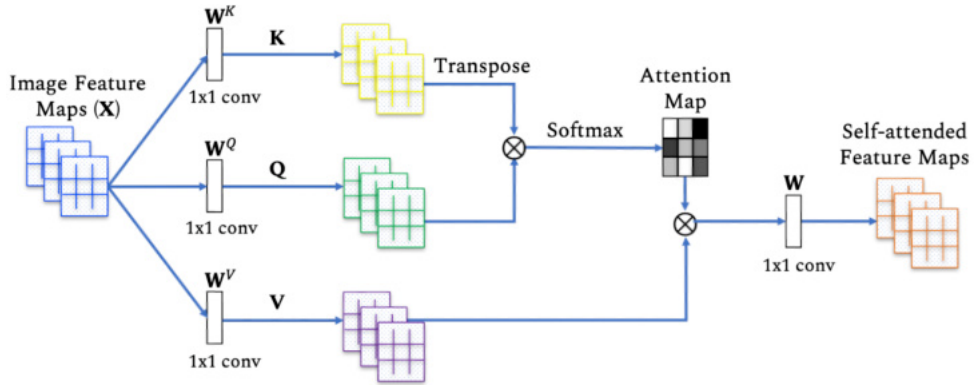


FIGURE 2.15: An example self-attention block used in CV. Given the input sequence of image features, the triplet of key, query, and value is calculated followed by attention calculation and applying it to reweight the values. A single head is shown here and an output projection W is finally applied to obtain output feature maps with the same dimension as the input feature maps (Khan et al., 2021).

2.7.2 Multi-Head (Self-)Attention

In order to capture multiple relationships between multiple features, enhancing the self-attended feature maps, Multi-Head (Self-)Attention (MHA) is used. This mechanism is achieved by running multiple SDPA layers in parallel as shown in figure 2.14, each with its own set of learnable parameters. Note that in the figure, the inputs are denoted as X and Y, which represent the possibility of using different input datasets. If $X = Y$, self-attention is used. If $X \neq Y$, then cross-attention is applied, where the queries come from one set X and the keys and values come from another set Y (Bi et al., 2021; Khan et al., 2021; Y. Liu et al., 2024).

In a MHA layer, each head calculates the SDPA for its own set of queries, keys, and values. The outputs of these attention layers are concatenated and passed through a final linear transformation W^O to generate the output of the multi-head attention layer, as in

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \ldots, \text{head}_h) \cdot W^O \\ \text{where head}_i &= \text{Attention}(Q \cdot W_i^Q, K \cdot W_i^K, V \cdot W_i^V), \end{aligned} \tag{2.16}$$

where W are learnable parameters of linear transformations multiplied with the attention calculation per matrix and h is the number of attention heads (Vaswani et al., 2017).

This mechanism allows the model to capture multiple relationships between features, enhancing the model's ability to learn complex patterns and relationships in the data. The number of layers or heads is a hyperparameter that can be adjusted to balance the model's capacity and computational complexity, although in the original work of Vaswani et al. (2017), they employed 8 parallel attention heads (Han et al., 2023; Khan et al., 2021; Y. Liu et al., 2024; Vaswani et al., 2017; A. Zhang et al., 2023).

2.7.3 Positional Encoding

During the attention process, the position of activations is not considered, as all inputs are processed simultaneously, but only the relationships between them. The position of features, however, is of high importance for recognizing larger objects and combining context information features to the activations. To address this, positional encoding, an additional positional vector, is added to the inputs, allowing the model to capture the positional information of activations. Positional encoding is often implemented using sinusoidal functions, enabling the model to learn relative positions and generalize to longer sequences during inference. Positional encoding can either be fixed or learned and relative, adapted to enhance performance for specific tasks and inputs (Cordonnier et al., 2020; Han et al., 2023; He et al., 2016; Khan et al., 2021; Y. Liu et al., 2024; Vaswani et al., 2017; A. Zhang et al., 2023).

2.8 Attention-Based Architectures for Computer Vision

Transformers rely solely on attention layers, bypassing the conventional use of convolutions and other techniques found in models like CNNs and similar. Consisting of a Transformer encoder and decoder, they have been widely used in NLP tasks, achieving state-of-the-art performance on various benchmarks, even serving as the basis for the GPT series (Tzepkenlis et al., 2023; A. Zhang et al., 2023). Transformers have also been adapted for CV tasks, leading to the development of Vision Transformers (ViTs), which have shown promising results on image classification, object detection, and semantic segmentation tasks (Bi et al., 2021; Han et al., 2023; Khan et al., 2021; Y. Liu et al., 2024). Khan et al. (2021) give an overview over a plethora of models utilizing self-attention for CV. They split all the models they reviewed into the two categories single-head self-attention and MHA, of which the latter are also denoted as ViTs. The Transformer architecture, introduced by Vaswani et al. (2017), is the foundation for many of the models reviewed by Khan et al. (2021) and also of the SegFormer, the model utilized by the research framework of this study, the LULC Utility.

2.8.1 Vision Transformer

Given the advancements achieved by Transformers in the realm of NLP in the recent years since their introduction in 2017, there has been an increasing interest to explore their potential within the scope of CV (Khan et al., 2021). Multiple models have been developed for this task, as Cordonnier et al. (2020) proposed that attention could replace convolutions completely, introducing a prototype called the Fully Attentional Network. However, none had a similar impact as the ViT, introduced by Dosovitskiy et al. (2020) in 2020. It functions as an extension of the Transformer model adapted for image classification tasks. The characteristic feature of the ViT is that images are split into patches and fed linearly into the model rather than as single pixels. When trained on mid-sized datasets, the ViT has shown modest accuracy compared to CNNs. But trained on large datasets, the ViT outperformed CNNs clearly (Bi et al., 2021; Dosovitskiy et al., 2020; Khan et al., 2021).

Figure 2.16 shows the architecture of the ViT. The key part is the Transformer Encoder, which is a combination of an attention block as described above and an MLP. The ViT disregards the Transformer decoder and replaces it with a standard MLP head as decoder. Because Transformers are mainly developed for NLP tasks with sequential data, the idea for the ViT was to split the input image so that it

resembles a sequence of inputs. However, treating every pixel as input is not efficient, therefore, the image is split into 16×16 non-overlapping image patches. These are flattened, linearly transformed into fixed-size vectors, enriched with positional encoding vectors, and finally fed into the Transformer encoder. When used for image classification tasks, an extra learnable classification vector $[class]$ is added during these preprocessing steps, which acts as a representative vector for the entire image (Han et al., 2023; Khan et al., 2021; Y. Liu et al., 2024; Park & Kim, 2022; Szeliski, 2022).

The encoder is based on the encoder block of the Transformer, as it, as well, consists of multiple blocks of normalization, MHA, and FFN layers. When the preprocessed image patches are fed into the encoder, layer normalization, as explained in 2.5.6, is applied, and then they are fed into a MHA layer. Residual connections, as explained in 2.6.3, are used to counter the vanishing gradient problem. The output of the MHA layer is again layer normalized before passed through a FFN layer, specifically an MLP, with another residual connection (Dosovitskiy et al., 2020; Geva et al., 2021; Y. Liu et al., 2024; Park & Kim, 2022; Szeliski, 2022; A. Zhang et al., 2023).

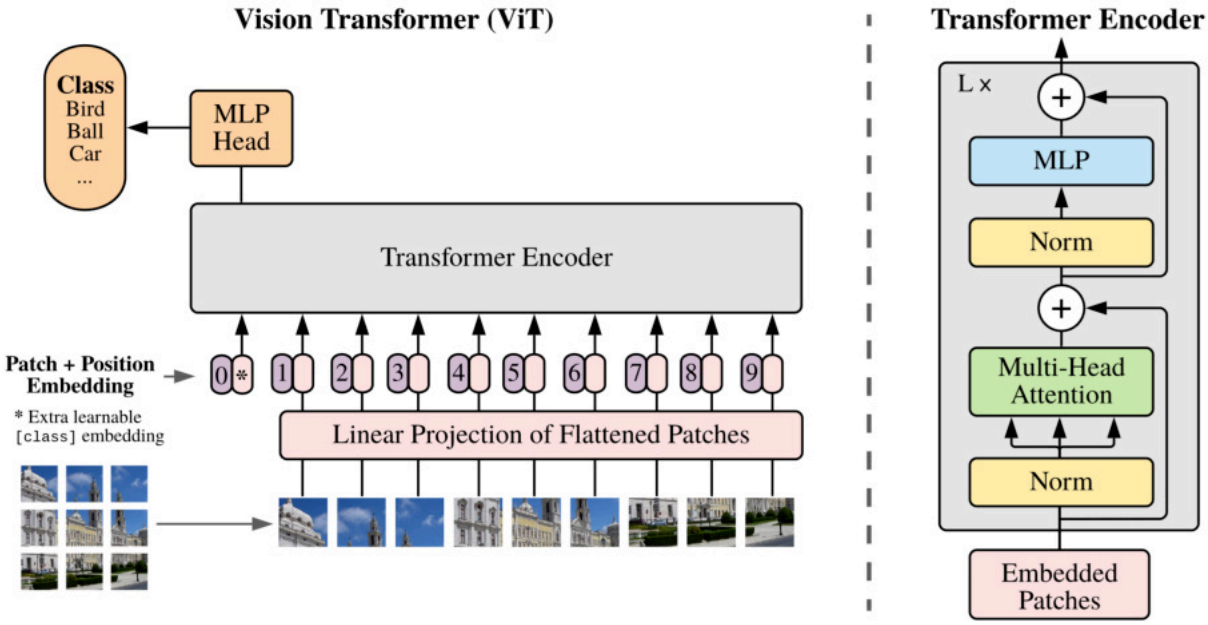


FIGURE 2.16: Graphical representation of the architecture of the ViT (Dosovitskiy et al., 2020).

The output of the Transformer encoder block is passed through an MLP classification head, which is a simple MLP with a final Gaussian Error Linear Unit (GELU) softmax activation function, an adapted ReLU, to predict the class of the input image. The ViT has shown state-of-the-art performance on various image classification benchmarks, demonstrating the potential of the Transformer architecture for CV tasks (Dosovitskiy et al., 2020; Geva et al., 2021; Y. Liu et al., 2024; Park & Kim, 2022; Szeliski, 2022).

2.8.2 Follow-Ups of the Vision Transformer

Despite its strong performance, ViT has two main limitations: first, it produces low-resolution features on a single scale instead of multi-scale ones, and second, it has high computational costs when processing large images (Xie et al., 2021). Building a base for many follow-ups, the ViT can be adapted

to tackle these limitations and be made more efficient and performant by changing the decoder head from an MLP to different other blocks, like using a Transformer decoder block or a CNN, and adapting the encoder by implementing additional components, such as convolutional layers, more efficient attention mechanisms, different positional encodings, and hierarchical structures (Han et al., 2023; Khan et al., 2021).

One milestone in enhancing the ViT and adapting it to dense prediction tasks was the Pyramid Vision Transformer (PVT) by W. Wang et al. (2021), introducing the first hierarchical design for ViTs with progressive shrinking spatial-reduction attention. This enhances the detection of different scaled features, similar to the architectures of CNNs and encoder-decoder models. Another one was the Swin Transformer by Z. Liu et al. (2021), which uses a shifting windows approach for splitting images in changing square-shaped blocks of patches and merging them afterwards, resembling block-wise convolution with a Transformer. The Segmentation Transformer (SETR) by Zheng et al. (2021) demonstrated the suitability of pure Transformer blocks without convolutions or resolution reduction for semantic segmentation, building the base for further follow-ups in this field (Han et al., 2023; Khan et al., 2021; Xie et al., 2021).

As the publication dates of these papers show, applying Transformer architectures to CV is a very current research field, leaving a lot of development potential for the future, as stated by Digra et al. (2022), Moharram and Sundaram (2023), Tzepkenlis et al. (2023), and Xie et al. (2021). That is why currently a plethora of adapted and enhanced architectures are published, aimed towards tackling various tasks, sometimes generally enhancing the architecture, sometimes tackling very specific use cases.

Evolving the ViT and some of its follow-ups, especially the PVT and the SETR, into a more flexible and efficient direction, the SegFormer was introduced also in 2021 by Xie et al. (2021). It is the model utilized by the LULC Utility in the experiment framework of this study, therefore, the following section gives an overview of the SegFormer and its architecture.

2.8.3 SegFormer

All background information given above, from perceptrons over training DNNs, convolutional layers, and attention mechanisms, to Transformers and the ViT, leads to the SegFormer, proposed by Xie et al. (2021). It incorporates advancements in DL of the last decades in one model architecture, aiming at providing an efficient and flexible way to tackle various segmentation tasks. Xie et al. (2021) demonstrate that their model performs equally well or even better in semantic segmentation than others while using significantly fewer parameters, thus being more efficient in terms of computational resources. Y. Chen et al. (2023), Lin et al. (2023), and Tzepkenlis et al. (2023) further show that due to its adaptability and performance, the SegFormer is a strong competitor to other DL models for CV and also for LULC-M. This makes the LULC Utility with the incorporated SegFormer a suitable candidate for the integration of a road network into the feature space in order to assess the impact of spatial contextual information on the model's performance and resource consumption.

The SegFormer is a robust and efficient semantic segmentation framework with a positional-encoding-free hierarchical Transformer encoder and a lightweight All-MLP decoder that "yields a powerful representation without complex and computationally demanding modules" (Xie et al., 2021, p. 2). The

hierarchical Transformer encoder with four stages is based on the PVT encoder and is referred to as Mix Transformer (MiT) (Khan et al., 2021; Y. Liu et al., 2024). Xie et al. (2021) state that they extend the ViT and SETR by adapting the encoder and decoder, a novel approach, as normally only mainly the Transformer encoder of the ViT is considered.

Improvements Over Predecessors

The SegFormer has some advantages over other ViTs, as stated by Xie et al. (2021) themselves, comparing the SegFormer primarily to the SETR. First, the training process is more efficient, as a much smaller training dataset is used for pre-training. Also, the encoder is smaller than the one implemented in the ViT and can capture both high-resolution coarse and low-resolution fine features due to its hierarchical structure in four stages. Most ViTs at this point, including the SETR, can only generate low-resolution feature maps. The omission of the positional encoding in the encoder is another advantage, as it allows the model to adapt to arbitrary image resolutions without impacting the performance. ViT and SETR use fixed shape positional encoding which decreases the accuracy when the inference resolution differs from the training resolution, as interpolation has to be used to make up the different resolutions. The All-MLP decoder takes advantage of the Transformer blocks which apply both local and global attention based on their hierarchy. The decoder aggregates these attentive features, combining local and global attention, resulting in powerful multi-scaled representations. Additionally, the decoder is simpler and less computationally demanding than the one implemented in the SETR, leading to a negligible computational overhead. Comparing the performance and efficiency of the SegFormer with the SETR, the Swin Transformer, the PVT, and a FCN, alongside others, it is safe to say that the SegFormer is able to outperform them all while being more efficient, achieving higher Intersection over Union (IoU) scores while using less parameters (Lin et al., 2023; Tzepkenlis et al., 2023; Z. Wang et al., 2023; Xie et al., 2021).

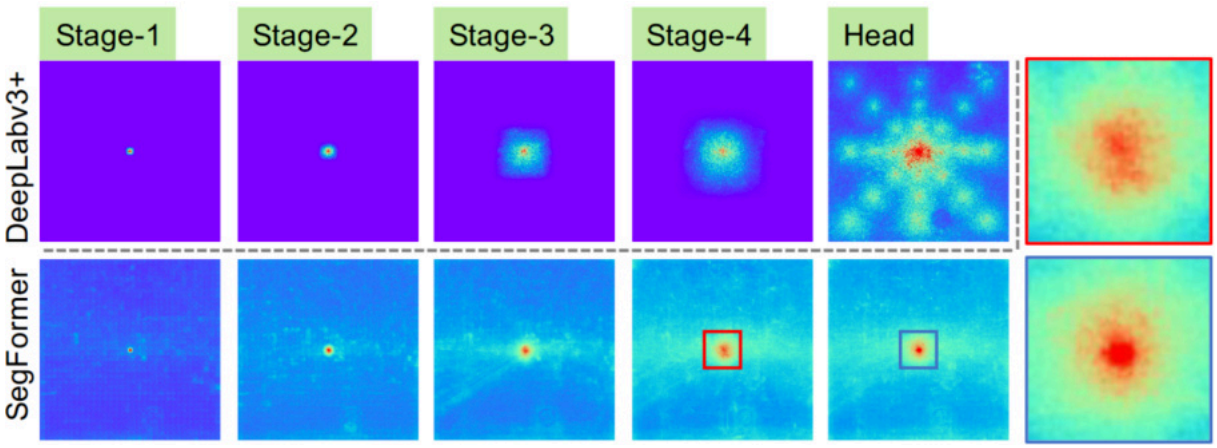


FIGURE 2.17: Effective receptive field analysis of all stages of the SegFormer compared to DeepLabv3+, average over 100 images from the Cityscapes benchmark. The boxes to the far right are zoomed-in representations of Stage-4 and the head of the SegFormer, respectively (Xie et al., 2021).

In regard to the task at hand, which focuses on the contextual information of spatial features, Xie et al. (2021) present an effective receptive field analysis, showing that the receptive field of the SegFormer is much larger without being complex, resembling "convolutions at lower stages, while able to output

highly non-local attentions that effectively capture contexts at Stage-4" (Xie et al., 2021, p. 5). The SegFormer takes advantage of the Transformer block as it generates both global and highly local attention at the same time, shown in the different phases of figure 2.17. These are merged in the decoder, where it "renders complementary and powerful representations by adding few parameters" (Xie et al., 2021, p. 5). Seen in the colored boxes to the far right in the figure, it is observable that the decoder produces a stronger local attention while keeping the same global attention level (Xie et al., 2021). This is useful for this study, as the SegFormer "understands" belonging spatial contextual information to features over a larger context area.

Architecture

Figure 2.18 shows the architecture of the SegFormer, emphasizing its simplicity, as it only has two main computation blocks, the MiT block in the encoder and the MLP in the decoder, shown broken down below the architecture. As in the ViT, before feeding the MiT with input tensors, the images have to be transformed into input sequences to extract features from them. This is done via Overlap Patch Embeddings, which split the images into multiple overlapping patches of size 4×4, contrary to 16×16 by the ViT, generating a vector for each of them (Lin et al., 2023; Z. Wang et al., 2023; Xie et al., 2021). These image patches are then fed into the MiT as the aforementioned Q, K, and V tensors. The MiT consists of three submodules, the Efficient MHA, the Mix-FFN, and Overlap Patch Merging.

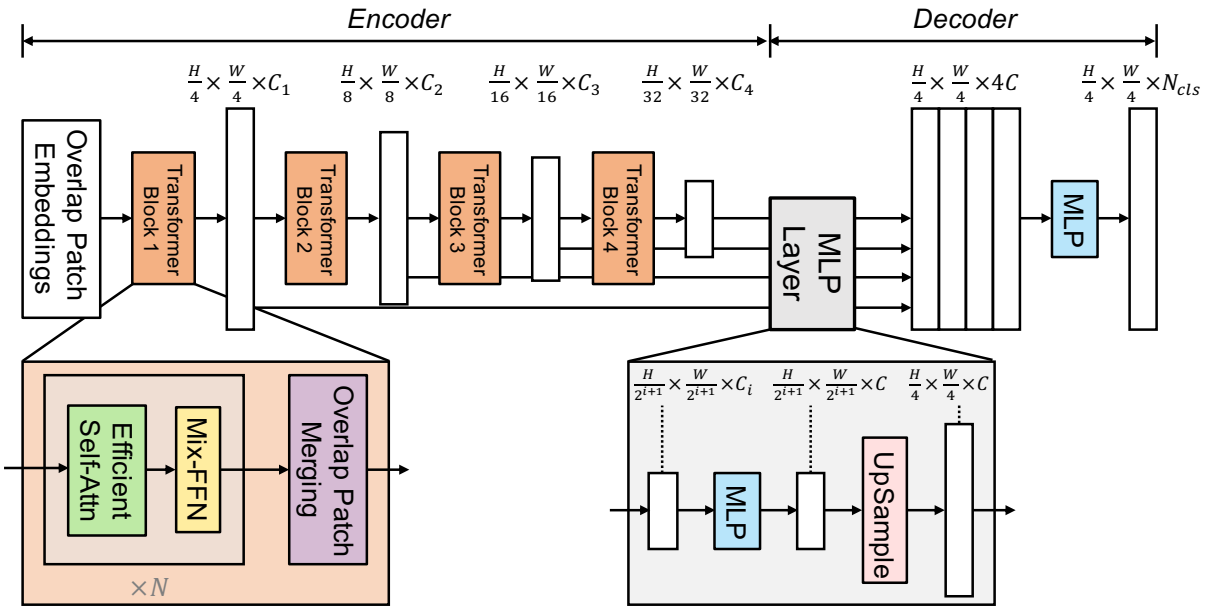


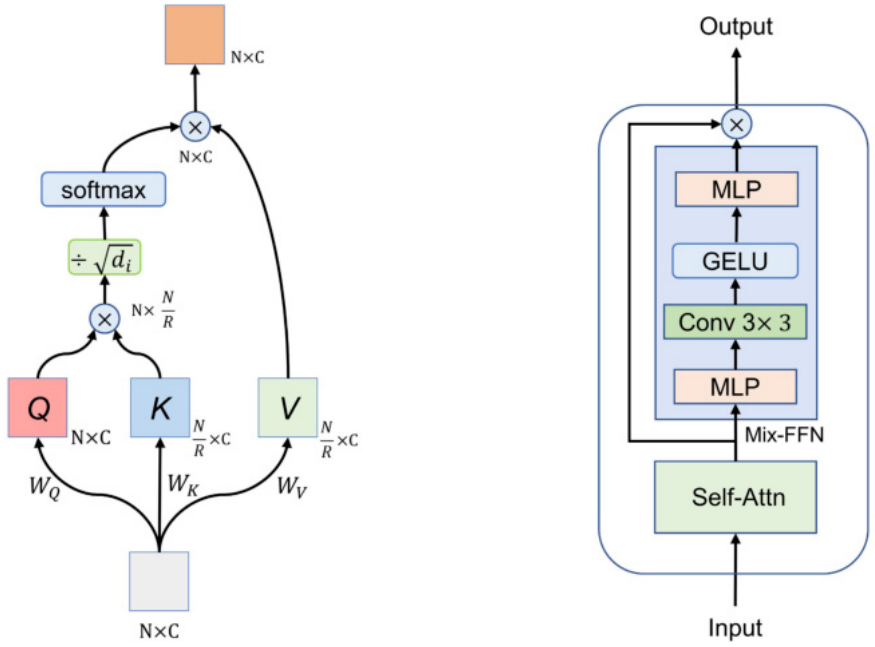
FIGURE 2.18: Graphical representation of the architecture of the SegFormer by Xie et al. (2021). MiT block and MLP layer are broken down below the architecture.

The efficient MHA block is similar to the MHA with the implemented SDPA explained above and shown in figure 2.14. Compared to the traditional SDPA, it reduces computational load and the number of parameters, enhancing runtime efficiency and mitigating overfitting. The MHA is a computational bottleneck as each of the attention heads has the same dimensions $N \times C$, where $N = H \times W$ is the length of the sequence, and the SDPA has to calculate the similarity matrix between any two positions. This results in a computational complexity of $O(N^2)$, which reduces the efficiency of the attention block

drastically for large images. The efficient self-attention block introduces a reduction ratio R, similar to the sequence reduction process introduced in the PVT, in order to reduce the input sequence length with

$$\begin{aligned} \hat{K} &= \text{Reshape}\left(\frac{N}{R}, C \cdot R\right)(K) \\ K &= \text{Linear}\left(C \cdot R, C\right)(\hat{K}). \end{aligned} \tag{2.17}$$

The reshaping operation divides the key tensor K into smaller patches of size $\frac{N}{R}$, each having $C \cdot R$ elements. After reshaping, a linear transformation is applied to transform the input from $C \cdot R$ to C dimensions, effectively transforming K from prior $N \times C$ to $\frac{N}{R} \times C$. This reduces the number of parameters and the computational complexity of the MHA block from $O(N^2)$ to $O\left(\frac{N^2}{R}\right)$, as shown in figure 2.19a. The figure shows a similar attention block like figure 2.14, with the difference that the according shapes with the reduction ratio are included for visualization (Lin et al., 2023; Z. Wang et al., 2023; Xie et al., 2021). R is set by Xie et al. (2021) to $[64, 16, 4, 1]$ from stage-1 to stage-4 in the SegFormer, representing the increase in spatial detail and decrease in the need of sequence reduction. These parameters are adaptable.



(A) Efficient Self-Attention, used in the MHA. (B) Mix-Feed-Forward Network after the efficient MHA.

FIGURE 2.19: Graphical representation of the Mix Transformer block of the SegFormer with the efficient self-attention (A) and the Mix-FFN (B). Adapted from Z. Wang et al. (2023), added a residual connection in the Mix-FFN as in Lin et al. (2023).

The output of the efficient MHA is then propagated into a Mix-FFN, which is broken down in figure 2.19b. The limitation of a fixed positional encoding is omitted by utilizing zero padding, as explained in section 2.6.1, in combination with 3×3 convolutions to indirectly encode the positional information. The according Mix-FFN, combining an and convolution into a FFN, can be described as

$$x_{out} = \text{MLP}(\text{GELU}(\text{Conv}_{3\times3}(\text{MLP}(x_{in})))) + x_{in}, \tag{2.18}$$

where x_{in} is the feature from the efficient MHA module (Lin et al., 2023; Z. Wang et al., 2023; Xie et al., 2021). Xie et al. (2021) show that these depth-wise convolutions are sufficient for efficiently providing positional information.

After propagating the images through the efficient MHA and Mix-FFN layers, Overlap Patch Merging is applied to merge the overlapping patches into a complete image. In the ViT, merging non-overlapping patches can lead to boundary effects where the transitions between patches are not smooth, resulting in discontinuities in segmentation tasks. In the SegFormer, overlapping patches are merged using a specific convolutional operation to blend the edges of patches. This ensures that adjacent patches with similar or identical category labels are fused together smoothly. This means that the semantic meaning of the image is preserved, and the segmentation map is semantically coherent (Lin et al., 2023; Xie et al., 2021).

The image patches derived from splitting the input image into 4×4 patches at the beginning are propagated through multiple of these MiT blocks, which obtain multi-level features at 1/4, 1/8, 1/16, and 1/32 of the original image resolution, annotated in figure 2.18 on the top. All of these features are directly fed into the All-MLP decoder, which fuses them together and generates the segmentation mask in four main steps (Lin et al., 2023; Xie et al., 2021)..

Initially, the multi-level features F_i from the encoder are passed through an MLP layer to normalize the channel dimensions. In the layer, these features are upsampled to 1/4th of their original size in a second step. In the third step, another MLP layer fuses the concatenated features together to F. Finally, a subsequent MLP layer processes the fused feature map to generate the segmentation mask M, achieving a resolution of $\frac{H}{4} \times \frac{W}{4} \times N_{cls}$, where N_{cls} is the number of categories for the segmentation. Lin et al. (2023), Z. Wang et al. (2023), and Xie et al. (2021) provide the according formulas to describe these operations.

The SegFormer is highly scalable, as it provides tuning the hyperparameters of the MiT encoder, resulting in multiple model variants from MiT-B0 with 3.7 to MiT-B5 with 82 Million parameters, offering the possibility to utilize it for real-time or high performance applications (Xie et al., 2021). For the LULC Utility, the SegFormer is a strong candidate, as it is designed to be efficient and flexible while achieving high performance. Its capability to capture contextual information over long-ranges effectively and efficiently is crucial and beneficial for the integration of spatial features into the feature space of the LULC Utility.

2.9 Summary

DL is a diverse field with many different models and techniques, each with its own strengths and weaknesses. This chapter provided an introduction into DL with an overview of some of the most important models and concepts for CV. From the simplest NNs, Perceptrons, to advanced architectures like CNNs and attention-based architectures like the ViT, all played a role in the development of the SegFormer, which is the basis for the LULC Utility experiment framework used in this study. The highly complex training process of these architectures is the reason why DL models are so computationally

demanding, especially as datasets become larger and computational power increases. That is why optimizing the computational efficiency of such models is increasingly critical. The SegFormer is designed to be more efficient and flexible than comparable architectures, making it a strong candidate for the LULC Utility experiment framework. In that regard, the next chapter will introduce the methodology used to evaluate the impact of spatial contextual information on the performance and resource consumption of the SegFormer in the LULC Utility, proofing a concept utilizable for other DL tasks in the CV domain as well.

Chapter 3

Methodology

The research questions contain three main points, outlining the methodology of the study: exploring the relationship between select road network attributes and adjacent LULC classes, integrating them into the feature space of the LULC Utility with a suitable encoding method, and in the end evaluating its impact regarding the model's performance. The assessment of the relationship between roads and adjacent LULC classes includes preprocessing the road network and following a methodology which spatially joins the road network to the LULC classes. Based on this spatial join, the relationship can be analyzed statistically. The findings are used for the development of multiple methods for feature engineering and encoding the road network, facilitating its integration into the feature space of the LULC Utility. The last step is to evaluate the impact of the road network as spatial contextual information on the model's performance and resource consumption, which is done by comparing the baseline model to different so called extensions with differently encoded road network attributes integrated into the feature space of the model.

3.1 Study Area

The selected study area is located in Baden-Württemberg in the southwest of Germany and spans an area of approximately 1,314 km². The area is composed of the home town of the university this thesis is written at, the city of Heidelberg, along with two adjacent areas, the city of Mannheim and the Rhein-Neckar-Kreis, all part of the Rhine-Neckar Metropolitan Region.

Heidelberg is a city with approximately 160,000 inhabitants on an area of about 109 km², known for its picturesque old town, the Heidelberg Castle, and the oldest university in Germany, the Ruprecht-Karls-Universität Heidelberg (City of Heidelberg, 2024b). Its landscape is characterized by a mix of urban areas and farmlands, with the Neckar river running through the city and the Odenwald, a low mountain range with large areas covered by forests, making a large part of the city's area in the east (City of Heidelberg, 2024a).

Mannheim is the second largest city of Baden-Württemberg, spanning roughly 145 km² with nearly 312,000 inhabitants further northwest of Heidelberg, known for its grid-like city layout and the Palace of Mannheim. Due to its size and location at the confluence of the rivers Rhine and Neckar, it is an important economic hub in the region. Its landscape is more urban than Heidelberg's (City of Mannheim, 2024). The Rhein-Neckar-Kreis almost surrounds the city of Heidelberg completely, the city of Mannheim only borders Heidelberg to the northwest.

Spanning an expansive area of about 1,061 km² and home to over 556,000 people, the Rhein-Neckar-Kreis is by far the largest of the three areas. This rural district is renowned for its vineyards, the Neckar river, and the Odenwald. It primarily consists of farmlands, forests, and many spatially distributed small towns and cities across the area (Rhein-Neckar-Kreis, 2024; Rhine-Neckar Metropolitan Region, 2024). Figure 3.1 shows the selected area of interest with LULC classes provided by OSMLanduse (Schultz et al., 2017).

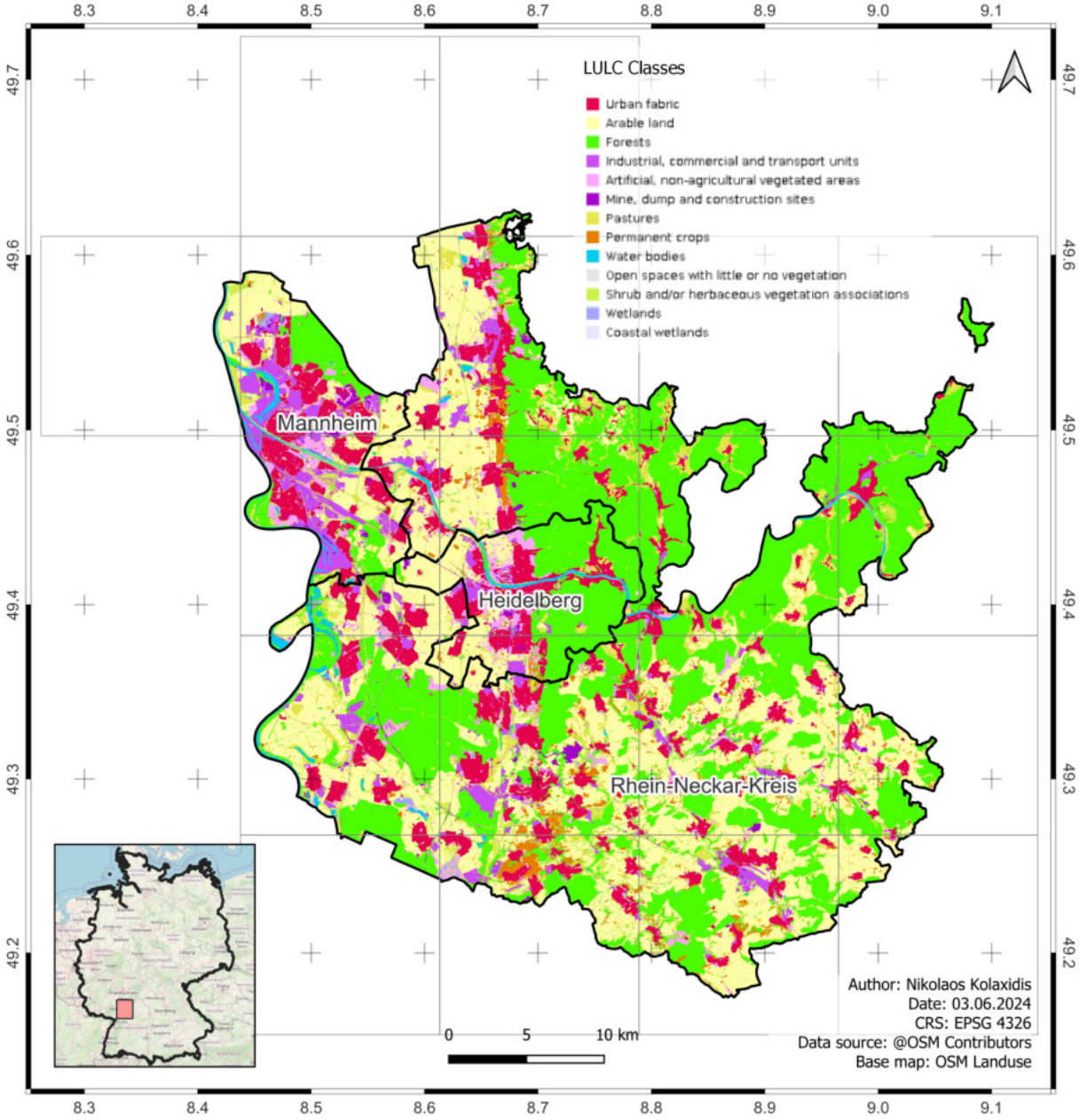


FIGURE 3.1: Selected AOI, composed of the cities of Mannheim and Heidelberg as well as the adjacent Rhein-Neckar-Kreis in southwest Germany. The colors represent the land use and land cover classes, provided by OSMLanduse, and the boxes represent the patches calculated by the area descriptor module of the LULC Utility.

The AOI, called Mannheim - Heidelberg - Rhein-Neckar-Kreis (MA-HD-RNK) from this point on, is located in the northwest of Baden-Württemberg, blending the parts of the Odenwald low mountain

range and the Upper Rhine Rift Valley. Incorporating two larger and many smaller cities as well as large rural regions, it is characterized by a heterogeneous LULC composition. As seen in figure 3.1, this includes urban and rural areas, water bodies, forests, different artificial land uses, as well as large farmland patches. Moreover, it features a diverse relief, a result of the geographical influences of the rift valley and the Odenwald, all within a comparatively small area.

The area descriptor module of the LULC Utility, explained further below, splits the AOI in 19 rectangular patches. These can also be seen in figure 3.1.

3.2 Data

This section describes the various datasets utilized for the explorative relationship analysis and model training, including OSM, road networks and LULC classes from OSM, as well as satellite imagery. All datasets, except those automatically created during model training, are queried for the timestamp *2024-03-15T 00:00:00Z* to ensure consistency and comparability.

3.2.1 OpenStreetMap

The LULC Utility utilizes OSM data as ground truth (true) labels for the LULC-M. That is because studies have shown its suitability for remote sensing based LULC-M (Usmani et al., 2023; Yang et al., 2017). Schott et al. (2024), furthermore, evaluated the suitability of OSM for deriving LULC labels for remote sensing-based models. They based their analysis on intrinsic quality indicators of the data and concluded that using OSM proves high potential for LULC-M.

OSM is the world's largest community-based Volunteered Geographic Information mapping project. It provides free geographical vector data of different scale, including buildings, roads, borders, as well as other geographical features. The database is publicly accessible and maintained by a global community of volunteers. The community comprises both individual contributors as well as organized mapping communities, including corporate and humanitarian organizations, which contribute to OSM collectively through organized mapathons (Anderson et al., 2019; Herfort et al., 2021). These contributors gather data from a variety of sources, including manual digitization of ortho-rectified satellite imagery, GPS data collection, data imports from other databases, and local knowledge for data verification (Usmani et al., 2023).

Data Structure

A unique aspect of OSM is its use of a hierarchical data structure to organize object-related data. This data can be utilized to efficiently extract or filter information relevant to specific geographical applications or problems (Schott et al., 2024). OSM utilizes three main data elements: nodes, ways, and relations.

A node, defined by latitude and longitude, represents a specific point on Earth. Each node has an ID and coordinates. Nodes can depict standalone features like a tree or a a park bench, or define a way's shape. When part of ways, nodes usually lack tags, but exceptions exist, like traffic signals on roads or

pylons on power lines. Nodes can also be members of relations, with their roles indicated within the relation (Minghini & Frassinelli, 2019; OSM Wiki, 2024b).

A way is the second fundamental map element, representing linear features like roads or rivers. It is an ordered list of at least two nodes and can be open or closed. Closed ways, where the first and last node are the same, can depict area boundaries like buildings or forests. However, they can also represent loops, such as roundabouts. Tags on the way help distinguish between these uses. If a way consists of more than 2,000 nodes, a more complex MultiLineString or MultiPolygon relation data structure will be required instead (Minghini & Frassinelli, 2019; OSM Wiki, 2024b).

A relation in OSM defines relationships between nodes, ways, or other relations. Examples include route relations for highways or bus routes, turn restrictions, and MultiPolygons. The relation's purpose is determined by its tags, particularly *type*. Relations consist of an ordered list of members, which can have optional roles. For instance, in a turn restriction, members have "from" and "to" roles indicating the prohibited turn (Minghini & Frassinelli, 2019; OSM Wiki, 2024b).

All three data elements can have tags. A tag, consisting of a key and a value, describes the element's meaning. For instance, a tag of *highway=residential* indicates a residential road. Each element can only have unique keys. There's no fixed tag dictionary, but conventions exist (Ludwig, Hecht, et al., 2021; Mocnik et al., 2017).

External Applications

Tag usage can be analyzed with the OSM Taginfo application (OSM Foundation, 2024). It provides global statistics on tag usage, including the number of elements with a specific tag, the number of unique values, and the number of users contributing to the tag. The application also provides a tag history, showing the tag's usage over time (Minghini & Frassinelli, 2019; OSM Foundation, 2024).

OSM data can be queried using different tools. One of them, the Ohsome API (Raifer et al., 2019), also developed by the HeiGIT, is a powerful solution for analyzing and extracting both historical and current OSM data. It is particularly useful for understanding spatial-temporal patterns, assessing data quality, and conducting research on volunteered geographic information. The Ohsome API provides a wide range of functionalities, including querying data by time, space, and tags, as well as analyzing data quality and completeness. It also allows users to export data in various formats (Auer et al., 2018).

Limitations & Considerations

Although OSM data is a valuable source for geographical data and specifically LULC-M, it has limitations that always have to be considered (Minghini & Frassinelli, 2019). First, because it is a volunteered geographic information project, data quality can vary depending on the contributors and their experience. There are many quality indicators for OSM, including extrinsic (comparisons to authoritative data) and intrinsic quality evaluations like mapping density and currentness. This is a prominent analysis topic regarding OSM data (Jokar Arsanjani et al., 2015; Schott et al., 2024).

Second, as Herfort et al. (2021) show, the data can also be biased, with more data available in urban areas in the global north or areas with very low income that are prone to disasters, often targeted by

humanitarian mapping projects like the Humanitarian OSM Team (HOT OSM, 2024), but with a lot of missing data especially in the global south. In addition to that, rural areas tend to be more incomplete than urban areas, because OSM also relies on local knowledge, which has to be contributed by the local population. This can lead to a lack of data in rural areas, because of the lower population and possible contributor density (Moradi et al., 2021). Nonetheless, the evolution of OSM, prophesied by Barrington-Leigh and Millard-Ball (2017) and Neis, Zielstra, and Zipf (2012) and later confirmed by Forget et al. (2018) and Herfort et al. (2021), shows the increasing interest of volunteers as well as organizations to contribute to OSM, observable through projects like Missing Maps (Missing Maps, 2024), who aim to fill missing data especially in developing regions. These increase OSM's fit-for-purpose, including LULC-M, as shown by Schott et al. (2024), and it is expected that its fit-for-purpose will even increase in the coming years.

Third, the data can also be inconsistent, with different contributors using different tags or mapping conventions (Ludwig, Hecht, et al., 2021; Majic et al., 2017). Although the OSM community tries to establish conventions and guidelines, the data is still open to interpretation by the contributors (Mocnik et al., 2017). The road network of OSM proves additional challenges in that regard. There are no conventions on how to map and tag a road correctly. It is possible to use the road name and map it as a whole feature, oftentimes resulting in long roads. Alternatively, roads can be split at every intersection, following the concept of a "edge", a street between two nodes (intersections), so features only consist of one delimited road segment and many of those segments have the same name tag (Barrington-Leigh & Millard-Ball, 2017; Marshall et al., 2018). In glsosm, both types of roads exist, but using both methods without specific rules makes it difficult to analyze road networks based on feature counts. That is why it is advised to work with road lengths rather than feature counts if applicable to given task (Hacar et al., 2018; Vargas-Munoz et al., 2021).

So all in all, OSM can be incomplete, outdated, mismapped, or inaccurate. However, its evolution over the last decade shows a significant increase in data quality as well as mapping completeness. Due to its open nature in adding and accessing data, contributors are very interested in enhancing and filling in missing data. A plethora of studies have shown the suitability of OSM data for research tasks, especially LULC-M, but always include quality assessments to prove its suitability for specific areas. Therefore, despite its limitations, OSM is a very valuable source for geographical data in the context of this study.

3.2.2 Road Network

The road network for the study area is queried using the aforementioned Ohsome API (Raifer et al., 2019) from OSM. Due to the heterogeneous LULC composition and overall rural character of the AOI, it is expected that a large part of the road network consists of rural roads, namely tracks and paths. Additionally, because of the two larger and many smaller cities in the AOI, urban areas with dense road networks of residential types are also expected.

Figure 3.2 shows the complete road network of the AOI MA-HD-RNK, queried by using the filter *highway=**, *way* as OSM type, and *line* as geometry type. This is important, because otherwise all nodes and relations, the elements of OSM as explained above, would be queried also. Additionally, polygons and points should be ignored as well, because they are not part of the road network and

probably hint towards wrongly mapped roads or features not usable for this analysis. Interestingly, even when using this filter, there still are 4 stray features of point geometry in the response. Their existence cannot be explained, for the filter explicitly excludes them. Apart from this, the road network consists of 141,486 LineStrings and 96 MultiLineStrings, which shows that the filter was successful in excluding other geometries and OSM elements. The total length of the road network, obtained by adding up the lengths of all road geometries, is 17,836.11 km.

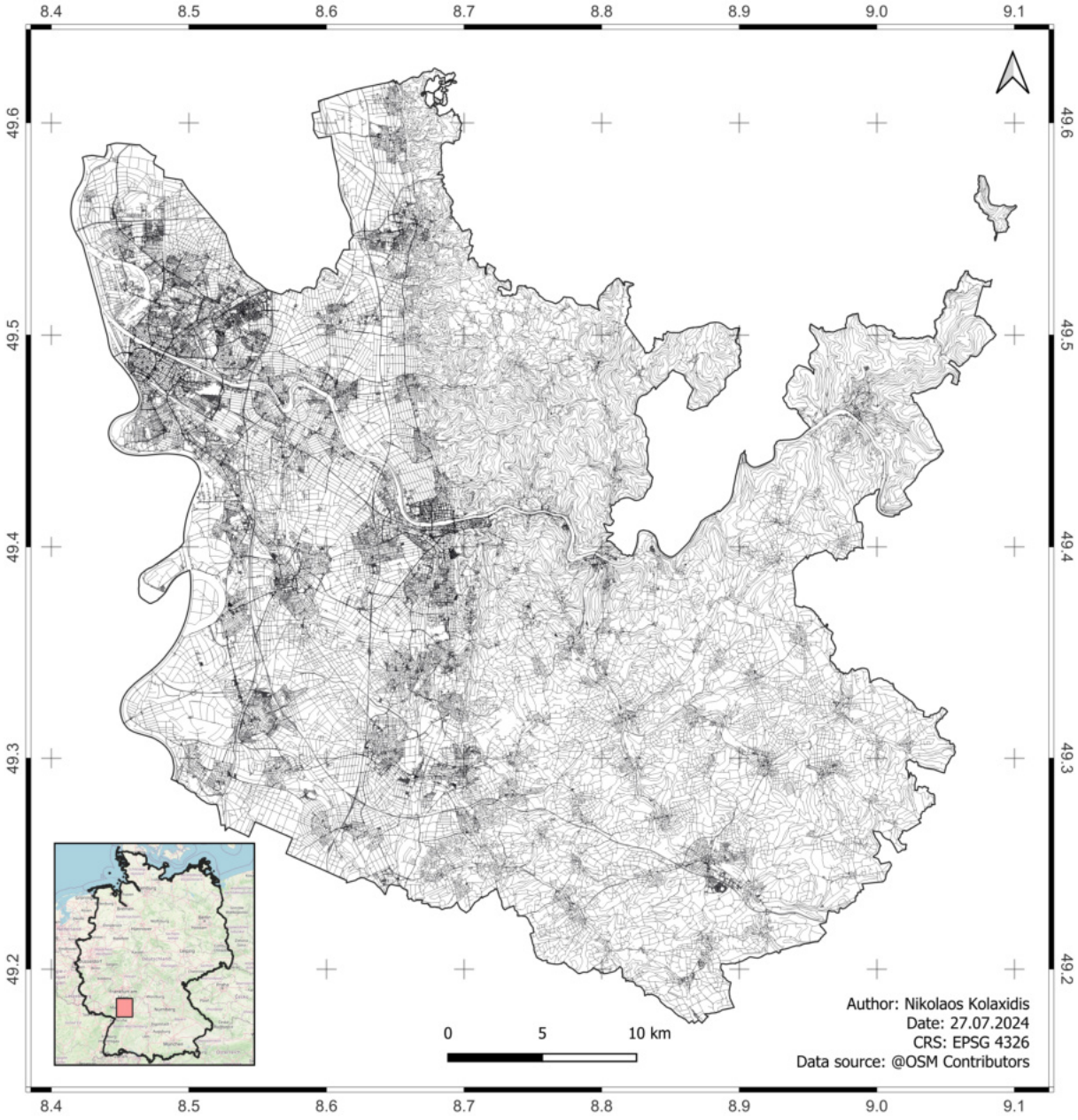


FIGURE 3.2: Complete road network of MA-HD-RNK. Urban areas can be recognized by dense road networks, while rural areas are characterized by fewer roads. The total length of the road network is 17,836.11 km.

Using the Ohsome Quality API (HeiGIT, 2024c), the online dashboard of the Ohsome API, the mapping saturation and completeness of roads in the AOI can be assessed. The mapping saturation of the road network of the last three years, which is calculated by comparing the mapped OSM data to a

modelled saturation curve, is at 97.25 %, meaning it has a high mapping saturation and completeness. Moreover, comparing the mapped OSM data to the Microsoft Roads dataset (Microsoft, 2024), a freely available dataset of roads detected using a DL model workflow, it is clear that in OSM much more roads are mapped (17,836.11 km in OSM vs. 8.760,20 km in Microsoft Roads). But, of all the roads Microsoft Roads has mapped, OSM covers 8.523,75 km of them, achieving a completeness score of 97.3 %. This means that the road network from OSM of the AOI does not cover every road, but is mapped much more than Microsoft Roads, although not covering all of them. Therefore, the road network from OSM can be used for a reliable analysis and model training.

Regarding the first research question, which attributes of the road network are useful for adding spatial contextual information to LULC classes, two main attributes of the introduced road network are in focus. Based on Ahmadzai (2020) and Forget et al. (2018), the road types in the OSM tag *highway*, and based on Levinson and Yerra (2005), Levinson et al. (2007), and Y. Zhao et al. (2023), the speed limits in the tag *maxspeed* are of interest due to their correlation to adjacent LULC classes. Other tags used by Y. Zhao et al. (2023), *bridge, oneway, layer*, and *tunnel*, do not play a role for adjacent LULC as Levinson and Yerra (2005) and Levinson et al. (2007) propose. They actually include traffic flow and congestion as additional parameters for modeling the evolutionary relationship between road network and LULC classes, which apparently are not part of the OSM data. Therefore, the road types and speed limits are the most important attributes for the road network in the context of this study.

Road Types

By looking at figure 3.2, urban areas can be recognized, as expected, by spatially distributed dense road networks, while the majority of the network is characterized by fewer roads, signifying the extensive rural areas. Table 3.1 shows the distribution of road types in percent of the total road length for the AOI MA-HD-RNK. Here, too, as expected, the two most common road types are *track*, which is typical for rural areas, as well as *residential*, which is typical for urban areas (Atwal et al., 2022; Forget et al., 2018). Furthermore, there are a lot of minor road types with a very low percentage (under 0.01 %) of the total road length, including *corridor, platform, bus_stop*, and *elevator*. Most of these road types are either remains of earlier mapping actions (like *razed*) or data not useful for analyzing the road network. This includes the type *steps*, which constitutes 0.32 % of the total road length, although it is not a "way" in the original sense. There also is a special type usually not found in most of the road networks, *raceway*, which has a comparably high percentage, likely attributed to the world-famous Hockenheimring, a racetrack situated near the city of Hockenheim in the Rhein-Neckar-Kreis.

The road network is quite complex, with a total of 32 different road types, observable in table 3.1. There are no roads without a mapped road type, which would be recognized as the value *road* or missing values. The most common road types are *track*, *residential*, *service*, and *path*, which together make up over 75 % of the total road length. The least common road types are *via_ferrata*, *elevator*, *street_lamp*, *bus_stop*, and *razed*, which together make up less than 0.01 % of the total road length.

Due to its complexity, the road network is preprocessed to reduce the computational cost for the subsequent methodology and model training. This includes filtering the road network and the OSM tags, effectively reducing their Dataframe size, and removing the mentioned stray points. Moreover,

in order to reduce the number of road types, only the most common road types (over 0.2 % globally) are included in the analysis, based on the OSM Taginfo homepage (OSM Taginfo, 2024). Global values are used to ensure that the road types are not only common in the AOI, but also in other areas, which makes the procedure more generalizable. This concludes to removing half of the prior 32 road types, resulting in a road network of 16 road types. After preprocessing, the total road network length is reduced to 17,606.58 km, a reduction of only approximately 1.29 % compared to the previous total road network length, although the channels relevant for encoding are reduced by half.

Table 3.1: Absolute and relative road lengths of road types in MA-HD-RNK, sorted by percentage. The percentage is the share of the total road length of 17,606.58 km.

Road Type	Length [km]	Percentage	Road Type	Length [km]	Percentage
track	7374.11	41.34	primary_link	31.67	0.18
residential	2539.51	14.24	pedestrian	31.85	0.18
service	2016.65	11.31	trunk_link	30.83	0.17
path	1856.03	10.41	secondary_link	19.25	0.11
footway	1318.57	7.39	construction	14.69	0.08
secondary	534.40	3.00	raceway	13.19	0.07
tertiary	530.99	2.98	proposed	8.00	0.05
unclassified	376.17	2.11	tertiary_link	3.37	0.02
primary	282.08	1.58	corridor	2.09	0.01
motorway	247.48	1.39	busway	1.23	0.01
living_street	230.64	1.29	platform	1.15	0.01
motorway_link	79.27	0.44	razed	0.25	0.00
bridleway	77.62	0.44	bus_stop	0.29	0.00
trunk	72.88	0.41	street_lamp	0.03	0.00
cycleway	85.13	0.48	elevator	0.04	0.00
steps	56.64	0.32	via_ferrata	0.02	0.00

Speed Limits

Another aspect of the road network are speed limits. In OSM, the most prominent tag for speed limits is *maxspeed*. This tag usually has an integer value mapped, which shows the maximum speed limit on this road segment. However, this tag can not only contain an integer value as speed limit, but also values like *variable, walk*, and even *DE:rural*. These are implicit speed limits, based on default speed limits defined by the OSM community or country specific regulations (OSM Wiki, 2024a). So, for example, a value of *DE:rural* means that the speed limit is 100 km/h, because this is the default speed limit for rural roads in Germany, not including the vehicle type, according to OSM Wiki (2024a). Another special case is the value *none*. This term is used not as an indicator of missing data, but to specify roads, mostly certain sections of German autobahns and a few other areas, where no speed limit is enforced. It is important to distinguish this from situations where the speed limit is unknown (OSM Wiki, 2024c). So, the speed limit is not always directly mapped, but can be converted to one, based on the country specific regulations or the OSM community's default speed limits.

Table 3.2 shows the values in the *maxspeed* tag for the AOI, excluding the ambiguous value *variable*. The largest difference in comparison to the road types is that there are missing values, represented by

None (mind the capital first letter) or Not a Number (NaN), as it is used from this point on. These values make up 76.34 % of the total feature count, which is a significant share. The most common mapped speed limits area 30, 50, and 70 km/h, which are typical for urban areas. Together they make up over 20 % of the total feature count. This coincides with the distribution of the road types, for urban roads like *residential* and *living_street* have a similar share in the road network.

Table 3.2: Value Counts in the Tag *Maxspeed* for MA-HD-RNK, sorted by percentage. The percentage is the share of the total feature count. Mind the different *None* (missing value) and *none* (no enforced speed limit). Value *variable* with 3 entries is excluded due to aesthetic reasons.

***Maxspeed* Value**	**Count**	**Percentage**	***Maxspeed* Value**	**Count**	**Percentage**
None/NaN	102871	76.34	15	80	0.06
30	16065	11.92	80	59	0.04
50	9341	6.93	5	24	0.02
70	2234	1.66	130	19	0.01
100	1209	0.90	40	16	0.01
10	641	0.48	90	13	0.01
20	544	0.40	6	12	0.01
120	533	0.40	32	9	0.01
none	437	0.32	8	5	0.00
walk	316	0.23	25	5	0.00
7	195	0.14	DE:urban	1	0.00
60	121	0.09	DE:walk	1	0.00

3.2.3 LULC Classes

OSM provides multiple tags to classify LULC areas. As Yang et al. (2017) show, the OSM tags *landuse* and *natural* are the most common tags for LULC-M. The *landuse* tag is used to describe the human use of the land (land use), while the *natural* tag is used to describe the natural state of the land (land cover). The *waterway* tag is additionally used to describe water bodies, like rivers, lakes, and reservoirs. Schott et al. (2024) suggest an approach of aggregating minor *landuse, natural*, and *waterway* tags into major classes. By using these, the landscape segmentation becomes less fragmented compared to using all the minor tags, thereby improving the interpretability of the results. Table 3.3 shows the LULC classes used in this study according to one of the label files ("label_v3") shipped with the LULC Utility. These aggregated major LULC classes are based on the works of Schott et al. (2024), but extend them with two additional classes, and include the three OSM minor tags *landuse, natural*, and *waterway*.

The major LULC classes in this study consist of six distinct classes. "Built-up" areas are mostly areas with anthropogenic impermeable surfaces, like residential areas, large building complexes, and parking lots, and are usually found in urban areas. The filter only uses the *landuse* tag, because these areas are technically not natural land cover. In contrast to the classes proposed by Schott et al. (2024), railways are also included here. "Forest" areas are characterized by dense tree coverage, both managed and not managed. Since forests can serve human purposes and also constitute land cover, both the *landuse* and *natural* tags are utilized. "Water" areas encapsulate bodies of water such as reservoirs, rivers, and

docks. This class is the only one to utilize all three minor tags. These classes are derived from Schott et al. (2024).

Table 3.3: Filters from OSM were used to aggregate minor LULC classes of MA-HD-RNK into major ones (adapted from Schott et al. (2024)), sorted by percentage. The percentage is the share of the total area covered by the LULC classes.

LULC Class	Percentage	OSM Tags
Forest	39.55	*landuse=forest* or *natural=wood*
Farmland	30.62	*landuse=farmland*
Built-up	18.33	*landuse* in (*civic_admin, commercial, depot, education, farmyard, garages, industrial, railway, residential, retail*)
Grass	7.71	*landuse=meadow* or *landuse=grass* or *natural=grassland*
Permanent crops	2.08	*landuse=vineyard* or *landuse=orchard*
Water	1.71	*landuse=reservoir* or *natural=water* or *waterway=dock* or *waterway=riverbank*

The other three classes, "farmland", "permanent crops", and "grass", differ from the proposed classes in order to test the model's usability for smaller distinctions of agricultural areas. "Farmland" refers to expanses of agricultural land regularly tilled, only using the tag *landuse=farmland*. "Permanent crops" denote areas dedicated to long-term crops like vineyards and orchards, solely consisting of the *landuse* tag. Lastly, "grass" areas, often but not exclusively rural, include meadows, grasslands, and urban green spaces, with no distinction made between them, for their cover usually is similar.

Using these filters, the LULC classes are queried for the AOI also by using the Ohsome API. The resulting classes cover 1,208 km² of the total area, which is nearly 92 %. The remaining 8 % are not covered by the LULC classes, which is due to the OSM data not being complete or the filters excluding some areas. Figure A.1 in appendix section A.3 shows the aggregated classes in contrast to prior figure 3.1, using the color codes defined also in the used label file.

The shares of the LULC classes are additionally shown in table 3.3. As expected for the mostly rural AOI, the most common LULC classes are "forest" and "farmland", which together make up over 70 % of the total area. Another large share of about 18 % is taken by the "built-up" class, which is also as expected due to the two larger and many other cities in the AOI. The least common class is "water", solely because of the rivers Rhine and Neckar and the lack of other large water bodies like lakes.

In the model, these LULC classes are used as ground truth labels for the model to learn from. The labels are rasterized images where each pixel is assigned to a specific LULC class. For every patch of the AOI, there is one ground truth label per LULC class. These are stacked on top of each other by area size, starting with the largest, and finally concatenated into one label image with all six LULC classes per patch.

3.2.4 Remote Sensing Data

The LULC Utility framework is designed for universal application, necessitating the use of globally available multi-spectral satellite imagery with, ideally, high spatial resolution for model training. A famous provider meeting the requirements is the Sentinel mission, operated by the European Space

Agency (ESA) (ESA, 2024). Their imagery is widely utilized in numerous studies regarding LULC-M or using LULC for other analyses due to its free accessibility and relatively high spatial resolution, including Ludwig, Hecht, et al. (2021), Thanh Noi and Kappas (2018), and Zong et al. (2020).

Based on the findings of Tzepkenlis et al. (2023), the LULC Utility utilizes nine input bands, consisting of the polarized VV and VH bands of Sentinel-1, the bands B2, B3, B4, B8, B11, and B12 of Sentinel-2, and a Digital Elevation Model (DEM). Tzepkenlis et al. (2023) show that a combination of actually 13 channels, including multiple indices like the Normalized Difference Vegetation Index, delivers the best results and even performs better than a 17 channel combination, but as the model does not need indices to understand the relationship between the according two bands, these additional index channels are disregarded. Furthermore, Tzepkenlis et al. (2023) split the DEM in the two channels "elevation" and "slope", but the LULC Utility uses the DEM as a single channel. The reduction of the feature space is based on the assumption that the model can learn the relationships between the bands without explicitly providing them. This is also supported by the findings of the same authors, who show in their ablation study that the model, especially the SegFormer, can still perform well even when removing bands from the input data.

But because the LULC Utility is a state-of-the-art tool for LULC-M with its nine input bands of satellite imagery, some bands have to be removed to increase the possible impact of the road network and enhance its evaluation. The input data for the baseline of the LULC Utility therefore is reduced to five bands, including the Sentinel-2 bands B2, B3, B4, B8, and the DEM. The polarized VV and VH bands of Sentinel-1 as well as Sentinel-2's bands B11 and B12 are removed from the input data, because Tzepkenlis et al. (2023) show that the DEM has a higher importance for the model's performance.

The Sentinel-2 mission includes two polar-orbiting satellites, Sentinel-2A and Sentinel-2B, launched in June 2015 and March 2017, respectively. These satellites capture multi-spectral images across 13 spectral bands, ranging from the visible to the near-infrared and shortwave-infrared parts of the spectrum. With a spatial resolution between 10 and 60 m and a revisit time of 2-3 days, they offer global coverage high-resolution images (Basheer et al., 2022; ESA, 2024; Ludwig, Hecht, et al., 2021). ESA offers two products of Sentinel-2 imagery, namely Level-1C, top-of-atmosphere reflectances, and Level-2A, atmospherically corrected surface reflectances (ESA, 2024). In the context of the LULC Utility, because the focus is on surface features, Level-2A is used. The aforementioned four bands of this product, as shown in table 3.4, are used as input data for the LULC Utility.

Table 3.4: Input data of the LULC Utility, including four bands of the Sentinel-2 mission and a DEM. All have a spatial resolution of 10 m. Adapted from ESA (2024).

Channel	**Central Wavelength [nm]**	**Band Width [nm]**
Sentinel-2 B2 (Blue)	490	65
Sentinel-2 B3 (Green)	560	35
Sentinel-2 B4 (Red)	665	30
Sentinel-2 B8 (NIR)	842	115
DEM	-	-

In addition to the Sentinel-2 bands, a DEM, a representation of the terrain without surface elements like vegetation or buildings, is also integrated in the input data to provide additional information about

the relief. Using Sentinel Hub's Process API (Sentinel Hub, 2024), the DEM is queried for the AOI using two options, depending on availability.

The default one is the Copernicus DEM (Airbus, 2022), which actually is a Digital Surface Model, including all natural and humanmade elevations on top of the terrain as well. Also operated by the ESA, there are Digital Surface Model products in three resolutions, namely 10 m (EEA-10), 30 m (GLO-30), and 90 m (GLO-90) (Airbus, 2022). Because of the availability and global coverage in contrast to the high resolution EEA-10, GLO-30 is the default product. The Sentinel Hub Process API enables upsampling the DEMs to the required resolution of 10 m to align with the Sentinel-2 imagery.

The other option is the Mapzen Terrain Tiles DEM (Mapzen DEM) (Mapzen, 2024). It is mainly based on the Shuttle Radar Topography Mission from 2000, which offers a near-global DEM, but also includes multiple other data sources to fill missing data or locally increase the spatial resolution. The data is provided as tiles, square segments of the Earth's surface, in different spatial resolutions, ranging from 3 m to 156 km, depending on the zoom level and source data. The tiles collection is static and does not depend on the date used for querying it (Mapzen, 2024).

The Sentinel Hub Process API chooses the best available DEM for the given area and required resolution, which is then stacked on top of the Sentinel-2 bands to create the input data for the LULC Utility. Because of the focus on LULC-M, the type of DEM actually does not have a high feature importance, mainly elevation and slope data are important for the model, as stated by Tzepkenlis et al. (2023).

The input data spans six distinct timeframes, which include the three months May, July, and September, for the years 2022 and 2023, respectively. Each interval lasts for one month, starting from the first day of the month and ending on the last day of the same month. In these periods, the image with the lowest cloud coverage is selected also using the Sentinel Hub Process API. For this image, the four bands and the DEM are downloaded. As mentioned above, the area descriptor module splits the AOI into 19 rectangular patches. Using the selected images for each patch, 114 samples are created for the model with five channels each.

3.3 Explorative Relationship Analysis Between Roads & LULC

The first research question explores the nature of the relationship between roads and the LULC classes adjacent to them. The basic idea of doing this is to see if the model would get feasible signals from integrating the road network into the feature space of the LULC Utility. Signals mean information which is interpretable by the model as "unique differences" between the LULC classes regarding their adjacency to specific roads. If the model can learn these signals and apply them to the input data, it can use them to improve its predictions (Y. LeCun et al., 2015; Vaswani et al., 2017).

The methodology to analyze the relationship is based on Atwal et al. (2022), who use a similar approach, but with the aim of predicting and categorizing buildings into residential or non-residential types by leveraging adjacent roads and other geometrical and topological properties. In their study, indicator attributes are assigned to each building based on its proximity to different types of roads. These indicators denote whether a building is within certain ranges of various types of roads. Buffers of multiple sizes (30, 60, and 90 m) are created around each road and a spatial join is performed between

these buffers and the building polygons from OSM. For each intersection, the corresponding indicator variable is set to 1, depending on the type of the road, as indicated by the *highway* tag. This results in a vector with 4×3 indicators, each showing the adjacency to a specific road type. This approach allows for an efficient analysis of the relationship between buildings and their surrounding road networks.

However, the relationship in this study shall not be assessed between buildings and roads, but between roads and LULC classes. Furthermore, the relationship between roads and LULC classes will not be included in the feature space of the model, so transforming this information into a metric for this methodology is not necessary. That is why the methodology of Atwal et al. (2022) is adjusted and explained below.

It involves several key steps. First, the LULC classes and the road network are preprocessed accordingly. Afterwards, the LULC classes are spatially joined by intersection to the road network, following the methodology by Atwal et al. (2022). By doing this, every road gets a LULC class assigned, if one is adjacent to the road. This is followed by the calculation of total road lengths for each LULC class, which helps to avoid aforementioned issues related to relying on feature counts. Subsequently, a pivot table is created to calculate the sum of the road length per LULC class for each road type. Pivot tables are a data summarization tool frequently used in data analysis and business intelligence for organizing and aggregating data in a meaningful way. They allow users to transform (pivot) data to view it from different perspectives, which is particularly useful for summarizing large datasets, spotting trends, and making comparisons, in order to analyze relationships between classes (Alexander & Kusleika, 2022).

In this case, they also can be used to compute the proportion of each LULC class relative to the road types and vice versa. This is accomplished by transforming the calculated absolute values into relative ones. This transformation is done by dividing the values by the sum for each LULC class, resulting in a distribution where the sum of road length shares per road type equals 100 % for each LULC class. Similarly, the values of each road type are divided by the aggregated road lengths for each LULC class, resulting in a reversed distribution where the sum equals 100 % for each road type. By normalizing the data in this manner, it enables a comparative analysis of the relationship of each LULC class and road type, both dependent and independent of the absolute total road lengths.

Utilizing the aggregated sums derived from the road types, a Pearson's Product Moment Correlation (PPMC) analysis is conducted. This shows how LULC classes differ among each other in their relationship to the road types. This analysis is a statistical method used to determine the strength and direction of the relationship between two continuous variables. It ranges from -1 to 1, where 1 indicates a perfect positive linear relationship, -1 a perfect negative (opposing) linear relationship, and 0 no linear relationship at all. The correlation coefficient is calculated by dividing the covariance of the two variables, in this case LULC classes and road type patterns per LULC class, by the product of their standard deviations (Awuh et al., 2019; Feng & Myint, 2016; Kibena et al., 2014). The results are visualized in a heatmap, where the correlation coefficients are color-coded. Based on similar visualizations by Al-Taei et al. (2023), X. Chen et al. (2016), and Khamchiangta and Dhakal (2020), the two-tailed significance levels of the values are shown as asterisks, where one asterisk indicates a p-value below 0.05, two asterisks a p-value below 0.01, and three asterisks a p-value below 0.001. This allows for an easy interpretation of the linear relationship between LULC classes and the patterns of road types in them.

Due to the fact, that the relationship between roads and LULC classes can vary depending on the distance between them, the influence of different buffer sizes is included into the analysis as well. For example, there are roads adjacent to forests, but not inside forests. By using the method explained above, the LULC class "forest" will not be assigned to these roads, because they are not intersecting them. But, these road types still can give spatial contextual information about these adjacent LULC classes. To account for this, the road network is buffered with different sizes, based on the methodology of Atwal et al. (2022), and then spatially joined to the LULC classes. In this case, every road is assigned to every LULC class, if they are spatially intersecting, resulting in larger datasets than the bufferless intersection. However, the buffer sizes are lowered to 0, 10, and 25 m, because LULC areas are usually closer to roads than buildings. Since multiple buffers are used and they conceptually represent the distance of adjacent LULC classes to roads, changes between the different buffers are also calculated. This way, the influence of different buffer sizes on the relationship between roads and LULC classes can be analyzed.

The whole methodology is repeated for the speed limits as well, resulting in a deeper understanding of the relationship between roads and LULC classes from different perspectives. The code for this analysis is referenced in appendix A.1.

3.4 LULC Utility

In order to integrate the processed road network as spatial contextual information and evaluate its impact on the model's performance, a suitable platform has to be used. The LULC Utility (HeiGIT, 2024b) is a state-of-the-art tool for LULC-M, developed by the Climate Action focus group of the HeiGIT. It is designed to segment LULC areas in satellite imagery by leveraging the capabilities of the SegFormer model architecture, as described in section 2.8.3. The creation of the LULC Utility was motivated by the need for a system that allows for the preparation of semantic segmentation models customized for particular research scenarios, for specific AOIs, and with custom data sources. Moreover, pre-trained segmentation models for multi-spectral remote sensing data are usually aimed toward specific research areas and questions, so offering a completely open tool fills this software gap (Guo et al., 2018).

It incorporates the possibility to adapt the LULC labels, based on OSM (see table 3.3), as well as the bands of the input data and its source (see table 3.4), by facilitating the Ohsome and Sentinel Hub Process APIs, as explained above. Additionally, it offers an easy platform for testing new segmentation methods and can function as a high-quality imputing mechanism of missing data for real-time OSM data. The LULC Utility is designed to be lightweight and adaptable, suitable for different hardware configurations, and to establish a project baseline that can be easily configured to incorporate various data sources and model implementations. It offers a comprehensive solution for training deep learning models tailored to specific research scenarios in the realm of semantic segmentation, managing these models through registration and tracking in an online store, Neptune.ai (Neptune Labs, 2024), as well as deploying them for local use via a Representational State Transfer API (HeiGIT, 2024b). This makes the LULC Utility a versatile and suitable tool for a multitude of tasks, including LULC-M.

3.4.1 Preparation Modules

The LULC Utility consists of three modules, namely the area descriptor module, the normalization module, and the training module, all required to successfully train a model from scratch for LULC-M. They are introduced in the following sections.

Area Descriptor Module

The area descriptor module (*compute_area_descriptor.py*) is an important step to reduce the computational cost of the model training and drastically decrease the runtime. It splits the AOI into smaller tiles (patches) with a specified extent, in order to reduce the dataset size of the input data and the total size of each batch during training. This is necessary because the model cannot process the entire AOI at once due to hardware limitations. The area descriptor module is designed to be flexible and adaptable, allowing the user to specify the size of the patches and the overlap between them. This enables the user to adjust the granularity of the split based on the specific requirements of the model and the available hardware resources. The descriptions of the patches are then saved to a file to be accessible for other functions, whereas each tile is described by a unique identifier, its bounding box coordinates and the according polygon geometry, and the provided start and end dates of the input data. Using the configuration of the vanilla LULC Utility, the AOI is split into 19 patches.

Normalization Module

The normalization module (*calculate_dataset_statistics.py*) executes a regularization process, advised when training DL models (Mehlin et al., 2023; A. Zhang et al., 2023). Particularly when training with images, it is essential to normalize the data across different sensor channels to ensure uniformity in the range of input values (Szeliski, 2022). This process is relatively straightforward for surface reflectance data from Sentinel-2 sensors, but DEMs require a more nuanced approach to normalization. Specifically, elevation data benefits from normalization that is tailored to local conditions, because this makes the data more interpretable. In order to normalize the input data, mean and standard deviation are calculated for a single batch.

Furthermore, LULC datasets often have class imbalances, a situation that can be worsened by OSM data due to the nature of its collection process. To mitigate the impact of this imbalance on model training, employing weights to classes to adjust the loss function is an effective strategy. This adjustment allows the model to place greater emphasis on underrepresented classes, thereby enhancing its ability to learn from a diverse range of data and improving overall performance.

When executing this module, the training dataset is created and saved to cache, including the input data as explained in 3.2.4, as well as the labels as explained in 3.2.3. The training module can then save the time of creating the dataset and, instead, directly load the cached dataset for training.

3.4.2 Training Module & Model Implementation

The training module (*train.py*) is responsible for loading the data, initializing and training the model, communicating with the online storage, and evaluating the model's performance. It is the main part of the LULC Utility and uses the data prepared by the other two modules for training the SegFormer

model. The model is implemented in *PyTorch Lightning* (Lightning AI, 2024), a popular open-source ML library, which provides a wide range of tools and utilities for building and training DL models. The model is trained with the *PyTorch Lightning* Trainer class, which simplifies the training process by providing a high-level API for training models in PyTorch. During the training phase, the model is optimized with the Adam optimizer with an initial learning rate of 0.0005, which computes adaptive learning rates for each parameter. It is a popular choice for training DL models due to its good performance and fast convergence (Kingma & Ba, 2015). The loss function used for training is a categorical CLE, which is commonly used for multi-class classification (segmentation) tasks, explained in detail in section 2.5.2 (Diakogiannis et al., 2020). Both the optimizer and the loss function are used in similar research studies like Diakogiannis et al. (2020) and Tzepkenlis et al. (2023) and are suitable for tasks including semantically segmenting remote sensing data.

The LULC Utility generates a model instance in the ONNX format (ONNX.ai, 2024), a prevalent format that specifies all required operations for a ML model to execute its inference function. This format is used to describe models across different ML frameworks. The LULC Utility does not output a segmentation map or mask, it only trains the model and deploys it for applications. Another thing to mention is the capability of the LULC Utility to predict the LULC classes of unknown areas, where no labels exist. The unknown areas are not used for training by masking the loss function, but in the prediction phase, the utility still tries to predict the according class based on the learned pattern from the training data, resulting in predicting 100 % of the area, even when there are large areas of unknown labels.

Regarding efficiency and resource consumption, the LULC Utility already implements some optimization techniques as proposed by Mehlin et al. (2023). One of them is data augmentation, which artificially increases the input data size through modifications. This method enhances diversity in the input data and bridges the gap between training datasets and real-world scenarios (Mehlin et al., 2023; Z. Wang et al., 2024). The training module applies multiple data augmentation methods on the training dataset. This includes horizontal and vertical flips, which flip the images on either x or y axis, elastic transform, which transforms the morphology of objects in images and produces a see-through-water-like effect, coarse dropout, which removes rectangle regions of the images, and random (training dataset) as well as center crops (validation and test datasets), which crop the images to a specified size (Torchvision, 2024; Z. Wang et al., 2024). All of these methods are applied randomly on the input data with a probability of 25 % and effectively provide the model with more data it can learn from than is initially available.

Other techniques include using *PyTorch Lightning* as a ML framework, hardware suitable for DL, and an already efficient model architecture (SegFormer), which itself implements some of the proposed techniques by Mehlin et al. (2023). Furthermore, the aforementioned normalization process accelerates convergence and, therefore, decreases the runtime and resource consumption of the LULC Utility. Additionally, an early stopping callback mechanism is implemented, which stops the training process if the validation loss does not improve for a specified number of validation epochs. This saves computational resources by stopping the validation process early if the best hyperparameter configuration is achieved (see section 2.5.6) (A. Zhang et al., 2023).

So all in all, the LULC Utility is already well-optimized for efficient training, but the integration of the road network into the feature space could potentially improve the model's performance and resource

consumption even more.

3.4.3 Baseline & Model Parameters

To create a basis for comparison for the research task, a baseline model has to be trained, not altering the architecture of the vanilla LULC Utility and adjusting the configuration to match the specifications of the machine used for training the model. The chosen model variant of the SegFormer family is the "MiT-b0" (Hugging Face, 2024), which is the smallest variant with the fastest inference, as shown by Tzepkenlis et al. (2023) and Xie et al. (2021). This is used along with a reduced feature space as explained in section 3.2.4 to enhance the possible impact of the road network and make its evaluation more expressive. A restriction of the runtime using a maximal epoch limit is not enforced, so the time needed for convergence can also be compared.

The input data of 114 samples is split into 80 % training data and 10 % test data, with the remaining 10 % used for validation. The crop size for the batches is set to 512×512, reducing the default 1024×1024 of the LULC Utility by half due to memory limitations (see 3.6). The batch size is kept at 4 patches per batch, along with an increase of the number of concurrent workers from 8 to 12 to speed up the data preparation process. All other hyperparameters are kept the same as in the vanilla configuration of the LULC Utility, including applied transformations for data augmentation, optimization parameters, and logging configurations for the interaction with the online store. It should be mentioned that the model has not been optimized for the task at hand, which could influence its performance.

The subsequent extensions are trained with the same hyperparameters, configuration, and LULC labels as the baseline model. The only difference is the number of channels in the input tensor, which include different encodings of the road network.

3.5 Integrate Road Network into Feature Space

Training the LULC Utility is done using multiple types of road network integrations, called "extensions", based on the nomenclature of Alhassan et al. (2020). Using the baseline defined by the LULC Utility, the extensions are designed to integrate different attributes of the road network into the feature space of the model, without altering other aspects of the LULC Utility. Due to its architecture, it enables the flexible integration of additional channels into the input tensor, effectively enlarging the feature space. All extensions follow a similar integration process, but differ in the applied feature engineering steps of the specific attributes. Figure 3.3 visualizes the research framework, showing the steps taken for each attribute of the road network.

Because the model is non-deterministic, the baseline and each extension are run three times each to get multiple results per extension, enhancing their comparability. The results are then grouped together, averaged, and compared to each other to evaluate the impact of the different encodings and road networks on the model's performance. Table 3.5 shows the number of channels for each model configuration. In order to to minimize the computational cost by keeping the feature space small, the number of channels for the second, third, and fourth extensions are determined by analyzing the relationship between the road network and the LULC classes and clustering features together. This way,

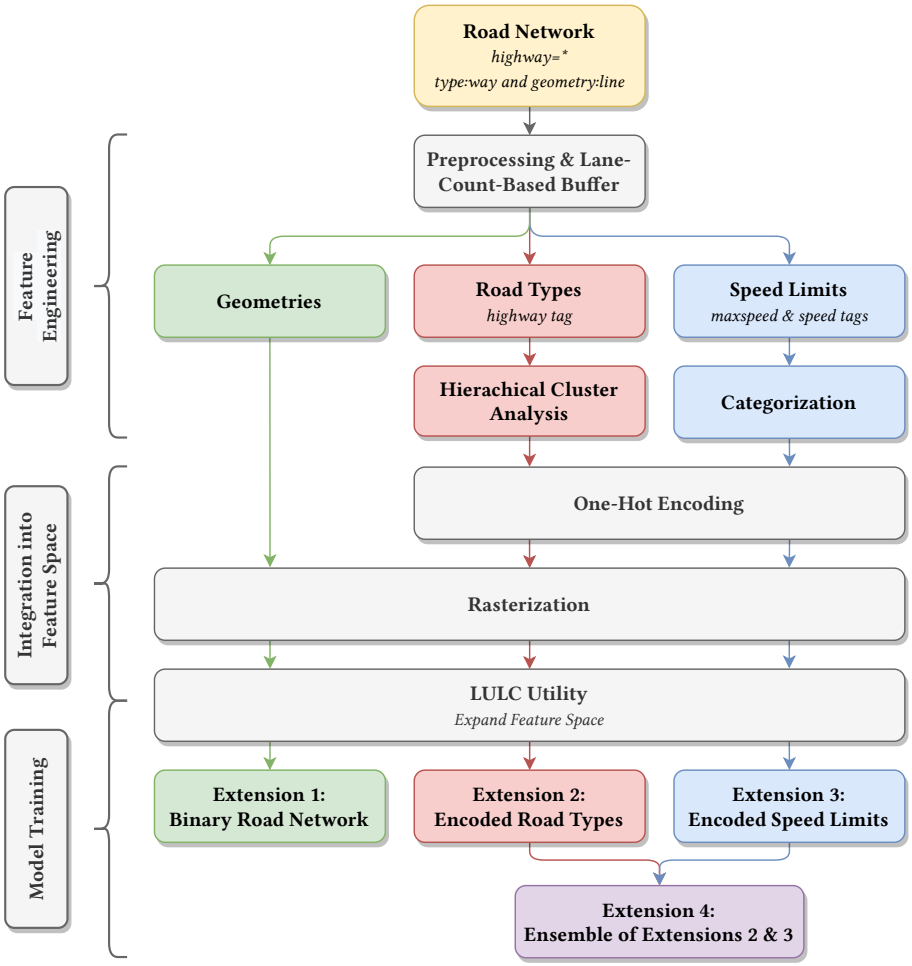


FIGURE 3.3: Research framework for integrating multiple road network attributes into the feature space of the LULC Utility. The colors represent steps taken for specific attributes of the road network, resulting in the different extensions for the LULC Utility.

road types and speed limits can be encoded in a manner that allows the model to distinguish between LULC classes effectively, without burdening it with unnecessary tensor channels.

TABLE 3.5: Tensor channel count for the baseline and the extensions, which are built on top of the baseline. The number of channels is the sum of the Sentinel-2 bands, the DEM, and the additional channels of the (encoded) road network for each extension.

Model Configuration	Number of Channels
Baseline	5
Extension 1 - Binary Road Network	6 (5 + 1)
Extension 2 - Encoded Road Types	16 (5 + 11)
Extension 3 - Encoded Speed Limits	13 (5 + 8)
Extension 4 - Ensemble of Extensions 2 & 3	24 (5 + 11 + 8)

The code for the integration steps of the road network into the LULC Utility is referenced in appendix A.1 and explained in the following sections.

Rasterization

The main difficulty to integrate the road network into the feature space is rasterizing the OSM vector data into a raster format, which has the same height and width dimensions as the channels of the baseline input tensor. This is necessary because the segmentation model works pixel-wise, as explained in section 2. If the pixels do not match between each channel of the tensor, the model cannot assign the values of the stack to the correct pixel (Dosovitskiy et al., 2020; Goodfellow et al., 2016). So for all extensions, the road network has to be rasterized, regardless of the encoded information.

This is done by using the same methodology as for the creation of the LULC labels, but adapted to the road network. After preprocessing the road network and encoding it according to the requirements of each extension, it still exists as vector data. By using the python package *Geocube* (Corteva Agriscience, 2024), this vector data can be rasterized by assigning a true or 1 to each pixel where a road is intersecting in the final raster.

The raster is created by merging a background layer with the same extent as the according patch and the road network and appending a boolean column to the resulting DataFrame. To ensure that the road network is appropriately overlaid on the background, the dataset is sorted based on the area size of each geometry in a larger-to-smaller manner, so that the road network overlays the background.

Rasterization is achieved through the utilization of the *make_geocube* function, which converts the prepared vector data into a Geocube — a georeferenced raster representation of vector data where each pixel's value corresponds to the presence or absence of the road network. This Geocube is then resampled to match the target spatial resolution specified by the satellite imagery shape returned from the Sentinel Hub Process API, employing bilinear interpolation to ensure a smooth transition between different spatial scales. This resampled raster can then be added as a new channel to the feature space.

Lane-Count-Based Buffer

In the baseline, some roads, which are missing in the labels, are predicted granulated. This means that parts of the roads are assigned to different LULC classes, even if adjacent LULC differ. For example, the road is predicted as "water", "built-up", and "grass", although it lies between "farmland" and "built-up". The full road should be either "built-up" due to its surface reflection, appended to either of the two LULC classes because of its proximity, or disregarded if both sides are the same LULC class (merged with the class).

Studies looking into road detection models using satellite imagery showed that the two largest road types *motorways* and *trunks* are quite well observable due to their size (Atwal et al., 2022). Both are strongly regulated in Germany due to their importance for the national transportation network (Road and Transportation Research Association, 2011). The regulations restrict them in their width, lane count, and speed limit. They classify them into three categories: long-distance and inter-regional motorways (EKA 1), motorway-like roads called trunks (EKA 2), and urban motorways (EKA 3) (Road and Transportation Research Association, 2011). The widths often span over multiple tens of meters, with lane counts of up to four lanes in each direction. Because the satellite imagery has a spatial resolution of 10 m, these roads and also intersections are usually visible in the imagery, as seen in figure 3.4.

FIGURE 3.4: A Sentinel-2 RGB composite image with a high zoom factor where single pixels are distinguishable, depicting a highway intersection in the AOI MA-HD-RNK. The added red pixel illustrates the spatial resolution of 10×10 m, i.e. one pixel. Wide roads and such intersections are both visible in the imagery, spanning multiple pixels, compared to narrow roads like in the lower left which only span maximal one or two pixels.

Based on these regulations, a Lane-Count-Based Buffer (LCBB) was developed to buffer the two largest road types and their links roughly to their expected real size. This is done in order to help the model recognize differences between natural linear features and these major roads. Because of the permanence of the regulations, the values in the function have been hard coded. They depend solely on the road type and lane count of the feature, found in the OSM tags *lanes* and *highway*. The algorithm has been implemented as an apply function, iterating and buffering row-wise, and looks as shown in algorithm 1.

There are some particularities in the regulations. First, *motorways* and *trunks*, so EKA 1 and 2 roads, with only one lane and connectors with more than two lanes are not possible in Germany, so they are not explicitly considered in the buffer function. Second, there are different width restrictions for EKA 1 and 2 roads depending on the construction type (safety lanes etc.), but the function only considers the largest possible width for each lane. This has been done to include more edge pixels of the roads in the buffer. Third, EKA 3 roads are not considered explicitly, because the roads types are not specific. They can be both *motorways* or *trunks*, but also *primary*. Because this would make the algorithm much more complex, EKA 3 roads are not considered as well. The LCBB function is applied during the preprocessing step, before extension specific feature engineering is performed.

Algorithm 1 Algorithm to assign buffer values to geometries based on their lane count.

if $lanecount$ == '2' and $roadtype$ in (*motorway_link, trunk_link*) **then**
  $width \leftarrow 9.5$
else if $lanecount$ == '2' and $roadtype$ == *trunk* (EKA 2) **then**
  $width \leftarrow 28/2$
else if $lanecount$ == '2' and $roadtype$ == *motorway* (EKA 1) **then**
  $width \leftarrow 31/2$
else if $lanecount$ == '3' **then** ▷ road type independent (EKA 1/2/3)
  $width \leftarrow 36/2$
else if $lanecount$ == '4' **then** ▷ road type independent (EKA 1/2/3)
  $width \leftarrow 43.5/2$
else
  $width \leftarrow 6$ ▷ width of a single lane connector
end if

3.5.1 Extension 1 - Binary Road Network

The first extension is a binary road network, which indicates the presence of a road, regardless of any additional information OSM has to offer. Although Courtial et al. (2022) state that only adding location and shape information of a feature does not convey all necessary information for a model to learn from, it can still be a valuable addition to the feature space. Boston et al. (2022) and J. Li et al. (2024) state that one challenge of DL models in LULC-M is the detection of edges between LULC patches. An increase in contextual information for segmention goes along with a decrease in detail of features, which renders the edge detection inaccurate. However, because roads often form boundaries in LULC patches, the road network could help to improve the detection of their edges.

Therefore, the Extension 1 only adds a binary road network to the input data. After preprocessing the road network, it only is rasterized using the aforementioned function and added as an additional channel to the input data, which then is transformed to a tensor utilizable by the *PyTorch Lightning* package. The input tensor therefore has 6 channels in the first extension, the 5 from the baseline with the additional rasterized binary road network.

3.5.2 One-Hot Encoding

For the subsequent extensions, the attributes road types and speed limits have to be encoded in order to integrate them into the feature space of the LULC Utility. This also aligns with the second research question, how select attributes of the road network can be encoded. Both tags are nominal categorical without any specific ordinality, so they have to be encoded in a way that an ordinal relationship between them is evaded. Although one could argue that, for example, roads with type *motorway* have a higher ranking than roads with type *secondary*, as soon as the road types *residential* to *service* are compared, the ordinality is not given anymore. For speed limits, the same applies, no speed limit is more important or better than another, they are just nominally different.

According to Hancock and Khoshgoftaar (2020), one of the most used encoding techniques for nominal categorical data is One-Hot Encoding (OHE). It is a simple and effective way to encode categorical data, where each category (value) is represented as a separate column in the dataset. The number of

columns is equal to the number of unique categories in the dataset, and each column is filled with ones and zeros based on the presence (1) or absence (0) of the category in the data (Hancock & Khoshgoftaar, 2020; Potdar et al., 2017). However, this encoding technique is appropriate only for data characterized by a limited number of possible categories, as the method can lead to an excessive increase in the number of columns. To tackle this problem, sparse representations have been developed, which only store the non-zero values of the matrix, drastically reducing the memory consumption (Hancock & Khoshgoftaar, 2020). The problem for the implementation in the model is, nevertheless, that no matter how dense or sparse the dataset is, every category is encoded into one channel of the tensor and every pixel is filled with a value, because the model cannot handle missing values (they are replaced with zeros). Also, the road network is limited in its complexity and it has already been reduced in the preprocessing step. Moreover, in regard to keeping or increasing the efficiency of the model, the aim is to only use a limited number of categories anyways, so the OHE technique is suitable for encoding of the road types and speed limits even without utilizing a sparse matrix.

As stated by Courtial et al. (2022), using OHE results in tensor shapes of $h \times w \times m$, whereas h is the height, w the width, and m the number of categories. This is reflected in the tensor channels shown in table 3.5 for extensions 2-4. However, additional methods can be used to reduce the road network complexity even further, which are presented in the following sections describing the extensions with encoded road attributes.

3.5.3 Extension 2 - Encoded Road Types

Ahmadzai (2020) and Forget et al. (2018) assumed a correlation between road types and specific LULC classes. In order to test whether the road types can enhance the classification of adjacent LULC classes, Extension 2 encodes the road types as separate channels in the input tensor, indicating both the presence of a road and also its type as additional information. This is done by utilizing OHE as well as the rasterization function. Courtial et al. (2022) used a similar approach to encode building types in their study. However, the road network can be quite complex and contain a lot of different road types, which is reflected in the size of the network with a total of 32 road types in the AOI MA-HD-RNK, as seen in table 3.1. By using the threshold of 0.2 % global share of the road types according to the OSM Taginfo homepage (OSM Taginfo, 2024), the number of road types is reduced to 16, reducing the total number of tensor channels to 21. Regarding the efficiency of the model, keeping the feature space small is advised, therefore, additional road network complexity reduction steps can be taken.

Based on the findings of the explorative relationship analysis between roads and adjacent LULC classes (see section 3.3), a Hierarchical Clustering Analysis (HCA) is conducted to find roads with highly similar patterns of their relationship towards the LULC classes. Similar approaches are proposed by Yao et al. (2023) who used HCA to cluster objects based on similar temporal behavior, Petrakis et al. (2021) who clustered sub-basins based on similar structural, biophysical, and hydrologic traits, and Montazeri et al. (2021) who stated that HCA is generally suitable for similarity analyses, although sensitive to outliers.

HCA is a clustering method, aiming to find similarities between features and grouping them together. The key characteristic of HCA is that it groups the features hierarchically, clustering them multiple

times on different levels/hierarchies. There are two clustering strategies in HCA, agglomerative and divisive. Agglomerative is a "bottom-up" approach where each feature starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy. In contrary, divisive is a "top-down" approach where all features start in one cluster, and splits are performed recursively as one moves down the hierarchy (Johnson, 1967; Murtagh & Legendre, 2014). In order to find similarities, the agglomerative strategy is used, which also is more common than the decisive strategy. An advantage of HCA opposite to other clustering methods like k-means is that it does not require the number of clusters to be specified beforehand, which is beneficial for this explorative approach because the number of clusters is not known (Johnson, 1967).

The HCA is conducted using Ward's method (Ward, 1963), which is a variance-minimizing distance calculation method, aiming to create more compact, spherical clusters. This method is particularly useful for creating clusters that are relatively similar in size (Murtagh & Legendre, 2014). The results are visualized in a dendrogram (see figure 3.5), which shows the averaged hierarchical clustering of the road types based on their pattern similarity over the LULC classes using the three aforementioned buffer sizes (0, 10, 25 m) for the road network. The dendrograms for the three buffer sizes can be seen in appendix A.10.

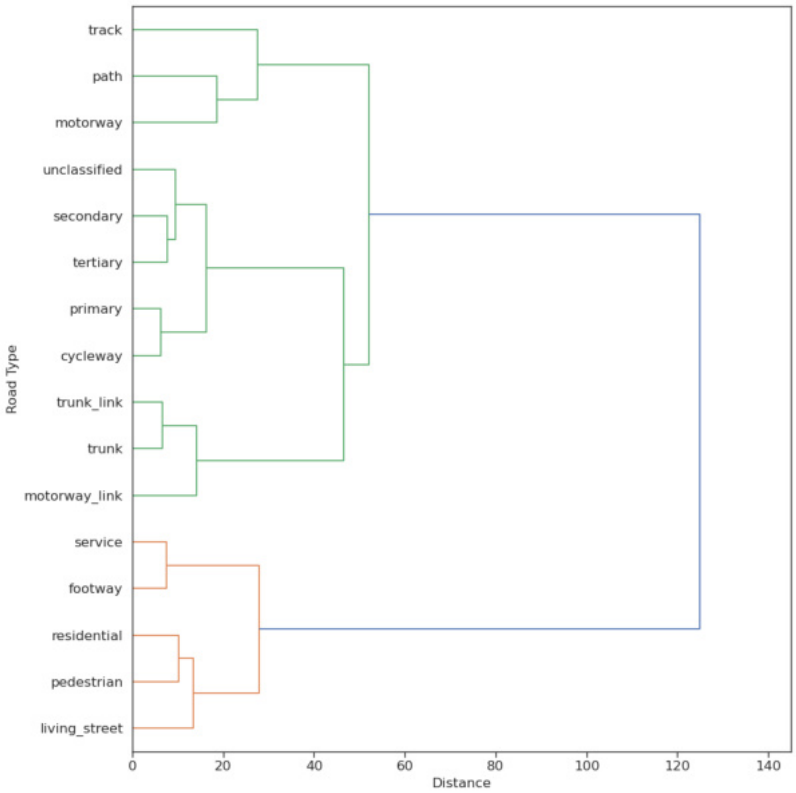


Figure 3.5: Dendrogram showing the HCA of road types based on their pattern similarity over LULC classes for an average over 0, 10, and 25 m buffers applied to the road network.

In order to effectively reduce the complexity of the road types, the results regarding the HCA have to be partially anticipated. Mainly, there are three large clusters (color-coded) on a high level, but a lot of low level clusters, which partly change depending on the applied buffer size. Atwal et al. (2022) merged

the road types *residential* and *living_street* based on their main relationship to the residential LULC class. This is adopted to this complexity reduction as a basis for clustering specific types. Because the LULC classes in the LULC Utility include built-up rather than residential areas, the *pedestrian* road type was also merged with the other two for its similar pattern over the LULC classes over all dendrograms and similar usage in OSM. In addition to these steps, because their total proportion of the total road network length is low (2.41 %) and they already have been handled differently because of their observable size, the roads with the types *motorway* and *trunk* as well as their links are merged into one class *highways*. They also show a similar relationship to LULC classes in the averaged dendrogram except for the road type *motorway*. Arguably, *footway* and *service* could also be merged for their similar pattern in all dendrograms, but as their usage in OSM is defined for slightly different purposes, they are kept separate. Some other low level clusters show greater variations in response to the buffers applied to the road network (e.g. *tertiary* and *unclassified* as well as *secondary* and *cycleway*), therefore, these road types are kept separate as well.

In summary, the concrete steps taken to reduce the road types complexity from 32 road types to 11 are as follows:

1. Disregard all road types with a global share of less than 0.2 % according to the OSM Taginfo homepage (OSM Taginfo, 2024), a reduction from 32 to 16 road types.
2. Merge *pedestrian* and *living_street* road types into *residential* due to their similar pattern over the LULC classes, a reduction from 16 to 14 road types.
3. Merge *motorway* and *trunk* road types as well as their links into one class *highways*, a reduction from 14 to 11 road types.

The input tensor of Extension 2 has a total of 16 tensor channels, the 5 from the baseline with additional 11 rasterized road types, some of them being a combination of multiple road types. Figure A.2 in appendix section A.3 shows the distribution of the road types in the AOI MA-HD-RNK after feature engineering and clustering.

3.5.4 Extension 3 - Encoded Speed Limits

The third extension aims to integrate an additional aspect of the OSM road network, specifically speed limits, in response to the first research question. The assumption here is that speed limits are an indication of adjacent LULC classes as well and, not exclusively, but still, road type dependent. This gives the possibility that they provide other information about adjacent LULC classes than using only road types. Studies by Levinson and Yerra (2005) and Levinson et al. (2007) have demonstrated a correlation between traffic flow and accessibility within road networks and the surrounding LULC classes, suggesting these elements evolve together. Furthermore, Y. Zhao et al. (2023) have identified the *maxspeed* tag in OSM road networks as a valuable indicator for estimating traffic flow. Therefore, this extension focuses on the *maxspeed* tag, which, as shown above in section 3.2.2, is not fully mapped in the OSM data.

In alignment with the second research question, some missing values can be filled by feature engineering and merging other tags of the road network, including *source:maxspeed, zone:maxspeed, zone:traffic* and *maxspeed:type* (called "speed tags" from this point on). These speed tags also include values like

DE:zone30 or *DE:rural*, but, interestingly, sometimes these values are mapped, although the *maxspeed* tag is empty. In these cases, these values can be converted to speed limit values to fill in missing values in the *maxspeed* tag. Still, there are also values like *sign* or *variable* in these speed tags, which have no useful meaning without additional values in other tags regarding the speed limit. In these cases, the values are disregarded and handled like *NaNs*. There are also values like *walk* or *default* or values including a country code (like *DE:*), which are country-specific and can be converted to speed limit values based on country-specific regulations. For these regulations, a mapping dictionary is utilized, which includes default speed limits for each road type, as well as country-specific regulations for speed limits and speed zones for specific country codes. Generalizing the country code (deriving it from values) increases the code complexity many times over, therefore, only a specified country code is supported by setting a configuration variable. The according mapping dictionary is referenced in appendix A.2. The feature engineering steps for *maxspeed* values and the speed tags if the *maxspeed* value is missing are as follows:

1. Replace all points with colons (":"), e.g. *DE.zone30* to *DE:zone30*. Points are usually wrongly mapped colons.
2. Remove everything before and including ":", e.g. *DE:zone30* to *zone30*.
3. Remove all occurrences of "zone", e.g. *zone30* to *30*. Remaining empty strings are replaced with *NaNs*.
4. *NaNs* and digits are kept, other values are converted using the mapping dictionary (either country-specific textual or the values *default*, *walk*, or *none*). Otherwise, disregard the values (this also applies to *variable* and *sign*).
5. The values of the speed tags are merged into *source:maxspeed* by using a specified priority order and then used to fill the missing values in *maxspeed*.

In the AOI MA-HD-RNK, a total of 528 missing values in *maxspeed*, a share of about 0.5 %, can be filled with values from the speed tags and another 318 converted from textual to integer values, which both add valuable information to the road network. After enhancing the *maxspeed* tag, the total length of roads with existing speed values is 4,231.83 km km, representing approximately 24 % of the total road length of 17,606.58 km, with 20 distinct speed limit values remaining in this AOI.

In order to further reduce the dependency on road types to achieve an even more different layer of information, a further step is taken to group the values together based on their expected meaning for adjacent LULC classes. Vitkienė et al. (2017) give an overview over different road classification systems in European countries. As also seen in the LCBB, roads are strongly regulated in Germany. Depending on their location (rural, built-up etc.) and their link function level (continental, regional, local etc.), they are assigned to different classes, which in turn have different restrictions like speed limits. Other countries like Denmark, Greece, Portugal, or Norway, distinguish the roads directly by speed limits, creating groups based on speed limit ranges.

This system is adapted in Extension 3 to group the values of the enhanced *maxspeed* tag into eight classes, effectively reducing the speed limit values. The resulting classes and their share in MA-HD-RNK can be seen in table 3.6. When generalizing the code and applying it to areas outside of Germany, the

class “unlimited” should be disregarded, because speed limits are enforced globally and do not exceed 130 km/h, therefore the tensor channel would stay empty.

Table 3.6: Distribution of Speed Limit Classes for MA-HD-RNK, adapted from the road classification system of Denmark (Vitkienė et al., 2017). A maximum of 255 is chosen due to the *int8* data type. The class “Unknown” is added to show the share of still missing values, it is not included as a separate class in the model. The percentage is the share of total road length (excluding “unknown”) of 4,231.83 km. Sorted by speed range.

Speed Limit Class	Range [km/h]	Length [km]	Percentage
Unlimited	“none” (130-255)	132.73	3.14
Very High+	110-130	136.70	3.23
Very High	90-110	361.76	8.55
High	70-90	18.49	0.44
Medium	50-70	265.73	6.28
Low	30-50	837.10	19.78
Very Low	10-30	2376.67	56.16
Walk	1-10	102.65	2.43
Unknown	NaN	13374.75	-

These speed limit classes are encoded similarly to road types, using OHE, creating a raster for each class and integrating these as channels into the input tensor. The input tensor of the third extension, therefore, has a total of 13 tensor channels, the 5 from the baseline with additional 8 rasterized speed limit classes. Figure A.3 in appendix section A.3 shows the spatial distribution of the speed limit classes in the AOI MA-HD-RNK.

3.5.5 Extension 4 - Ensemble of Extensions 2 and 3

The last extension combines the second and third extension, integrating both road types and speed limit classes into the input tensor and making it the largest extension. But, it also is the most informative one, as it includes all three layers of information about the road network, namely the presence of a road, its type, as well as its speed limit class. However, due to the large proportion of missing values in the speed limits and, therefore, reduction of information mostly to only the presence of a road and its type, the impact will mainly be similar to Extension 2.

Extension 4 boasts a total of 24 tensor channels, 5 from the baseline, 11 rasterized road types, as well as 8 rasterized speed limit classes.

3.6 Machine Configuration & Optimization

The used machine is a desktop PC running Windows Subsystem Linux (WSL) inside Windows 11, because the LULC Utility is developed for Linux Operating Systems (OSs) and a dual boot setup was unwanted by the author. An NVIDIA Graphics Processing Unit (GPU) utilizes the Compute Unified Device Architecture (CUDA) (NVIDIA Corporation, 2024), NVIDIA’s parallel computing architecture and programming framework, enabling significant acceleration of computing tasks by leveraging the

computational power of GPUs. This technology allows for parallel computing, where the Central Processing Unit (CPU) handles sequential tasks in combination with the Random Access Memory (RAM), optimized for tasks that run best as single threads, and the GPU takes on the compute-heavy tasks across its thousands of cores and its separate memory (Video Random Access Memory (VRAM)) simultaneously (NVIDIA Corporation, 2024; A. Zhang et al., 2023). Although tensors can be created on the CPU, handling them with the GPU massively speeds up the process, therefore, utilizing a CUDA-capable GPU is strongly advised (A. Zhang et al., 2023). Tables 3.7 and 3.8 show the configuration of the machine including the used CUDA device, the setup with WSL, as well as the listed or estimated maximal power consumption of the hardware.

Table 3.7: Hardware configuration of the machine. Power consumption is either the listed maximal value from the manufacturer or an estimation of the maximal consumption. Parts have not been acquired solely for DL applications, therefore, they are not harmonized. The total power consumption of these parts under full load is ~364 W.

Part	Model	Power Consumption
CPU	Intel Core i5 12500	65-117 W
RAM	32 GB DDR4 3200 MHz 30 GB assigned to WSL	~12 W
GPU VRAM	Zotac GeForce RTX 2070 SUPER amp 8 GB GDDR6	215 W
Drives	2x NVMe SSD with 1 TB & 256 GB 500 GB SATA-SSD (for LULC Utility)	~20 W

Table 3.8: Software configuration of the machine. A Windows 11 machine with an integrated WSL environment was used.

Configuration	Version
WSL	2.0.9.0
OS	Ubuntu 22.04.3 LTS inside Windows 11
Language	Python 3.11.4
Framework	PyTorch 2.0.1 PyTorch Lightning 2.2.2 Lightning 2.1.4 CUDA 11.7

When integrating the road network data, it is advised to use suitable data formats for handling the data, in order to reduce memory and disk space usage. During the normalization process of the LULC Utility, where the tensors with the integrated road network are created, the road network data is requested from the Ohsome API for each patch computed by the LULC Utility area descriptor module. After preprocessing and before rasterization, the road network is saved as a file in the cache directory. But, because loading cache files can be time consuming and, depending on the complexity of the road network, the files can grow quite large, the popular GeoJSON and CSV formats are not suitable for handling this data in the cache. Instead, the road network should be handled in a binary format which is much faster and smaller according to tests by Zaitsev (2019) and Rangaraj et al. (2022).

A small test was conducted for this caching task of road networks with the broadly used GeoJSON

format for geographical data as baseline and some of the formats proposed by Zaitsev (2019) and Rangaraj et al. (2022) as comparisons, namely CSV and the binary formats Feather, Parquet and Pickle. The test was conducted using the road network data for the AOI MA-HD-RNK. This test area is larger than a typical area from a patch; it was chosen to test the performance of the caching process with a larger road network and under stress conditions.

Table 3.9: Comparison of different formats for caching operations. Sorted descending by loading time, the most important aspect for the given task.

Format	Saving time	Loading time	File size (MB)
GeoJSON	3490 ms ± 1340 s	1570 ms ± 246 ms	107.3
CSV	2060 ms ± 64.8 ms	1110 ms ± 67.3 ms	35.5
Pickle	340 ms ± 30.1 ms	235 ms ± 27.8 ms	34.3
Parquet	265 ms ± 23.5 ms	186 ms ± 16.5 ms	14.0
Feather	241 ms ± 11.4 ms	185 ms ± 8.78 ms	16.2

The measurements in table 3.9 show that using binary formats like Feather, Parquet or Pickle result in much faster saving and loading times compared to GeoJSON and CSV, which aligns with the findings of Rangaraj et al. (2022) and Zaitsev (2019). Feather and Parquet also offer much smaller file sizes than the other formats, which gets important for the caching process of very large areas used in the model's training process (if the cache is not built beforehand). Using CSV and GeoJSON would have a bad impact on the model's runtime and required disk space of the cache. Based on this test and the findings of aforementioned literature, the Feather format was chosen for this this due to its slightly better performance in saving and loading times compared to Parquet. For the given caching process and aim of minimizing the computational impact, the speed of saving and loading is more important than the used space on disk, although the file size of the Feather file is still very small compared to CSV, GeoJSON and Pickle.

3.7 Evaluation metrics

In ML based LULC-M, there are some common metrics to evaluate the performance of the models. These metrics are used to compare the performance of different models or runs and to assess the quality of the predictions (Moharram & Sundaram, 2023; S. Zhao et al., 2023). The most common metrics used in LULC-M are the Overall Accuracy, the F_1 score, and the Intersection over Union. Additionally, regarding the resource consumption, along with runtime and epoch count needed for convergence, hardware utilization and energy consumption are monitored. The following sections will describe the metrics used in this study. In the LULC Utility, the package *TorchMetrics* (Torchmetrics, 2024c) is primarily used to calculate the metrics, because it offers seamless integration with *PyTorch*.

3.7.1 Performance

The metrics are derived from the relationship between correct (true) and incorrect (false) predictions, and whether these predictions are positive (class) or negative (not the class). Four key terms form the basis for these evaluation metrics: True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN), which often are visualized in a confusion matrix (see figure 3.6), plotting the predicted

values versus the labels (actual values). The TPs denote the total number of instances where the model correctly predicted a positive outcome. Similarly, the TNs represent instances where the model correctly predicted a negative outcome. On the other hand, the FPs and FNs represent instances where the model incorrectly predicted a positive and a negative outcome, respectively. These four values, in conjunction with the total number of samples, are used to compute various evaluation metrics that provide a comprehensive understanding of the model's performance (Basheer et al., 2022; Fawcett, 2006; Hu et al., 2020).

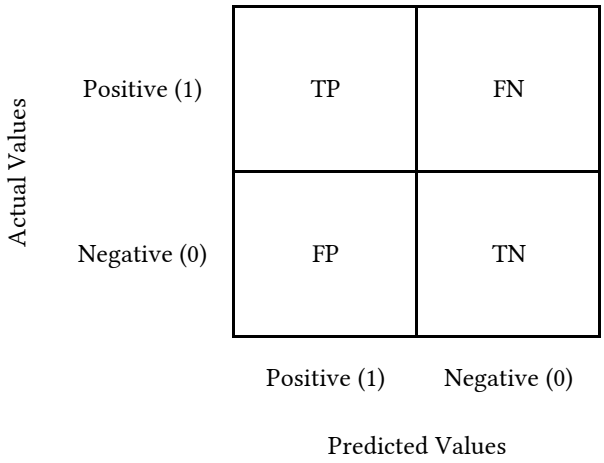


FIGURE 3.6: Conceptual representation of a confusion matrix, depicting the relationship between the predicted values and labels (actual values), adapted from (Fawcett, 2006).

Overall Accuracy

Overall Accuracy (OA) measures the proportion of correctly classified predictions out of the total predictions. It is a general measure of how well a classifier performs across all classes (Moharram & Sundaram, 2023; S. Zhao et al., 2023). It is calculated by dividing the sum of the TP and TN predictions by the sum of all predictions. The formula for OA, used by the *TorchMetrics* package (Torchmetrics, 2024a), is described as

$$\text{OA} = \frac{1}{n} \sum_{i=1}^{n} 1(y_i = \hat{y}_i), \tag{3.1}$$

whereas n is the total number of samples, y_i is the true label for the i-th sample, $\hat{y}_i$ is the predicted label for the i-th sample, and $1(y_i = \hat{y}_i)$ is an indicator function that equals one if $y_i = \hat{y}_i$, so when the prediction is correct, otherwise results to zero.

An alternative depiction of this formula including the aforementioned terms, seen in according literature like in Tzepkenlis et al. (2023) and S. Zhao et al. (2023), is

$$\text{OA} = \frac{TP + TN}{TP + TN + FP + FN}. \tag{3.2}$$

OA should be used with caution, especially when the classes are highly imbalanced, because it can be misleading in such cases. For example, it can increase in cases where the accuracy of a class increases more than it decreases of another class (Chicco & Jurman, 2020; Powers & Ailab, 2011). By solely

looking at the OA, this imbalanced behavior cannot be detected, which is why additional metrics are needed to evaluate the model's performance more accurately.

Class-Wise Accuracy

Because of the limitations of OA, Class-Wise Accuracy (CA) is used as an additional metric to split the OA into each class. It measures the proportion of correctly classified predictions for them (Fawcett, 2006). A confusion matrix, as shown above, shows the class-wise accuracy, but for multiple model runs, averaging these values is done by using the arithmetic mean

$$\text{CA} = \sum_{i=1}^{n} \frac{TP_i}{n}, \tag{3.3}$$

whereas n is the total number of model runs and TP_i the TP for the i-th sample/model run.

By looking at specific classes, conclusions about the used OSM filters can be drawn in addition to possible issues with the input data. For example, if the model has a high accuracy for the class *built-up*, but a low accuracy for the class *grass*, it can be assumed that the OSM filter for *grass* contains contradictory OSM tags or that the input data is not sufficient for the model to extract all the information needed to correctly predict the class. The CA, therefore, is more accurate for evaluating the accuracy of imbalanced datasets than the OA.

F_1 Score

Another common performance evaluation metric is the F_1 score. It is the harmonic mean of precision and recall, two metrics to measure the proportion of TPs in relation to the sum of certain types of predictions. Precision measures the exactness of a model, so how many of the TPs are actually TP. On the other hand, recall measures the model's completeness, so how many of all as positive predicted classes were correctly predicted (Hu et al., 2020; Mi & Chen, 2020; Mohammed & Kora, 2023; S. Zhao et al., 2023). The F_1 score is calculated by dividing the product of precision and recall by the sum of precision and recall. A high F_1 score near one indicates a good precision and recall, and a low value towards zero indicates the opposite. Basically, the F_1 score measures the model's accuracy, but differently than OA, for it takes into account how the data is distributed, therefore suitable for imbalanced data (Chicco & Jurman, 2020; Mi & Chen, 2020; Powers & Ailab, 2011; S. Zhao et al., 2023).

It should be reported alongside the OA to get a better understanding of the model's performance. This approach enhances the evaluation of both balanced and imbalanced data. Unfortunately, by using the "multiclass" parameter in the *TorchMetrics* package, the result is exactly the same as OA. Regarding these both, this results in ignoring the F_1 score and only using the OA for the results report.

Intersection over Union

The IoU, or as it is also sometimes called, the Jaccard Index or Jaccard Similarity Coefficient, is a metric that measures the overlap between the predicted output and the ground truth, divided by the

combined area of both (Csurka & Larlus, 2013; Demir & Musaoglu, 2023). It is defined as

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{TP}{TP + FP + FN} = \text{IoU}, \tag{3.4}$$

whereas A is the predicted output and B the ground truth (Demir & Musaoglu, 2023; Torchmetrics, 2024b).

Using the "multiclass" parameter, this metric is applied to each class separately and then averaged to get the overall IoU value. That is why this actually is the mean IoU, as showed in Minaee et al. (2022) and S. Zhao et al. (2023). The (mean) IoU value lies between 0 (no overlap between predicted value and label) and 1 (perfect overlap between predicted value and label). It functions as an alternative measurement of the accuracy and is particularly useful when the location of the prediction matters, for it compares segmentation patches with each other and not only classification values (Chicco & Jurman, 2020; Mi & Chen, 2020).

Loss & Accuracy Progression

Finally, the loss progression in comparison to the accuracy progression can also be used as a performance metric. As explained in section 2.5.2, the loss function is used to optimize the model's parameters during training, so the loss should decrease over time, while the accuracy should increase. Monitoring these for the training and validation sets, the models can be checked for their fitting behavior, and, therefore, the reliability of the other metrics and the overall training process.

3.7.2 Resource Consumption & Efficiency

A model can be using a lot of resources, having a high resource consumption, but still be efficient in its performance when utilizing these resources effectively. This means that if a model is using a lot of resources, but also has a high performance, it can be considered efficient, although a low resource consumption paired with a high performance would be even more efficient, of course. On the other hand, a model can be using a lot of resources, but still have a low performance, which would make it inefficient. Resource consumption and efficiency of a model are important to monitor, because they can tell if a model run is successful in reducing resource consumption while still achieving high performance (B. Li et al., 2021; J. Xu et al., 2021).

The consumption of energy, the emission of carbon, and the overall efficiency solely of the model are difficult to monitor without additional precautions and hardware. Mehlin et al. (2023) introduce different tools for this task, but point out that these tools oftentimes provide only an estimation and have to be reported alongside the geographic region and its infrastructure. Even then, a direct comparison and evaluation, for example of the carbon emission, is still imprecise. In addition to that, the accuracy of the model should also always be included, because achieving a lower energy consumption but also a lower accuracy, for example, is not a good trade-off (B. Li et al., 2021; Tao et al., 2022). Furthermore, if a code-dependent tool is used, the energy consumption can vary depending on the code and the machine it is running on (Mehlin et al., 2023). Nevertheless, there are some metrics that can be used to estimate the resource consumption and efficiency of the model, which are described below.

Runtime & Number of Epochs

The primary metrics for assessing a model's resource consumption and efficiency are its runtime and the number of epochs required for convergence. These metrics are a reliable indication, as long as the same hardware and software settings are used (Mehlin et al., 2023). However, as explained also in section 2.5.6, less epochs or a shorter runtime do not necessarily mean a more efficient model. A shorter runtime and comparable performance imply higher efficiency, while a higher number of epochs and a longer runtime imply a higher resource consumption regardless of the performance. Similarly, a low resource consumption alongside low performance are not a good trade-off and not efficient as well. The model's total runtime serves as a indicator of its overall resource consumption, as longer runtimes typically correlate with a higher energy consumption (B. Li et al., 2021; J. Xu et al., 2021). To enhance comparability, the time needed per epoch is calculated as well, enabling a better understanding of the model's efficiency in comparison to the performance.

Model Size

As showed in Mehlin et al. (2023) and Tao et al. (2022), the model size can also be important for the resource consumption of the model. It is usually directly correlated to the complexity of the model, therefore, needed resources to train and infer (J. Xu et al., 2021).

Additionally, although today a good internet connection and enough disk space are not a rarity anymore, handling and distributing models with multiple hundreds or thousands of MB can still have an indirect impact on the model's resource consumption. The framework of the LULC Utility is designed to offer an online storage of the model, so the larger the model, the more space is required on the server and cloud which themselves burn a lot of energy to offer this service. By keeping the model size low, these indirect resource consumptions can be reduced. The model size is automatically measured in MB and is the space taken up by the model on the online storage.

Energy Consumption

Regarding energy consumption, the LULC Utility has an energy tracker from the *pyJoules* package (PowerAPI, 2024) implemented, which measures the consumed energy in Millijoules. But, a large drawback of the measurement is, that it is not possible to monitor the energy consumption of the model alone. The tracker measures the energy consumption of the whole machine in its current state, which includes the main parts CPU, RAM, and GPU, but also cooling systems and drives, all drastically influenced by running software and the unoptimized harmonization of hardware. Running a machine with as few additional programs in the background as possible helps in getting a more accurate measurement of the model's energy consumption, but it will never be the exact amount the model has consumed. B. Li et al. (2021) and Mehlin et al. (2023) present other tools for energy consumption measurement as well, but all have similar drawbacks and limitations.

Because additional hardware for measuring energy could not be installed on the machine and the comparability could not be given due to the setup, this measurement is disregarded and replaced with the following one.

Hardware Utilization

Measuring the energy consumption of hardware parts without hardware-near tools is much harder than measuring their utilization, therefore, this is monitored instead. The CPU, GPU and its memory (VRAM), as well as the RAM utilization, are measured during the model run and their maxima and averages compared. The largest focus is on the GPU, for this part is mainly used for training the model and the largest consumer on the machine, as shown above in table 3.7. Additionally, because the CPU is the second largest consumer and also highly involved in successfully training the model, it is monitored alongside the GPU. RAM and VRAM play a minor role, because there should be a linear relationship between larger feature spaces and higher memory utilization. That is why reducing the complexity of the road network, as explained in the extension sections, is important. Still, their utilization is monitored to check for memory leaks or other issues, possibly explaining other performance or efficiency issues.

As measuring the energy consumption, the utilization of those parts can also be influenced by running software in the background. However, the monitoring is much more detailed with more tools at hand, therefore, using the same machine configuration for all the runs and minimizing background processes, the utilization should still give better tendencies on how much the model uses the hardware resources. This gives an good estimation on how difficult it is to train the model and how much computational power is needed, giving a good insight into the resource consumption of the models.

Carbon Emission

Finally, measuring the carbon emission depends on the energy source mix of the geographical location which the machine uses (Mehlin et al., 2023). Because this is difficult to determine, the carbon emission is not measured additionally in this study. B. Li et al. (2021) suggest using a “Greenness” score for evaluating the carbon emissions of the model, but this was redeemed unfeasible for the task at hand, because it consists of multiple energy measurements, which are disregarded. The presented metrics are good indicators for the efficiency and resource consumption of the different extensions to the model, therefore, carbon emissions will not be included in this assessment.

Chapter 4

Results

Three research questions have been put up in the context of the hypothesis, that "by adding information about the spatial context of LULC classes using a road network, the model convergence should be supported and the overall resource consumption reduced alongside an increase of the accuracy, as the model can make more informed decisions about the LULC classes".

The second research question, *"what is a suitable encoding method to integrate them into the feature space of the model"*, has been answered in the methodology chapter, partly based on the results regarding the first research question. Road types are reduced in their complexity beforehand by using a HCA and merging road types with similar relationships over LULC classes, and speed limits are feature engineered by merging multiple OSM tags together and categorizing the values into fixed speed limit ranges. Both attributes are encoded via OHE and integrated into the feature space of the LULC Utility.

Their suitability for the task at hand is shown by analyzing their relationship to LULC classes, fully answering the first research question *"which attributes of the OSM road network can provide the model spatial contextual information regarding LULC classes, and what is their relationship to adjacent LULC classes derived from OSM"*.

Finally, the third research question, *"how much does the integrated road network and its attributes impact the model's performance and resource consumption"*, is also answered in this chapter by presenting the results of the model runs and analyzing them.

4.1 Relationship Between Roads & LULC Classes

RQ 1: Which attributes of the OSM road network can provide the model spatial contextual information regarding LULC classes, and what is their relationship to adjacent LULC classes derived from OSM?

In order to give an answer to this question, the results of the explorative relationship analysis between road network attributes and LULC classes is presented. The analysis is divided into two sections according to the two attributes of the road network types and speed limits, respectively.

4.1.1 Road Types vs. LULC Classes

This section presents the results of the relationship analysis between road types and LULC classes. The analysis is split into two parts: assessing the relationship visually and statistically.

Visual Relationship

Figure 4.1 visualizes the distribution of four out of the 16 road types shown in table 3.1. They are plotted on top of the LULC classes before feature engineering them as explained in section 3.5.3. The remaining road types are shown in appendix section A.4.1.

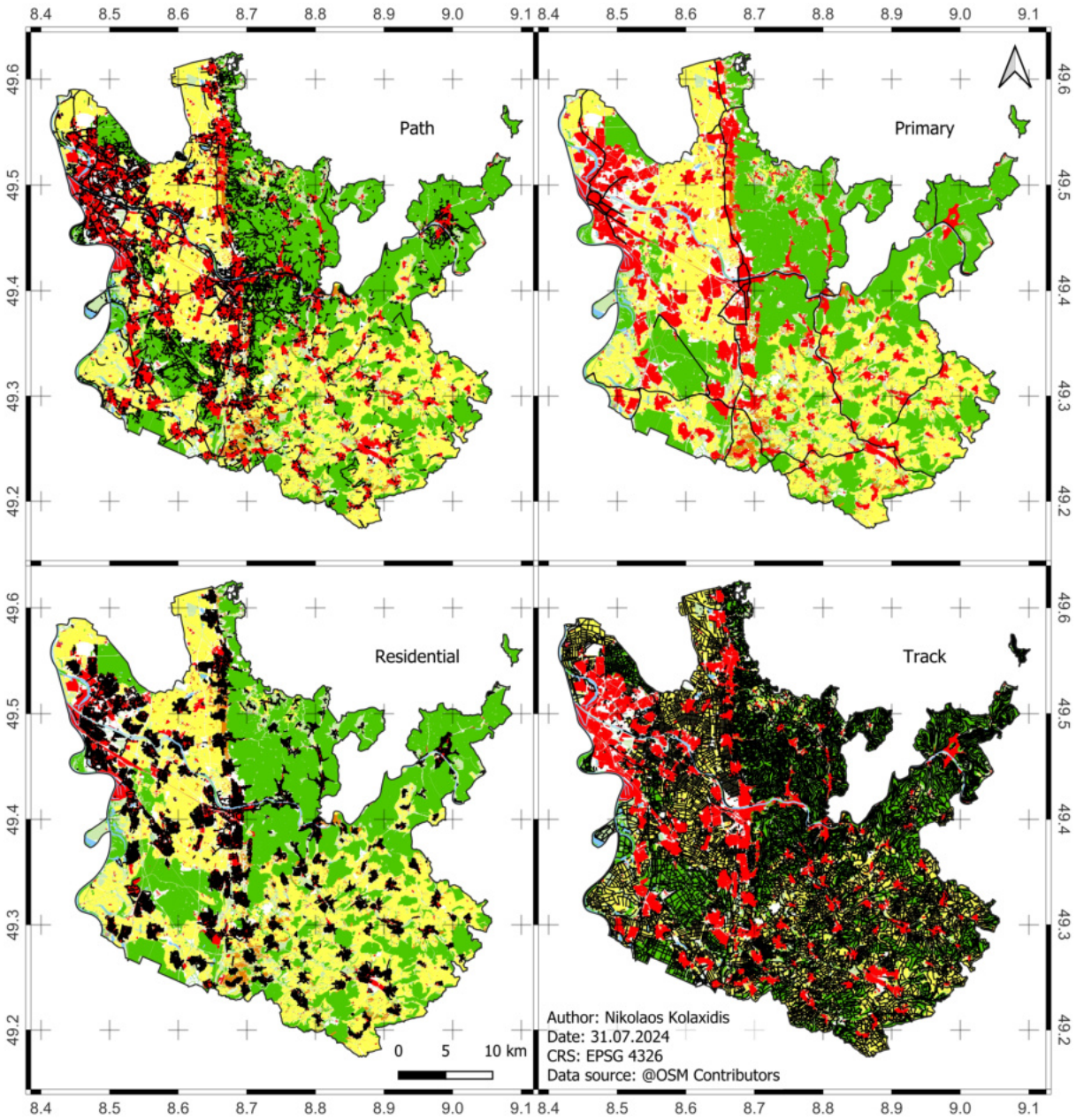


FIGURE 4.1: Visual Relationship between select road types and LULC classes for MA-HD-RNK. Additional road types are found in appendix section A.4.1. Colors represent LULC classes, whereas "built-up" is red, "forest" is green, "water" is blue, "farmland" is yellow, "permanent_crops" is orange, and "grass" is light green (as in figure A.1 in appendix section A.3). Roads are black.

The figures primarily show that the occurrence and density of specific road types is highly heterogeneous. Some road types cover nearly the full AOI, the high share of over 41 % of the *track* roads is clearly visible, while others are either very distributed or barely represented. Both links of *motorway*

and *trunk* (figure A.4) and types *cycleway* and *pedestrian* (figure A.5), all with a share of the total road network length of under 1 %, are barely visible, making it difficult to draw conclusions about their spatial relationship to LULC classes. All in all, the road network is more dense on the western part of the AOI and sparse in the southeast.

The major road types, including *motorway, trunk, primary, secondary*, and *tertiary*, are distributed across all LULC classes, although *primary* seems to rather occur in built-up areas and *motorway* rather in farmlands. The same applies for *path* and *unclassified*, although both are very granulated, *unclassified* more than *path. Path* is more distributed over all LULC classes, *unclassified* seems to rather avoid forest areas. Types *footway* and *service* are quite similar in their spatial distribution, whereas *service* covers more area, as it has a higher share.

Some roads seem to be bound to specific LULC classes. *Footway, living_street, residential*, and *service* seem to occur rather in built-up areas, whereas *track* seems to avoid built-up areas completely. It is noticeable that *track* and *residential* are nearly the opposite of each other. Interestingly, *path* is not similar to *track*, although the first impression would suggest that. Forest areas are usually avoided, except for *track* and *path*, although only *track* roads cover nearly all forest areas. Farmlands usually have a sparse network, except for *track* and some roads of the major road types.

So, at first glance, it seems that the road types *residential, footway, living_street*, and *service* are bound to built-up areas, while *track* is the opposite. The other road types do not seem to have a preference for a specific LULC class. This will be investigated further in the statistical relationship analysis.

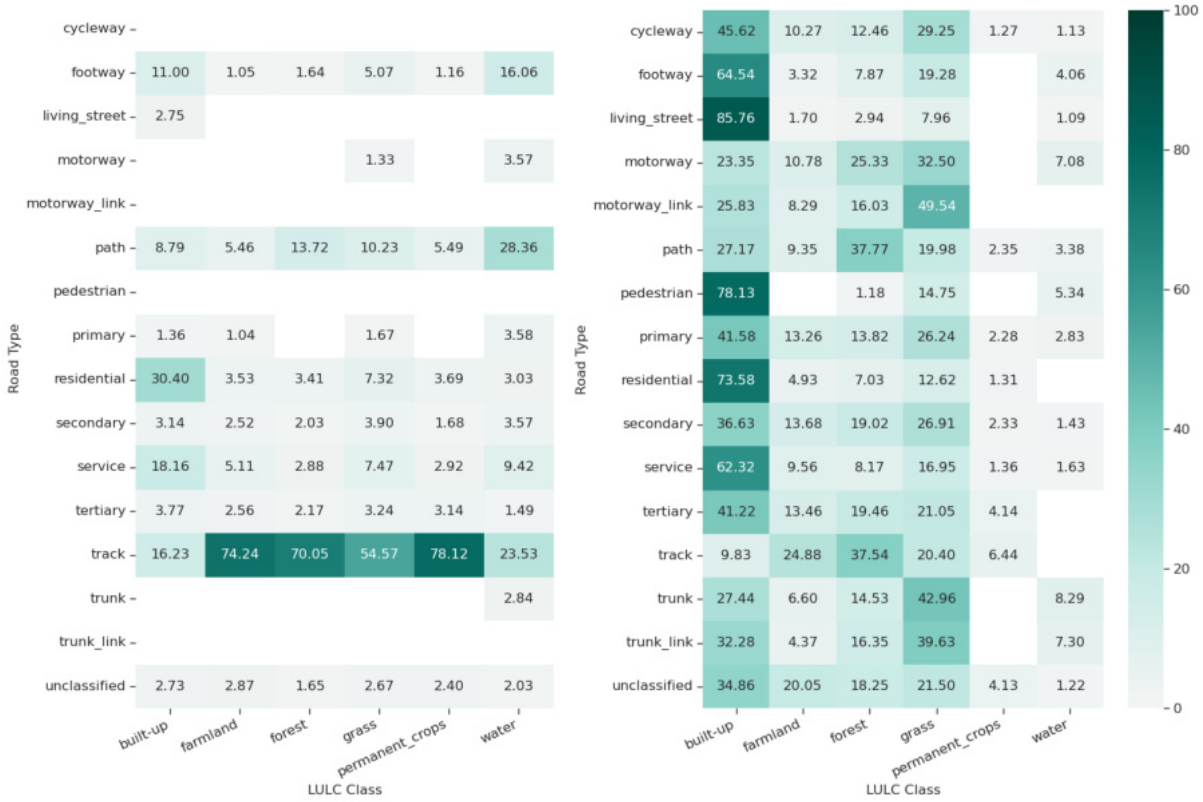
Statistical Relationship

The relationship between road types and LULC classes is further analyzed statistically. The results are presented in two heatmaps, showing the distribution of road types over LULC classes and vice versa (see figure 4.2). The heatmaps are based on the total road length per road type and LULC class, respectively. They are the averages of the applied buffers as explained in section 3.3 and as shown in figure A.6 and A.8 in the appendix section A.4.2. The according figures and the changes between them are not explained in detail, because for the model it only is important to see if there are differences in the signals between the road types for the LULC classes. It is sufficient to say that the buffers do have an impact on the distribution, but the average pattern represents the distributions well.

Figure 4.2a shows the quantified relative distribution of road types over LULC classes. As explained in section 3.3, the shares of the total road length per road type have been calculated per LULC class, so the figure shows which roads are more prevalent in each LULC class, based on the total road length in each LULC class. The highest values are achieved by *track*, covering over 70 % of farmlands, forests, and permanent crops, as well as over 54 % of grass. *Residential* has the highest share in built-up areas with 30.4 %, *path* in water areas with 28.4 %, followed by *track* with 23.5 %. A strong relation between road types and LULC classes (over 50 %) is only visible for the road type *track* for the three mentioned LULC classes as well as permanent crops.

Peculiarities are that 16.2 % of built-up areas are intersecting with *track*, which was not observable in the visual relationship, as well as the four types *path, track, footway*, and *service* near water areas, which also was not observable in the figure due to the small share of water areas in the AOI (1.7 %).

Also, although *service* has a high share in the road network (11.3 %), only slightly less than *residential* (14.2 %), it has not a similar relation to built-up like *residential*, it is more distributed towards water with otherwise similar values in the other LULC classes.



(A) Heatmap of Road Types per LULC Class. Summation to 100 % vertically.

(B) Heatmap of LULC Classes per Road Type. Summation to 100 % horizontally.

FIGURE 4.2: Heatmaps showing the relationship between road types and LULC classes, averaged over all buffer sizes (0, 10, 25 m; see appendix section A.4.2). The distribution depends strongly on the share of road types (A) and the area of LULC classes (B) in the AOI. All values under 1 % are omitted for clarity, summation includes the omitted values.

Because of the high share of *track* roads (41.3 %), the other road types have mostly low distribution values, so the figure is quite biased and, therefore, not very informative about the relationship between road types and LULC classes, except those mentioned. That is why for this data no HCA was conducted, as it would not provide any additional information other than *track* being largely different from the other road types.

In order to get a better understanding of the relationship between road types and LULC classes, the distribution is turned around. In figure 4.2b, the share of LULC classes adjacent to the road types is shown, summing up to 100 % horizontally based on the total road length of each road type. In this figure, the distribution gets more distinct. The prior very prevalent road type *track* has now a differentiated distribution over the LULC classes, having its highest intersection with the forest class with 37.5 %, but falling under 1 % in the water class and under 10 % in the classes built-up and permanent crops.

The only road types whose largest share (over 50 %) intersects with a specific LULC class are *footway, living_street, pedestrian, residential*, and *service*, which are the most prevalent in the built-up class. This

was also already shown in the HCA in section 3.5.3, where *living_street, pedestrian*, and *residential* road types were merged together into one class. These three have an even higher relation to built-up areas than *footway* and *service* with each over 70 %, whereas the other two are between 60 and 65 %. No other road type intersects over 50 % with a LULC class, only *motorway_link* does so with the grass class just below at 49.5 %.

The other road types have a more diverse distribution over the LULC classes. There are not many types whose share is the largest in one specific LULC class except for those mentioned above, but some road types split themselves over two or three specific LULC classes. *Cycleway, motorway_link, primary, secondary, trunk*, as well as *trunk_link* all have more than 25 % of them distributed over the LULC classes built-up and grass.

As assumed in section 3.5.3, the major road classes are distributed differently over the LULC classes, therefore not merging them in one class was the right decision. Especially *tertiary* and *unclassified* differ from the others with their largest shares distributed over the three LULC classes built-up, forest, as well as grass, and built-up, farmland, and grass, respectively. The largest share of *track* are distributed over farmland, forest, and grass, with its largest (37.5 %) over forest.

For the other merged road types, *motorway, trunk*, and their links, the relationships differ in some aspects. *Motorway* and *trunk* are similar for their largest share intersects with grass, but differ for their second and third largest share intersect oppositely with built-up and forest. *Trunk* and its link seem very similar, but *motorway* and its link have a large difference in the grass class, with the link having a share of 49.5 %, whereas *motorway* has only 32.5 %. It seems that the link is more prevalent in grass areas and the *motorway* in forest. The distributions of these road types could provide unclear signals to the model.

Comparing these values with the shares of LULC classes in the AOI, stronger relationships can be seen. Although forest has a share of 39.6 % of all LULC classes, only *motorway, path*, and *trunk* intersect to over 25 % with it. Contrary, although grass only has a share of 7.7 %, *cycleway, motorway, motorway_link, primary, secondary, trunk*, and *trunk_link* all intersect with it over 25 %, suggesting a stronger relationship. Water is even more underrepresented than in the other figure, as its share is only 1.7 %. Additionally, although farmlands have a share of 30.6 %, no road type intersects with it over 25 %. And, although built-up areas have a smaller share of 18.3 %, many road types intersect with it over 25 %.

In order to compare the pattern similarities of the road types to determine if specific signals for the LULC classes exist for the model to train on, a HCA has been conducted based on these results, as explained in section 3.5.3. Figure 4.3 combines the two distinct figures, the heatmap in figure 4.2 and the dendrogram in figure 3.5 into a clustermap, enabling the fast comparison of similar patterns of road types and LULC classes.

The first thing that catches the eye is that the most of the largest shares of road types are in the built-up and grass classes, although not always above 50 %. Additionally, *motorway, path*, and *track* have a large intersection with forest. For farmlands, only *track* has a large share. Permanent crops and water have only have shares under 10 %.

When looking on the left dendrogram, which shows similar patterns of road types, there are two major groups, splitting into the ones stronger associated with built-up and those with a more distributed pattern, as seen above. They are further split into smaller groups, based on their distributions over specific LULC classes. For the model, it would mean that there are primarily signals to better distinguish between built-up and not built-up areas, but also between grass and forest areas. The other two LULC classes are not as distinct, therefore it is expected, that for these LULC classes the performance will not be enhanced as much.

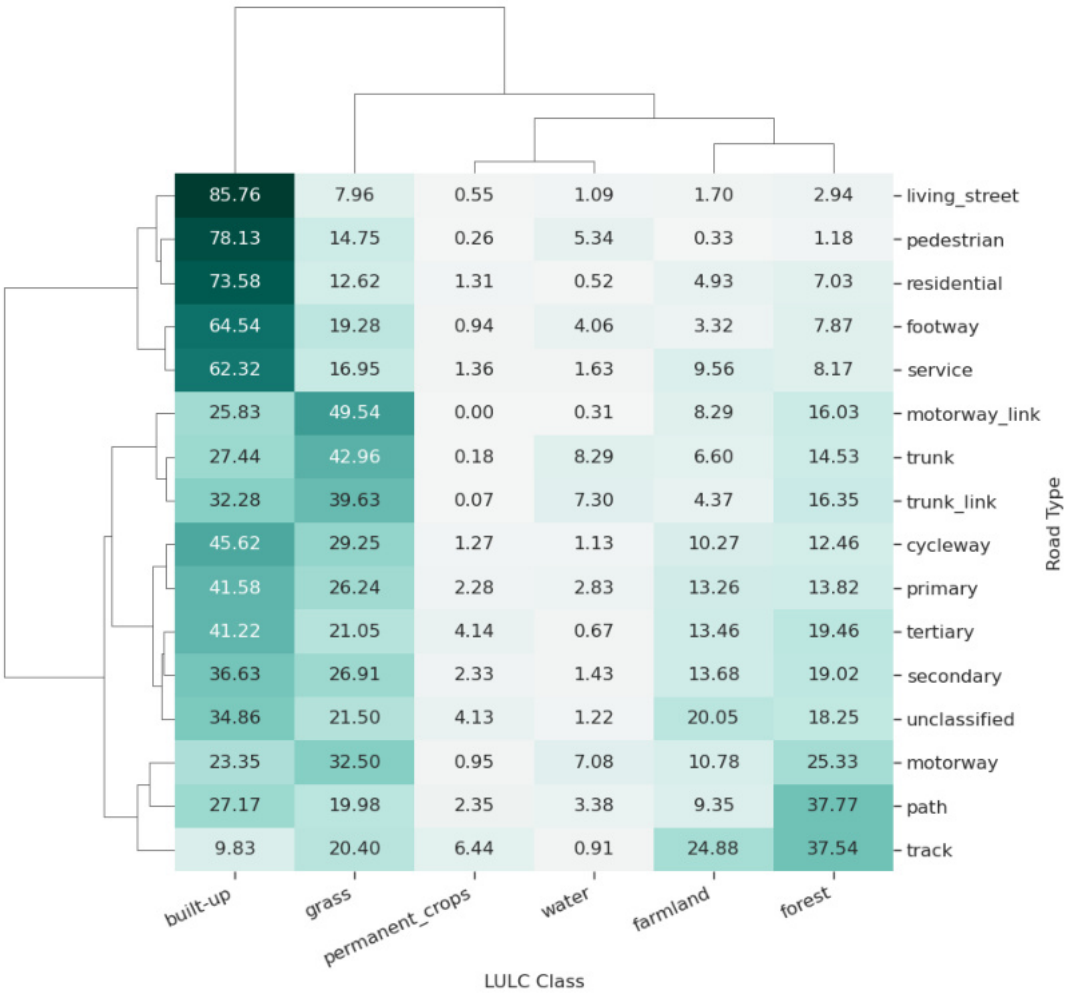


FIGURE 4.3: Clustermap showing the hierarchical clustering of road types, averaged over all buffer sizes (0, 10, 25 m; see figure A.10) based on their distribution pattern similarity over LULC classes (see figure 4.2b). The left dendrogram shows the clustering of the road types, the top dendrogram the clustering of the LULC classes. The nearer the lines of the dendrograms to the heatmap, the more similar the patterns (unitless).

Looking at the top dendrogram, which shows the pattern similarity between LULC classes, this assumption is confirmed. Most similar are water and permanent crops, then farmland and forest, grass is more different than these two groups together, and built-up the most different. For the model it would mean that it gets signals it can interpret, but their effectivity depends on the LULC class.

Because the clustermap is also influenced by the imbalanced LULC classes, a PPMC analysis has been conducted as well in order to see if LULC classes correlate with each other in their relationship to road

types, regardless of the length of road types or the area of LULC classes. Figure 4.4 shows the result of this analysis, again averaged for all buffer sizes. A positive relationship near 1 means that the classes are strongly linearly related to each other, a negative relationship near -1 that they are strongly opposed, and values around 0 impose a missing linear relationship.

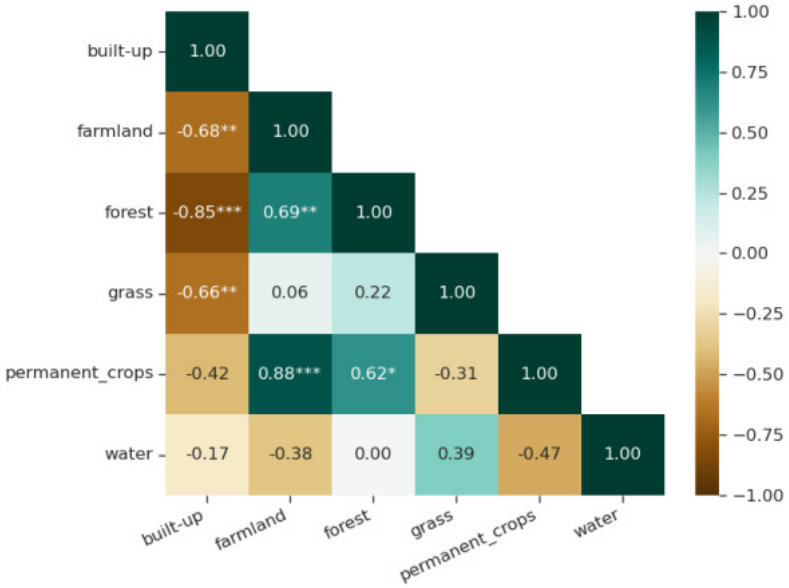


FIGURE 4.4: PPMC of LULC classes based on road type distribution, averaged over all buffer sizes (0, 10, 25 m; see figure A.11). Asterisks show according the significance level at 0.05 (*), 0.01 (**), and 0.001 (***), respectively. The upper triangle is omitted for clarity.

The PPMC confirms that there are linear relationships between the LULC classes, with the strongest positive one between farmland and permanent crops, and the strongest negative one between forest and built-up, as assumed in the elaborations above.

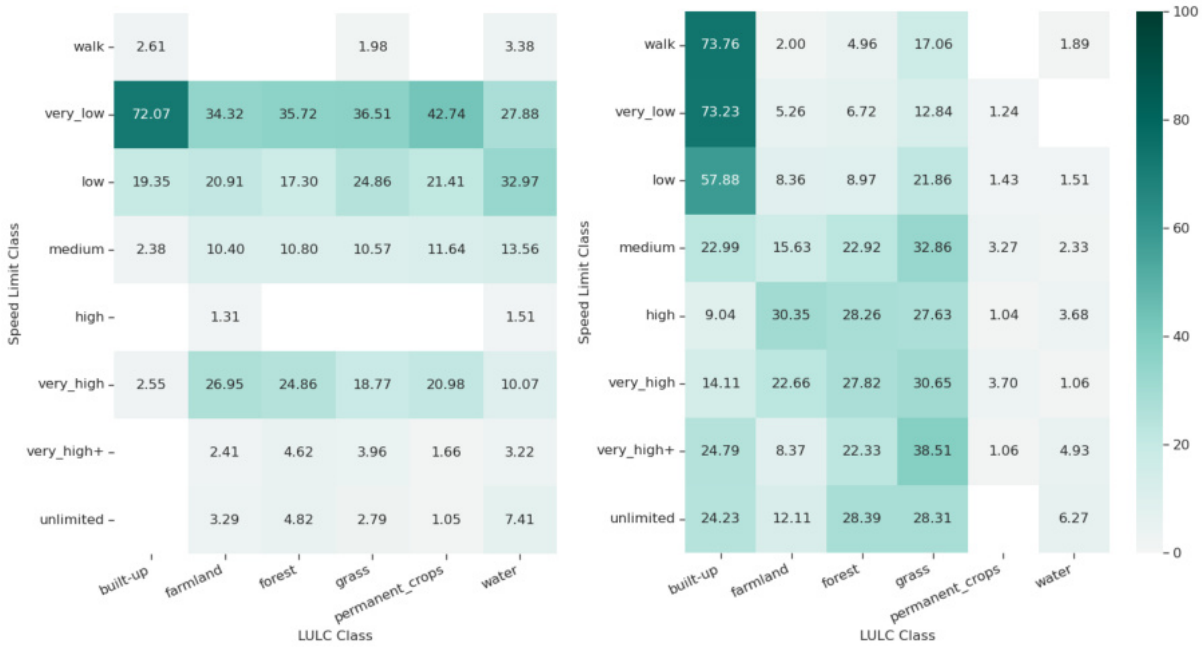
Interestingly, there is nearly no relationship between grass and farmlands, which is surprising, as they are both partially agricultural areas (*landuse=meadow* in grass class). Also surprising is the relatively strong relation between forest and farmland as well as permanent crops. This shows that there is a relationship between agricultural areas and forests based on the road network in them. Unsurprising is the negative relationship between built-up and all other classes, as this has been shown in all figures above.

So, all in all, it seems that road types can give the model according signals towards adjacent LULC classes, although not every class will be strongly positively impacted by the integration of the road types into the feature space of the LULC Utility.

4.1.2 Speed Limit Classes vs. LULC Classes

This section focuses on the relationship between speed limits and LULC classes, the second attribute of the road network that is investigated. Because of the many missing values, the visual relationship analysis is skipped. The statistical relationship is analyzed similarly to the road types by using two heatmaps, a HCA, and the PPMC analysis, showing the distribution of speed limit classes over LULC

classes and vice versa (see figure 4.5). Similar to analyzing the road types, the heatmaps are based on the total road length per speed limit class and LULC class, respectively. They are the averages of the applied buffers as explained in section 3.3 and as shown in figure A.12 and A.14 in the appendix section A.4.3. As previously, the according figures and changes between them are not explained in detail. The buffers also affect the distribution in this case, yet the average pattern adequately reflects the distributions as well.



(A) Heatmap of Speed Limit Classes per LULC Class. Summation to 100 % vertically.

(B) Heatmap of LULC Classes per Speed Limit Class. Summation to 100 % horizontally.

FIGURE 4.5: Heatmaps showing the relationship between speed limit classes and LULC classes, averaged over all buffer sizes (0, 10, 25 m; see appendix section A.4.3). The distribution depends strongly on the share of speed limit classes (A) and the area of LULC classes (B) in the AOI. All values under 1 % are omitted for clarity, summation includes the omitted values.

Figure 4.5a shows the relative distribution of the speed limit classes over LULC classes. As explained in section 3.3, the shares of the total road length per speed limit class have been calculated per LULC class analog to the procedure for the road types, so the figure shows which roads are more prevalent in each LULC class, based on the total road length in each LULC class.

Quite noticeable is the one high value of 72 % in the relation between the speed limit class "very low" and the built-up LULC class. "Very low" has the highest share in the road network with over 56 %, explaining its highest prevalence in every LULC class except water. The second highest share of the road network has the speed limit class "low" with nearly 20 %, which explains why it has high shares over the LULC classes as well. Interestingly, it intersects the most with water areas, surpassing even the class "very low". Although "very high" has the third largest share of the road network with 8.55 %, it is the second most prevalent speed limit class in farmlands and forests, suggesting an even stronger relationship towards these LULC classes. "Medium", with 6.3 % the fourth largest share of the network, is usually the fourth largest share in all LULC classes as well except for built-up. The

other speed limit classes have low values, with "high" and "walk" having the lowest shares in the road network, observable in this figure.

The assumption relating the speed limit classes is, that the further outside of cities, the higher the speed limit. Because of the imbalanced shares of speed limit classes this assumption can not be confirmed by only using this figure, that is why the figure next to it, figure 4.5b, is consulted. In this figure, again the share of LULC classes adjacent to the speed limit classes is shown, summing up to 100 % horizontally based on the total road length of each speed limit class.

Partially confirming the assumption, the largest shares of the three slowest speed limit classes "walk", "very low", and "low" intersect with the built-up class. But, nearly one quarter of each of the two fastest speed limit classes "very high+" and "unlimited" intersect with built-up also, showing a opposite distribution as expected. In both of them, the other shares mainly split over the forest and grass classes, although "very high+" has a shifted distribution with less of its roads in the LULC class forest and more in grass.

As before, due to the low shares of water and permanent crops, they are underrepresented. Additionally, only the speed limit class "unlimited" has a share of over 5 % intersecting with water areas, all other classes are below. The higher mid range speed limits "high" and "very high" have a more distributed pattern over the three LULC classes farmland, forest, and grass.

The figures show a diverse distribution, although the assumption, that lower speed limits are more prevalent in built-up areas and higher speed limits in rural areas, is partially confirmed. However, a linear pattern is not clearly observable, as the highest speed limit classes have a high share in built-up areas as well. All speed limit classes intersect with the LULC class grass, although the lower speed limits less than the faster ones. Farmlands and forests are mainly intersected by the mid range speed limits, especially "high" and "very high".

The HCA in figure 4.6 shows the pattern similarities of the speed limit classes. As the figure shows in the left dendrogram, the speed limit classes are mainly clustered into two groups, one with the slowest speed limits "walk", "very low", and "low", and the other with the faster speed limits. The most similar pattern have "walk" and "very low" with the highest share (over 70 %) of each in built-up areas and the second highest in grass. "Low" has also a similar pattern, but with a higher share in grass and a lower share in built-up.

Interestingly, "unlimited" is more similar to "medium" than to "very high" and "high", and these two more to "very high+" than to the other two high speed classes "very high" and "high". "Unlimited" and "medium", therefore, break up the assumption than high speed limits are found outside of built-up areas.

The top dendrogram shows that the LULC classes are clustered into two groups as well, one with the built-up class and the other with the other five classes, similar to the road types. The highest similarity have water and permanent crops, but this is not representative due to their small share of the total LULC area, as explained above. Farmland and forest seem more related to each other than to grass, repeating the pattern seen in the road types, although in this case grass belongs to the same subgroup.

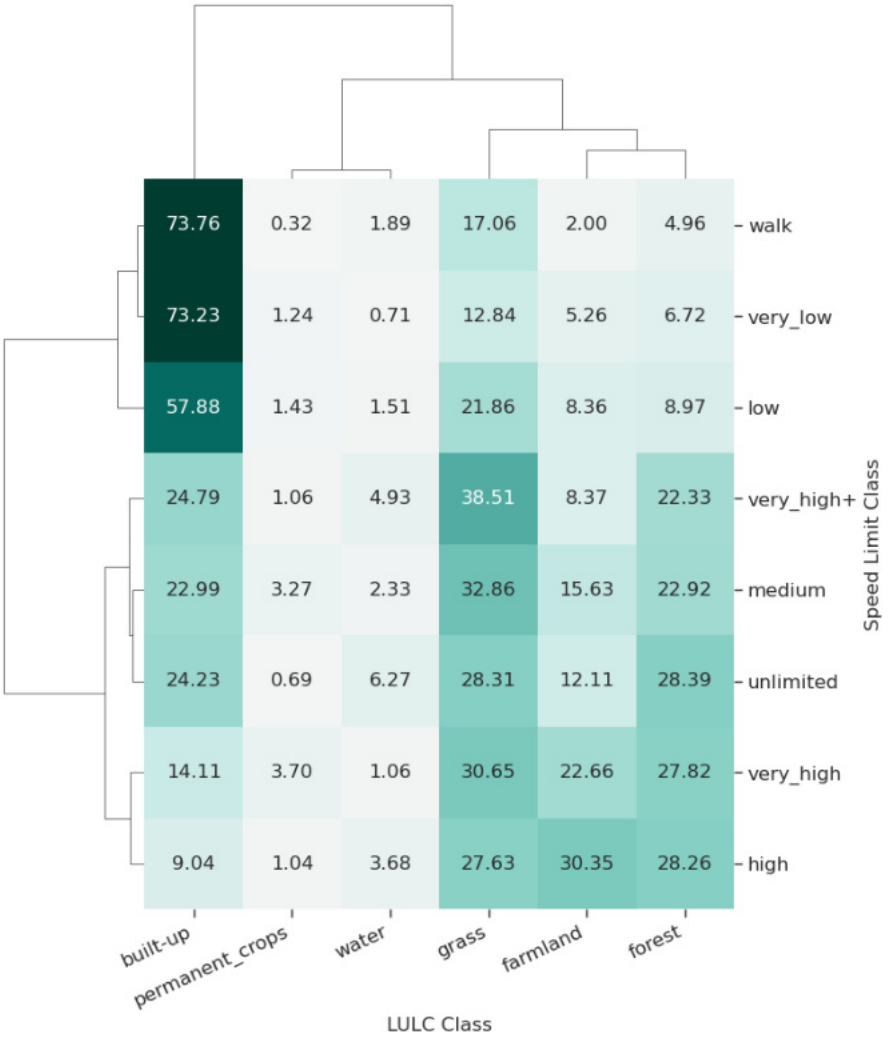


FIGURE 4.6: Clustermap showing the hierarchical clustering of speed limit classes, averaged over all buffer sizes (0, 10, 25 m; see figure A.16) based on their distribution pattern similarity over LULC classes (see figure 4.5b). The left dendrogram shows the clustering of the speed limit classes, the top dendrogram the clustering of the LULC classes. The nearer the lines of the dendrograms to the heatmap, the more similar the patterns (unitless).

The PPMC confirms that there are linear relationships between the LULC classes, with the strongest positive one, contrary to the road types, between forest and grass, closely followed by farmland and forest, and the strongest negative one at a significant -0.98 between forest and built-up, taking the statement of the road type analysis a step further. The correlation matrix generally looks different from the one for the road types, in general intensifying relations shown in the road types, like between built-up and the classes farmland, forest, and grass, also between forest and farmland, and between grass and forest. Surprising differences are especially in permanent crops and grass, which now have a positive relation, contrary to their prior negative relation with the same strength. Also, there is a relation between farmland and grass now, which was not observable before. Unsurprising is the negative relationship between built-up and all other classes, similar to the road types, as this has been shown in all figures above.

So, all in all, it seems that speed limits also can give the model signals towards adjacent LULC classes, but the signals are different from those of the road types. Some relations are similar and stronger, others opposite, so not every class will be strongly positively impacted by the integration of the speed limit classes into the feature space of the LULC Utility.

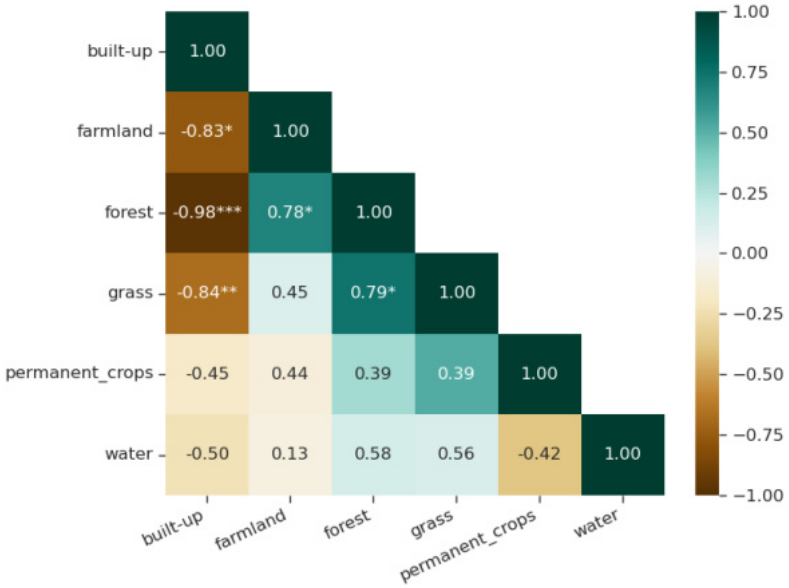


FIGURE 4.7: PPMC of LULC classes based on speed limit class distribution, averaged over all buffer sizes (0, 10, 25 m; see figure A.17 in appendix section A.4.2). Asterisks show the according significance level at 0.05 (*), 0.01 (**), and 0.001 (***), respectively. The upper triangle is omitted for clarity.

For Extension 4, the merge of both encodings, it is expected that the context inclusion will have a strong impact on built-up, as there are strong signals from both road types and speed limits. For the other classes, the impact is expected to be less strong, possibly even negative, as the signals might be contradictory, reducing the clarity of signals towards LULC classes.

4.2 Model Evaluation

RQ 3: What is the impact of the integrated road network and its attributes on the model's performance and resource consumption?

In section 3.7, multiple metrics have been presented in order to answer this question. The aim of this is to evaluate how the different extensions perform in comparison to the baseline runs and whether integrating a road network has a positive or negative impact on the performance and/or resource consumption of the model, if there is any. This should show if this approach is feasible and build the basis for further studies regarding spatial contextual information inclusion. The section above, where the results of the explorative relationship analysis between the road network attributes and LULC classes were presented, showed that there are signals the model could utilize to make more informed predictions. The following evaluation will show if the model can make use of these signals and if the signals are strong enough to enhance the model's performance or provide useful information to increase its efficiency.

The following sections present the metrics of the different models. Note that in all following tables, some cells are colored to enhance the perception. Depending on the metric, either the highest or the lowest value is the best. Usually for performance metrics, the higher the better, and the opposite for resource consumption metrics. The according best one will be marked green, and the according worst value will be colored yellow. Additionally, each cell will have an arrow indicating its performance in comparison to the baseline, which is an average of the three baseline runs. The arrows show the direction of the difference (increase/decrease), and the color the according rating (better/worse), as used for rating the best and worst value. The numbers of channels of each extension are added to the first two result tables to show their complexity in addition to the metrics for clarity. In subsequent tables, these will be omitted to reduce the table size.

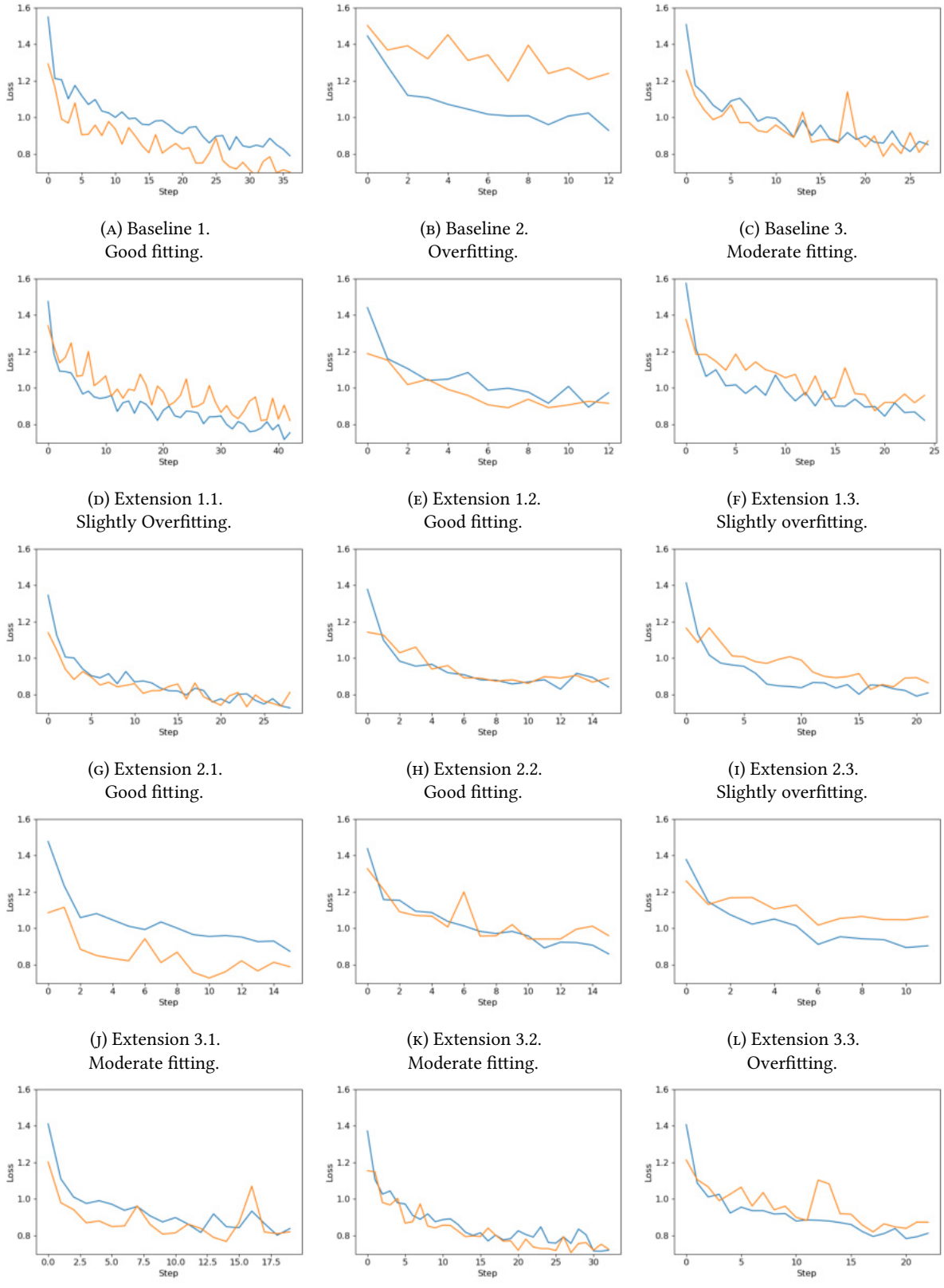
4.2.1 Overview over Resource Consumption & Efficiency

Before diving into the two metric groups and specific metrics, the general metrics of the model runs are presented in table 4.1 in an overview manner, showing first efficiency estimations. The assumption for the three general resource consumption metrics number of epochs, runtime, and model size, is, the lower the values, the lower the resource consumption, regardless of the performance of the model runs. The opposite is assumed for the performance metrics OA and IoU of the test phases, also shown in this table. The time needed per epoch is added for comparability, as it is dependent the number of epochs and the runtime.

The first thing to notice is that the baselines are neither the best nor the worst in all metrics. They have the lowest runtime, lowest time per epoch, and the highest OA score, but the other metrics are dominated by the extensions, especially the IoU, where only Extension 3.3 achieves a lower value than all baseline runs, and the number of epochs needed for convergence, where only three of the twelve extension runs needed more epochs than the baseline. Extension 4.2 has the longest runtime with 2231 s, more than triple the longest baseline run, and Extension 1.2 the shortest with 348 s, which is shorter than two of the baseline runs. The model size has a very high variability between 16.43 MB by Extension 3.3 and 115.58 MB by Extension 1.1, with the baselines ranging in between. The OA scores range between the highest value by Baseline 1 with 76.69 % and the lowest by Extension 3.3 with 64.31 %. The IoU scores are closer together, with the highest value achieved by Extension 2.2 with 39.29 %, better than all baseline runs, and the lowest by Extension 3.3 with 28.27 %, worse than all baseline runs.

In order to put the values into perspective, the fitting behavior of the model runs, monitored by comparing the loss of the training and validation phases and rating the curve progressions, is shown in figure 4.8. As the step count is not equalized, the curves are not directly comparable, but the general behavior can be observed and rated.

All training curves show an overall decrease, which means that they all achieved convergence. Some models did not perform well and overfitted the data, and some other models even have lower validation than training loss. This can be due to data augmentation and regularization techniques used in training and not in validation or having a too small validation set, which then happens to be an “easier” dataset to predict. Nevertheless, as the curves decrease and follow similar patterns, they signify successful model runs. Their different fitting behavior is mirrored in table 4.1, which shows that all groups incorporate



(A) Baseline 1.
Good fitting.

(B) Baseline 2.
Overfitting.

(C) Baseline 3.
Moderate fitting.

(D) Extension 1.1.
Slightly Overfitting.

(E) Extension 1.2.
Good fitting.

(F) Extension 1.3.
Slightly overfitting.

(G) Extension 2.1.
Good fitting.

(H) Extension 2.2.
Good fitting.

(I) Extension 2.3.
Slightly overfitting.

(J) Extension 3.1.
Moderate fitting.

(K) Extension 3.2.
Moderate fitting.

(L) Extension 3.3.
Overfitting.

(M) Extension 4.1.
Moderate fitting.

(N) Extension 4.2.
Good fitting.

(O) Extension 4.3.
Moderate fitting.

FIGURE 4.8: Loss of the training (blue) and validation (orange) phase for each model run. As the number of epochs is not equalized, the comparability between the model runs is limited. The fitting behavior is categorized in good, moderate, and overfitting.

TABLE 4.1: Overview over the general performance (test sets) and resource consumption metrics for all model runs. The baseline runs are averaged and the extension runs compared to this average. The assumption for the resource consumption metrics number of epochs, runtime, and model size is, the lower the values, the lower the resource consumption. The opposite is assumed for the performance metrics OA and IoU. Time per epoch is added for comparability. In green the best value per metric and in yellow the worst (if baseline runs are one of them, the according extension run is colored as well). The arrows show the direction of the difference (increase/decrease).

Model Run	Channels	Epochs	Runtime [s]	Time/Epoch [s]	Size [MB]	OA [%]	IoU [%]
Baseline 1	5	37	691	18.68	83.01	76.69	36.42
Baseline 2	5	13	269	20.69	30.79	71.85	34.24
Baseline 3	5	28	536	19.14	65.28	66.34	29.68
Baseline	5	26	498.67	19.50	59.69	71.63	33.45
Extension 1.1	6	43 ↑	1030 ↑	23.95 ↑	115.58 ↑	72.57 ↑	37.24 ↑
Extension 1.2	6	13 ↓	348 ↓	26.77 ↑	35.55 ↓	70.74 ↓	35.34 ↑
Extension 1.3	6	25 ↓	609 ↑	24.36 ↑	68.02 ↑	68.86 ↓	38.68 ↑
Extension 2.1	16	29 ↑	1444 ↑	49.79 ↑	40.47 ↓	68.90 ↓	37.42 ↑
Extension 2.2	16	16 ↓	826 ↑	51.63 ↑	20.98 ↓	70.56 ↓	39.29 ↑
Extension 2.3	16	22 ↓	1099 ↑	49.95 ↑	29.62 ↓	72.62 ↑	37.44 ↑
Extension 3.1	13	16 ↓	703 ↑	43.94 ↑	21.26 ↓	69.73 ↓	34.02 ↑
Extension 3.2	13	16 ↓	680 ↑	42.50 ↑	21.73 ↓	69.37 ↓	33.80 ↑
Extension 3.3	13	12 ↓	535 ↑	44.58 ↑	16.43 ↓	64.31 ↓	28.27 ↓
Extension 4.1	24	20 ↓	1405 ↑	70.25 ↑	24.54 ↓	74.63 ↑	38.61 ↑
Extension 4.2	24	33 ↑	2231 ↑	67.61 ↑	38.41 ↓	66.20 ↓	37.01 ↑
Extension 4.3	24	23 ↓	1583 ↑	68.83 ↑	27.00 ↓	68.01 ↓	38.14 ↑

a high variability between their runs. Nevertheless, the models are all included in the comparison as they all have a similar fitting behavior in general per extension group and use the same overall dataset, architecture, and hyperparameters, which means they all have the same limitations.

Averaging the values per baseline and extension, respectively, ensures better comparability and seems feasible based on the high variability in every group around a somewhat average-like medium model. Therefore, from this point on, only the averaged metrics will be analyzed in regard to the research question, where the extension groups are more important rather than single model runs for evaluating the extensions' impact on the model's performance and resource consumption. It should kept in mind that for all metrics, the variability is high. The metrics of all model runs are found in the appendix section A.5. Additionally, the plotted curves of the performance metrics OA and IoU for the training and validation phases for each model run can be found in the appendix sections A.5.3 and A.5.4, respectively, as well. These will also not be discussed here and function only as verification for claimed results.

Continuing in table 4.2, the metrics from table 4.1 are averaged per extension and compared to the baseline. As seen here, the main tendencies in comparison to the baseline do not change much, but it is clearer now that the extensions perform differently among themselves. Extension 3 is interesting for it has the lowest resource consumption values, but also the worst performance metrics. As the time per epoch also is much higher than the baseline or even Extension 1, it has a low efficiency and provides a

bad trade-off between performance and resource consumption, signifying a failed attempt to enhance the model. Extension 4, on the other hand, has the worst runtime and time per epoch, but performs comparably good, achieving the second best IoU. This also is a bad trade-off, as achieving comparable performance with higher computational costs is a failed enhancing attempt as well. Extension 1 and 2 seem to achieve better results, as they perform better than the baseline in regard to IoU and comparable in OA (less than 1 % lower), while having higher computational costs. So, they achieve an increase in performance with increased computational costs, a usual trade-off.

TABLE 4.2: General performance (test sets) and resource consumption metrics grouped and averaged by extension, compared to the baseline.

Extension	Channels	Epochs	Runtime [s]	Time/Epoch [s]	Size [MB]	OA [%]	IoU [%]
Baseline	5	26	498.67	19.18	59.69	71.63	33.45
Extension 1	6	27.00 ↑	662.33 ↑	24.51 ↑	73.05 ↑	70.72 ↓	37.09 ↑
Extension 2	16	22.33 ↓	1123.00 ↑	50.29 ↑	30.36 ↓	70.69 ↓	38.05 ↑
Extension 3	13	14.67 ↓	639.33 ↑	43.58 ↑	19.81 ↓	67.81 ↓	32.03 ↓
Extension 4	24	25.33 ↓	1739.67 ↑	68.68 ↑	29.98 ↓	69.61 ↓	37.92 ↑
Extensions	-	22.08 ↓	1066.08 ↑	48.28 ↑	38.05 ↓	69.71 ↓	36.75 ↑

Interestingly, the number of epochs and the model size of all extensions are much lower except for Extension 1. Similarly, all IoU values are higher than the baseline except for Extension 3. Comparing these values to the complexity of the models derived from the number of channels, it is difficult to observe a linear relationship between complexity and resource consumption or performance. Model sizes and epochs, for example, both are the worst in Extension 1 and best in Extension 3, but for Extension 2 and 4, the increase/decrease relation is turned around, giving a example against a linear relationship.

Evaluating the model runs solely on these metrics is not enough to make final statements about the impact of the road attributes on the model. Therefore, the following sections will dive deeper into the according metrics to better evaluate the impact of the integrated road network and its attributes on the model.

4.2.2 Performance

Looking further into the performance of the models, the maximal OA and IoU values for each phase of the model training process are shown in table 4.3. The prior tendencies are changed completely, as the extensions perform nearly always better than the baseline, except for the OA in the test phase, of course. There are single exceptions in every phase metric, usually in Extension 3, which performs worse in four of six metrics compared to the baseline, but the extensions usually perform better than the baseline in all other cases, usually in four to five out of the six cases.

Extension 4 has the highest values in the training and validation phases in both metrics, which makes sense compared to it being the most complex extension with the highest runtime. Although Extension 1 is less complex than Extension 3, it still achieves better performance in all cases except IoU during validation, while having a slightly longer runtime. Extension 2, being the second most complex extension and also having the second longest runtime, achieves always the second best value of the extension

except for the IoU in the test phase, where it surpasses Extension 4. Without Extension 3, a linear relationship is visible, where the metrics increase the higher the complexity, but as Extension 3 has the worst values and the IoU in the test phase of Extension 4 is lower than Extension 2, this assumption cannot be considered valid.

TABLE 4.3: Maximal achieved values for OA and IoU in all three phases, grouped and averaged by extension. The full table for all model runs can be found in table A.1.

Model Run	Train OA [%]	Val OA [%]	Test OA [%]	Train IoU [%]	Val IoU [%]	Test IoU [%]
Baseline	66.46	68.52	71.63	35.10	35.37	33.45
Extension 1	66.92 ↑	69.94 ↑	70.72 ↓	36.40 ↑	34.55 ↓	37.09 ↑
Extension 2	68.34 ↑	73.69 ↑	70.69 ↓	37.63 ↑	39.84 ↑	38.05 ↑
Extension 3	65.86 ↓	69.77 ↑	67.81 ↓	34.82 ↓	36.55 ↑	32.03 ↓
Extension 4	68.36 ↑	75.53 ↑	69.61 ↓	38.19 ↑	40.44 ↑	37.92 ↑
Extensions	67.37 ↑	72.48 ↑	69.71 ↓	36.26 ↑	38.05 ↑	36.52 ↑

In addition to comparing the maximal values achieved by the extensions, table 4.4 shows the metrics at step 13, the lowest number of epochs needed for convergence in the baseline runs (see table 4.1). This allows for the evaluation of performance at identical steps, under the premise that the models were ultimately exposed to the same amount of data for learning.

TABLE 4.4: OA and IoU values at step 13, grouped and averaged by extension. The full table for all model runs can be found in table A.2.

Model Run	Train OA [%]	Val OA [%]	Train IoU [%]	Val IoU [%]
Baseline	61.98	57.24	31.71	27.26
Extension 1	63.78 ↑	64.77 ↑	33.05 ↑	30.93 ↑
Extension 2	65.83 ↑	66.07 ↑	35.59 ↑	35.32 ↑
Extension 3	64.54 ↑	62.96 ↑	33.67 ↑	33.46 ↑
Extension 4	65.63 ↑	65.34 ↑	35.64 ↑	34.45 ↑
Extensions	64.95 ↑	64.79 ↑	34.49 ↑	33.54 ↑

Surprisingly, the baseline has the worst values in all phases and for both metrics, often by a magnitude of multiple percent points (%p) difference to the lowest performing extension. As for the best performing extension at step 13, Extension 2 achieves the best values in all phases except in the IoU during the training phase, where it performs only slightly worse than Extension 4. Extension 1 performs the worst among the extensions, with the exception of the OA during the validation phase, where only Extension 3 performs worse. The extensions perform better than the baseline in all cases without exception, but, as the time per epoch metric in table 4.2 shows, they all will have consumed more resources at step 13 than the baseline. Yet, the better performance is a good trade-off, showing the potential of the extensions to at least enhance the model's performance based on the same amount of data.

As OA and IoU are merged metrics, calculated using all values of the confusion matrices, the averaged TP values for each LULC class are shown in table 4.5. These values represent the CA metric, which helps

in comparing class-wise performance. The according confusion matrices of which the CA is derived from can be found in the appendix section A.5.2.

TABLE 4.5: CA values of the test phases, grouped and averaged by extension. The full table for all model runs can be found in table A.3 in appendix section A.5.1. Also, the according confusion matrices can be found in appendix section A.5.2.

Extension	Built-up	Forest	Water	Farmland	Permanent Crops	Grass
Baseline	70.33	84.33	78.00	69.00	43.00	41.33
Extension 1	82.00 ↑	84.67 ↑	86.33 ↑	74.00 ↑	51.00 ↑	34.67 ↓
Extension 2	86.00 ↑	84.00 ↓	91.67 ↑	68.33 ↓	56.67 ↑	39.33 ↓
Extension 3	79.00 ↑	87.00 ↑	84.00 ↑	57.00 ↓	53.33 ↑	20.33 ↓
Extension 4	87.67 ↑	82.67 ↓	90.67 ↑	64.33 ↓	67.00 ↑	37.33 ↓
Extensions	83.67 ↑	84.59 ↑	88.17 ↑	66.67 ↓	57 ↑	32.92 ↓

The extensions perform better than the baseline in four of the six LULC classes, except farmland and grass. The baseline then again performs best in the LULC class grass, and worst in the classes built-up, water, and permanent crops. As shown above, the relationship between the road attributes were the most evident towards the built-up class, which experiences a considerable increase in prediction accuracy in all extensions. The same applies to the classes water and permanent crops. In the forest class, extensions and baseline perform similarly. In the grass class, all models achieve a performance below 50 %, the extensions averaged even under 33 %, making it by far the worst predicted class. Interestingly, integrating only the road network without further data (Extension 1) already improves the prediction of all LULC classes except grass.

The extensions perform similarly in the farmland class, where the CA is worse than the baseline except in Extension 1, which outperforms the baseline by 5 %. However, the average CA of the extensions is over 65 %, therefore, the overall prediction is much better than in the grass class. The class permanent crops also has a subpar CA in the baseline with 43 %, but the extensions achieve a considerable increase in performance, especially Extension 4, which achieves by far the highest value in this class with 67 %. Interestingly, every extension performs the best in one of the classes, although Extension 4 holds the highest value in the two classes built-up and permanent crops. Because of this distribution, a linear relationship between complexity and performance again cannot be detected, as the most complex extension does not perform the best in all classes. The only extension which performs better than the baseline while all other extensions perform worse is Extension 1 in the farmland class.

Extensions predict water the best, followed by forest and built-up, although the range of CA differs in them. Of these three, forest has the smallest variation with 4.33 %p, then water with 7.67 %p, and finally built-up with the largest variation of 8.67 %p. The other three classes, which are predicted worse, also have a much higher variability between 16 %p in permanent crops to 19 % in grass. This shows that the model is more stable in predicting the classes water, forest, and built-up, while the other classes are more difficult to predict and have a higher variability in the predictions.

4.2.3 Hardware Utilization

Although some resource consumption metrics have already been explored in regard to the research question, the utilization of the hardware components is analyzed below, replacing energy consumption and carbon emission measurements. Again, all following tables show averaged values per extension group for better comparability with the baseline.

Because they are the two highest consumers in the machine, the utilization of GPU and CPU is looked into first. Their utilization in percent is shown in table 4.6. Contrary to the values above in table 4.1, the resource consumption in form of hardware utilization seems to be the highest in the baseline, with an exception in Extension 2, which reached a higher maximal GPU utilization. However, all cases in the table have a lower utilization, usually more than five %p, except for the mentioned value and the same metric for Extension 3, which reached the same maximal utilization of the GPU as the baseline.

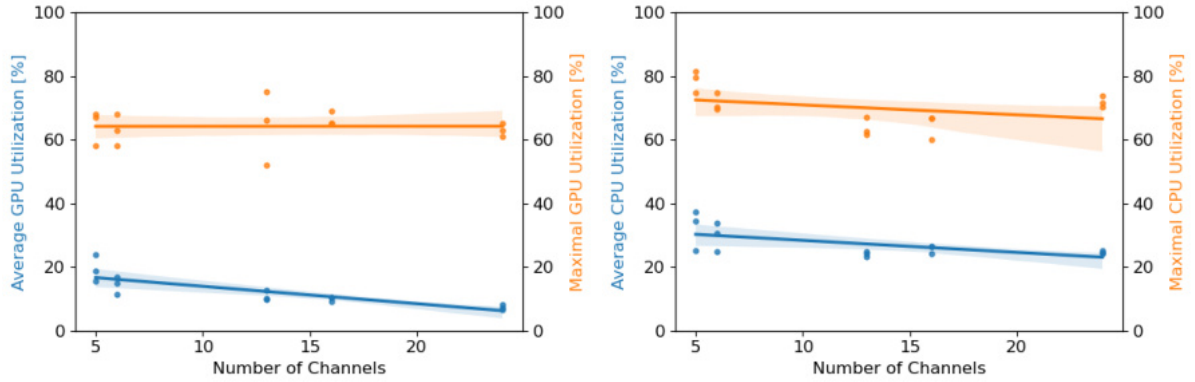
TABLE 4.6: Average and maximal utilization of GPU and CPU, grouped and averaged per extension. The values are compared to the baseline. Ø denotes the average value, and Max the maximum value. The full table for all model runs can be found in table A.4.

Extension	GPU Ø [%]	GPU Max [%]	CPU Ø [%]	CPU Max [%]
Baseline	19.40	64.33	32.30	78.60
Extension 1	14.37 ↓	63.00 ↓	29.86 ↓	71.47 ↓
Extension 2	10.08 ↓	66.33 ↑	25.71 ↓	64.47 ↓
Extension 3	10.90 ↓	64.33	24.07 ↓	63.87 ↓
Extension 4	7.32 ↓	63.00 ↓	24.65 ↓	71.83 ↓
Extensions	10.67 ↓	64.17 ↓	26.07 ↓	67.91 ↓

The expectation would be that the more complex the model, the more are GPU and CPU utilized. But, the values show a different picture, as the most complex extension, Extension 4, utilized the GPU the least both maximal and averaged, and Extension 1 the most averaged. Also, despite having more than the double amount of channels, Extension 3 utilizes the CPU less than Extension 1 in both cases, which definitely negates the expectation of a linear increase of hardware utilization with complexity and also its opposite, at least regarding the largest consumers.

In order to investigate a possible linear relationship, a regression plot is used, shown in figure 4.9, which uses the utilization values of all model runs as seen in table A.4 in appendix section A.5.5. The regression plot shows the relationship between the complexity, denoted as the number of channels, and the average and maximal GPU and CPU utilization.

As seen in table 4.6, it is difficult to pinpoint a linear relationship between complexity and hardware utilization. This can be seen in the regression plots as well, as either the variability is high or some extensions behave contrary to a linear relationship. The maximal GPU utilization stays on a similar level for all groups, although two runs of Extension 3 with 13 channels have comparably high and low values, respectively. The maximal CPU utilization is non-linear, as Extension 2 (16 channels) and 3 both utilize it less than Extension 1 and 4, both with lower and higher complexity, respectively. The CPU is utilized in average similarly by Extensions 2 to 4, only the baseline and Extension 1 utilize it more. The average GPU utilization is similar for all extensions with a slight negative linearity, although Extension



(A) Regression Plot of GPU Utilization.

(B) Regression Plot of CPU Utilization.

FIGURE 4.9: Regression plots showing the relationship between number of channels (equates to feature space size and extension type) and average (blue) and maximal (orange) GPU (A) and CPU (B) utilization.

1 has a higher variability than the other extensions. The baseline utilizes the GPU the most. So, a linear relationship is not really present due to a combination of high and missing variability in the runs and between the extensions.

The other two important hardware components for DL training, as shown in table 3.7, are RAM and VRAM. The utilization of these two is shown in table 4.7. However, contrary to GPU and CPU, the values are in Gigabytes. This is the first and only table which shows a clear relation between complexity and utilization in all metrics.

The baseline utilizes both RAM and VRAM the least, while Extension 4 utilizes them the most. Extension 4, also, has a peak RAM utilization of 28.30 GB, just below to maximal allocation for the WSL at 30 GB, as shown in table 3.7. Surprisingly, the VRAM utilization peaks just under four GB in Extension 4, which is only half of the maximal allocation of 8 GB.

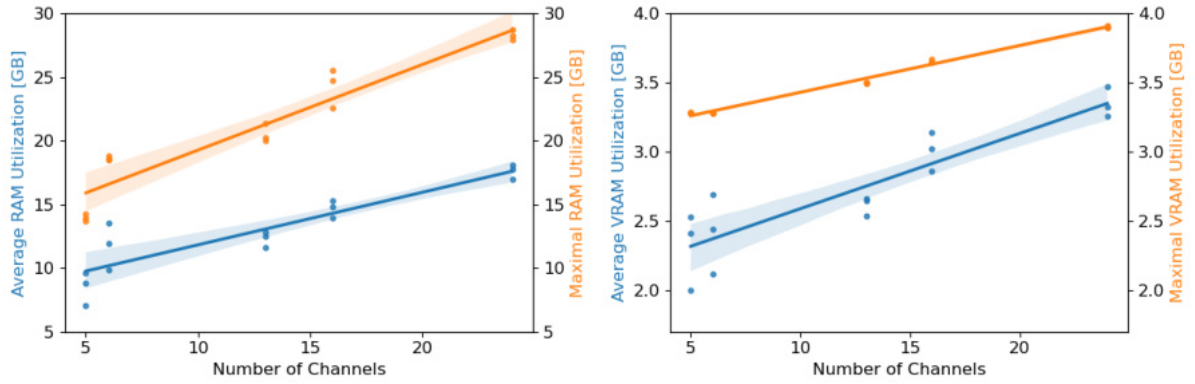
TABLE 4.7: Average and maximal utilization of RAM and VRAM, grouped and averaged per extension. The values are compared to the baseline. Ø denotes the average value, and Max the maximum value. The full table for all model runs can be found in table A.5.

Extension	RAM Ø [GB]	RAM Max [GB]	VRAM Ø [GB]	VRAM Max [GB]
Baseline	8.49	13.97	2.31	3.28
Extension 1	11.80 ↑	18.63 ↑	2.42 ↑	3.28
Extension 2	14.67 ↑	24.26 ↑	3.01 ↑	3.66 ↑
Extension 3	12.32 ↑	20.54 ↑	2.62 ↑	3.50 ↑
Extension 4	17.61 ↑	28.30 ↑	3.35 ↑	3.90 ↑
Extensions	14.10 ↑	23.43 ↑	2.85 ↑	3.59 ↑

In order to see if the assumed linear relationship exists here, a similar regression plot is used as well as for the other two components, shown in figure 4.10. It is based on the utilization values of all model runs as seen in table A.5 in appendix section A.5.5.

As expected, the regression lines show a rising regression line between feature space size and memory

utilization for both RAM and VRAM. However, contrary to the average values in table 4.7, the linear relationship is not as clear as expected, as the regression lines show a large variability. Especially in the average utilization values of both components, Extension 1 with 6 channels seems to break up the linearity, particularly in comparison to Extension 3 with 13 channels. Also, the step between the baseline and Extension 1 is large in both average and maximal RAM utilization, but opposing small in the VRAM utilization. However, when leaving out Extension 1, the values show a linear increase in utilization of RAM and VRAM with increasing complexity.



(A) Regression Plot of RAM Utilization.

(B) Regression Plot of VRAM Utilization.

FIGURE 4.10: Regression plots showing the relationship between number of channels (equates to feature space size and extension type) and average (blue) and maximal (orange) RAM (A) and VRAM (B) utilization.

The relationship of hardware utilization for the extensions of RAM and VRAM can be described as: the larger and more complex the feature space, the higher the RAM and VRAM utilization. This completely aligns with the assumption stated in section 3.6. Yet, because these two components, particularly the RAM, do not consume much energy even on full load, their impact on the resource consumption is negligible. As the VRAM is part of the GPU, a higher utilization would mean a higher load on the GPU, which, however, is not the case as seen above. Still, both components and their utilization show that there is a tendency towards a higher resource consumption with higher complexity, especially when combining these values with the runtime, number of epochs, and time per epoch metrics from table 4.2. The GPU and CPU follow a opposite pattern in their average utilization, but do not have a linear relationship in their maximal utilization, which is why a concrete statement about the relationship between complexity and hardware utilization cannot be made.

Chapter 5

Discussion

The study was built around proofing the concept of integrating spatial contextual information into a semantic segmentation model for LULC-M in order to enhance its performance and resource consumption. Hypothetically, by adding information about the spatial context of LULC classes using a road network, the model convergence should be supported and the overall resource consumption reduced alongside an increase of the performance, as the model can make more informed decisions about the LULC classes. For this purpose, select attributes of the OSM road network, road types and speed limits, were encoded and integrated in the feature space of the LULC Utility framework for LULC-M, which utilizes the SegFormer model for semantic segmentation. This resulted in four extensions, Extension 1 with a boolean road network without further encodings, Extension 2 with encoded road types, Extension 3 with encoded speed limit classes, and Extension 4 with a combination of Extensions 2 and 3. This section discusses the key findings and limitations of the study, implications for future research, as well as giving suggestions for improvement in order to enhance the generalizability and adaptability of the methodology.

5.1 Key Findings

The results show that the OSM road network with its selected attributes road types and speed limits indeed provides spatial contextual information about adjacent LULC classes. Integrating roads into the feature space of the model has an impact on the model's performance, resource consumption, and efficiency. In order to enlarge the impact of the road network on the model, the vanilla feature space of the LULC Utility was altered, reducing the satellite imagery based input data from nine bands to five, creating a baseline to compare the results to. The values of the IoU for the baseline are similar to the performance achieved on the ADE20K and COCO-Stuff benchmark with the SegFormer, shown in the original paper of the SegFormer by Xie et al. (2021). Although not directly comparable, this nevertheless means that even with a reduced feature space, the model still performs well, making the baseline a meaningful comparison basis.

All in all, by integrating road network data into the model, the performance is increased except for select cases, while GPU and CPU are utilized less, but over a longer runtime. When comparing the OA and IoU of the extensions to the baseline on the same amount of data, the extensions outperform the baseline in all cases, showing that the model can make more informed decisions about the LULC classes using the road network. Interestingly, the road geometries alone already increase the performance of the model. As the epochs needed for convergence are reduced in the extensions, they also help the model to converge earlier with less iterations through the data, therefore, the convergence

also is supported. However, as the time needed per epoch and runtime are increased considerably for all extensions, even when utilizing GPU and CPU less, the resource consumption was not improved, meaning that the integration of the road network attributes primarily increases the model's performance while using more resources. The trade-off has to be rated by the possible users and their priority when enhancing models, as the performance and resource consumption do not scale proportionally.

During the analysis of the results, it became clear that road types have generally a greater positive impact than speed limit classes, as the OA, IoU, and CA of the according Extension 2 is generally higher than Extension 3. The only exception is the LULC classes forest, where the speed limits achieved a higher CA. It should be noted that speed limit classes still improve the performance of the model, although usually to a lesser extent than road types. However, the combination of both road types and speed limits can have an even greater positive impact on the prediction of specific classes, as shown by Extension 4 for the LULC classes built-up and permanent crops and the OA and IoU scores for training and validation. But, only three of the six classes experienced an increase in performance in comparison to the other extensions, similar to Extension 2, emphasizing that most of the performance in Extension 4 derives from the road types. This aligns with the assumption stated above, that the speed limit classes will only provide a reduced amount of information, as the majority of the speed limits in the AOI is unknown. Still, the impact of the existing speed limits is clearly noticeable, as the performance of the model is increased in the majority of the cases also in Extension 3. Therefore, increasing the complexity and providing as much information as possible does not necessarily lead to better performance, but has the potential to do so if the information has an according density and distribution. However, this comes with the cost of increased resource consumption, as the model has to process more information, which is especially noticeable in the runtime of the model.

According optimization techniques could be applied to reduce the resource consumption, making the model more efficient and sustainable. In section 2.5, the training of DNNs was explained and the importance of optimization techniques emphasized. Optimization is one of the biggest challenges of DL, as it is necessary in every aspect of the training process. This project was done based on a preset model framework, the LULC Utility, without further adapting possible necessary optimizations for the adapted feature space, both for the baseline as well as for the extensions. Therefore, there is even more potential to be expected regarding the model's performance and efficiency with according tuning of the model's hyperparameters and optimization techniques. But, this has to be evaluated in separate studies, preferably by utilizing the vanilla LULC Utility with its full feature space, as this would show the real potential of the road network as spatial contextual information and enhancement of the model.

5.2 Road Attributes & LULC Classes

The impact on performance and resource consumption varies between the extensions and the LULC classes. The reason for this are the road network attributes utilized in this study themselves. The explorative relationship analysis showed that the tags of the selected attributes have a differentiated relationship towards adjacent LULC classes on the one hand. But, the relationship is not as clear as expected on the other hand, as it shows overlaps between features of the attributes in their relationship distribution. Only the minority of the features has a distinct relationship towards only one LULC

class, most are distributed over multiple classes. Additionally, finding significant relationships or defining thresholds which divide the values into different strengths of relations is difficult without further statistical analyses. But still, the information found in the evaluated relationships still enhance the performance of the model, therefore, for this study, the chosen methodology was suitable.

Furthermore, the results show that the chosen encoding method, OHE, was feasible as the information of the features was utilizable by the model. In OHE, the features of the road network are rasterized and encoded into boolean tensor channels, which expand the model's feature space. Other encoding methods, including ordinal or binary encoding (Potdar et al., 2017), could also be utilized if the data meets their requirements, leading to different results for the model, as the values are offered differently to the model to learn from.

Nevertheless, encoding all features of the road network includes many negligible ones, which expand the feature space unnecessary as they do not provide usable information to the model. Also, many features have such a low share in the road network, that including them in the feature space, even when they have a comparably distinct distribution, does not provide the model with a learnable concept or relationship that it can apply for inference, as these roads are barely present in other AOIs, for example London, as shown by Alghanim et al. (2021). This applies for both road types and speed limit classes. Therefore, to reduce the road network's complexity, feature engineering techniques were used. As the model evaluation shows, these were also feasible, as they effectively reduced the number of road types by half and categorized a variable number of different speed limits into 13 fixed categories.

As they are categorized into fixed ranges of speeds based on road classification systems, the categories of the speed limits could be used globally. But, the relationship analysis showed that the categories could be further reduced, as the distribution of the features is not distinct enough to justify the eight categories. This would reduce the complexity of the feature space, so that Extensions 3 and 4 could be more efficient while offering a similar level of information. Mentioned above, the class "unlimited" is only feasible in Germany and some other minor cases, and as the distribution is similar to "very_high+", these two could be merged. Also, the HCA showed that "walk" and "very_low" are very similar as well, so merging these too would not reduce the information content of the feature space. Furthermore, the categories are derived from road classification systems of different countries, summarized by Vitkienė et al. (2017). Using a different system could lead to different categories, which could be more distinct or more general, depending on the system and the LULC classes to predict. This could also be a way to reduce the complexity of the feature space, as the categories could be more distinct and therefore provide more information to the model. However, their feasibility would have to be evaluated in a separate study and the problem of a low mapping coverage of the speed limits still persists. In order to tackle this, merging multiple datasets with OSM, for example social media data or measured data from traffic monitoring stations as proposed by Camargo et al. (2020) and Ludwig et al. (2023), could be used to enhance the coverage of the speed limits. This would also enhance the generalizability of the model, as the speed limits would be more representative of the actual speed limits in the AOI, maybe evn enhancing the relationship to adjacent LULC classes due to their realistic behavior indirectly or directly based on the surrounding LULC areas.

As for the road types, the complexity of the features can be reduced further by omitting more road

types and by only keeping those with the strongest relation to LULC classes, like *residential* and *track*, which both enhance the prediction of built-up areas due to their occurrence and avoidance of these areas, whereby *track* additionally hints towards forests. Atwal et al. (2022) additionally merged the types *primary*, *secondary*, and *tertiary*, but these were not merged in this study due to their different importance in the road network. However, as their distribution over LULC classes is similar, they additionally could be merged to reduce the complexity of the feature space. Additionally, increasing the threshold of the global share also reduces the number of road types and helps in generalizing the procedure only to the most important types. The merge of the road types *motorway, trunk*, and their links could be seen as oversimplification, as especially the links have a different distribution than their counterparts. The background for this merge was the applied LCBB, whose impact on the model cannot be directly evaluated. As maximal values were used for the buffers, it is possible that road greenery next to roads or grass patches inside and around intersections, like shown in figure 3.4, were included in the buffers. This could have a negative impact on the model, as the model could learn to predict roads as grass or vice versa, reducing the accuracy of the grass class.

However, the CA of the grass class is already low in the baseline with an accuracy of only 41 %, so the LCBB cannot be the reason for the low performance of the model in the grass class. This probably is based on the reduced feature space and missing satellite imagery, which would be beneficial for the prediction of grass. When comparing the grass to the permanent crops class, which also has a low CA of 43 % in the baseline, there is a large difference in the extensions. The road network enhances the CA of the permanent crops class by 8 to 24 %p, while the performance for the grass class is reduced in all extensions up to 21 %p, although arguably the performance of Extension 2 is only 2 %p lower. Looking at the confusion matrices in appendix section A.5.2, it is visible that the grass class is confused primarily with the classes permanent crops and farmland, although in some cases forest and built-up are also predicted instead. Orchards, one of the tags of the class permanent crops, are often covered by grass, and farmlands also resemble grasslands in some of their growth states, so the confusion is partially understandable. Even adding vegetation indices as proposed by Tzepkenlis et al. (2023) would enhance this prediction only slightly, as the cover is indeed grass. The model predicts the cover correctly, but wrong for the evaluation metric, as the LULC class mapped in OSM is different. Given the additional low CA of permanent crops, it is advisable to reconsider the LULC classes in order to enhance the model's performance, as these two classes contribute to the lower OA compared to the baseline. Suitable LULC classes for natural areas could be, for example, forest, bare land, and vegetation, as proposed by Basheer et al. (2022). This would enhance the model's performance, as the classes are more distinct and the model can learn more easily to differentiate between them. The impact of the road network would have to be evaluated separately, as the model has to learn new relationships between the road network and the LULC classes.

In that regard, it is to be expected that for different AOIs, different shares of the features are present, questioning the generalizability of the model. This is especially true for the road types, as the global distribution of road types is not the same as in the AOI of the study. Furthermore, the chosen AOI has a characteristic distribution of LULC classes and therefore a special relationship between roads and LULC classes. The characteristic border between the Odenwald and the Upper Rhine Rift Valley in the MA-HD-RNK region provokes a distinct road architecture, avoiding building paved roads in the

forested areas with rougher relief and connecting the built-up areas in the valley with more major road types with higher speed limits. But still, as multiple regions with different characteristic distributions were merged into this AOI, the overall distribution is representative at least for Germany. All these factors influence the relationship between road types and LULC classes. Still, orienting on the global share of the road types is a good basis for reducing the complexity of the according road network to a feasible and utilizable amount.

5.3 Methodology

As this study is a proof-of-concept of integrating spatial context into semantic segmentation models to improve the model's performance and efficiency, the concrete relationships between roads and LULC classes are only secondary to the overall goal of the study. Nevertheless, the results show tendencies, which are rediscovered analogous in the model evaluation, and by using the road lengths rather than the number of roads, the chosen methodology was generally feasible and sufficient for the task at hand.

The relationships could be explored further in future studies, as the relationships are not as clear as expected and the study only takes into account the relative intersection of roads with adjacent LULC classes. The methodology proposed by Atwal et al. (2022) seems more suitable for a more elaborate analysis of the relationship between road networks and LULC classes, as it includes a vectorized approach to assess the spatial relationship between the two. These vectors could also be encoded into the feature space of the model to give the model a more detailed understanding of the relationship between roads and LULC classes. This would also enhance the generalizability of the model, as the vectors could be applied to different AOIs and the model could learn the relationship between roads and LULC classes in a more general way. However, to achieve this, the model has to be trained on a much larger dataset, preferably with globally distributed AOIs, in order to quantify the relationship realistically and generalized. This would also help in tackling the loss behavior of the models, shown in figure 4.8, where another reason for the lower validation than training loss could be noisy input data and not enough data for a representative validation, which could both be fixed by a larger dataset.

The clustering method to find similarities between the patterns of the explorative relationship analysis in order to reduce the complexity of the road network is based on the HCA, paired with elaborate feature engineering. This requires domain knowledge and many explicit preprocessing steps. Although suitable for this study, others like k-means and DBSCAN (Montazeri et al., 2021), maximum likelihood exploratory factor analysis (Petrakis et al., 2021), or the principal component analysis (Dharani & Sreenivasulu, 2019) are also proposed as suitable algorithms for effective automated dimensionality reduction. Utilizing them has the potential to offer other insights on the relationships between road attributes and LULC classes, enhancing the explorative relationship analysis. However, as OSM comes with drawbacks like completeness and correctness, feature engineering is required nonetheless to ensure a suitable quality level and a successful utilization of the data. Also, researchers should keep in mind that some information about the LULC of the area could be lost if road types are merged together with a not strong enough similarity in their distribution over LULC classes. The according task determines the level of detail needed and the most suitable approach.

In that regard, the applied automated feature engineering steps for the road network, including disregarding and merging features as well as filling missing values, are based on domain knowledge, literature, and the OSM Wiki. However, as OSM has many anomalies due to wrong mapping, locally restricted information and regional differences (Ludwig, Fendrich, & Zipf, 2021), or even vandalism (Neis, Goetz, & Zipf, 2012; Vargas-Munoz et al., 2021), the information of these is lost during this feature engineering procedure, as only few anomalies are included. Although many tools exist to assess the quality of the volunteered OSM (Minghini & Frassinelli, 2019; Moradi et al., 2021; Schott et al., 2024), researches have to still "trust" the data of OSM to some extent. It is advised to base the feature engineering procedure on the OSM Wiki, where suggested tags and values are summarized, as including every anomaly would require a highly detailed data analysis and a similarly complex feature engineering process.

As for the data itself, the chosen attributes road types and speed limits were selected based on literature and similar studies. Other attributes could be utilized for such a project as well, primarily speed restrictions in specific timeframes or access restrictions. These could also hint towards adjacent LULC, for example residential areas when a speed limit is applied only during night time hours or industrial areas where only specific vehicles are allowed. The problem here is the mapping completeness, which is even lower than for the tag *maxspeed* and the speed tags, as well as the low number of occurrences altogether according to OSM Taginfo. Based on the list of global feature shares and their combinations in OSM Taginfo (2024), another idea is to utilize road surface types (tag *surface*), because they also could hint towards specific LULC classes. For example, a cobblestone road could be an indicator for a historic city center, while a gravel road could be an indicator for a rural area. The mapping completeness here (25.61 %) is much higher than for *maxspeed* (7.40 %) or *access* (6.20 %), so this could be a possible attribute to utilize. The problem here, on the other hand, is that the distribution of values is more biased, as 44.89 % of the tag *surface* has the value *asphalt*. The question here, of course, is also if there are similar or even stronger distinct signals between the different features to adjacent LULC areas, which is doubtful. A combination of multiple tags can, therefore, be a solution for the bias. However, that would have to be evaluated in a separate study especially in regard to the resource consumption of the model due to the possibly large feature space. Nevertheless, as seen in the evaluation of the extension, the best attribute of the road network to utilize is the road type, as it has the most positive influence on the model's performance and the highest mapping completeness globally, which is further increasing, as shown by Anderson et al. (2019), Barrington-Leigh and Millard-Ball (2017), and Herfort et al. (2021).

5.4 Objective & Data Redundancy

As mentioned above, this study functions as a proof-of-concept that integrating spatial contextual information enhances LULC-M. The example of using road networks as according information was chosen based on the idea that they are crucial parts of the landscape and constructed in mind of the adjacent LULC areas. But, using roads to classify the adjacent LULC areas would be redundant, as LULC areas are used for constructing the roads and the roads are then used to classify the LULC areas. In other words, it would be a model leakage where the same information is recycled and used again. However, this is based on the assumption that the mapping completeness of the OSM data is sufficient in the according AOIs, regarding both the LULC areas as well as the roads. Also, it is based on the assumption

that both are correctly mapped, which sometimes is not the case, as OSM includes many anomalies and is often based on volunteered action, possible with missing domain knowledge. Barrington-Leigh and Millard-Ball (2017) state that the road network, which is the main emphasis of OSM (hence the name), usually has a higher mapping completeness than other features, including LULC. Also, there is a large bias in mapping completeness of additional features, which concentrate on very densely or sparsely populated areas, as shown by Herfort et al. (2021). Furthermore, utilizing OSM data always comes with additional drawbacks, including currentness and correctness, which applies also to LULC areas.

Yet, as the main task of ML models is to utilize learned concepts and models for inference on unknown data (Sarker, 2021; Shinde & Shah, 2018), completeness is the drawback that has to be tackled the most. In order to tackle the completeness of roads and their attributes in specific AOIs, merging data from different sources could be a solution, for example authoritative datasets or the Microsoft Roads dataset, as proposed by Anderson et al. (2019), S. Srivastava et al. (2019), and Vargas-Munoz et al. (2021). Moreover, multiple studies introduced ML based road type classification algorithms (Alghanim et al., 2021; Y. Zhao et al., 2023) using remote sensing and street-view imagery, which, in combination with road detection algorithms (Abdollahi et al., 2020; Z. Chen et al., 2022), could provide a fully automated "road detection to LULC-M" approach. Additionally, average speeds could also be estimated using DL approaches, as proposed by Keller et al. (2020), further enriching or filling missing data. These methods could tackle the problem of applying this methodology to areas with a low mapping completeness, as roads could be extracted and classified by utilizing DL approaches. Both could then be used to enhance the prediction of surrounding LULC areas, which in turn could function as a verification for the classified road types. In such cases, utilizing road networks for LULC-M can provide a real case scenario where LULC-M of unknown areas can be enhanced with DL generated roads. However, such an elaborated approach has to be evaluated in future studies, as the complexity of the model and the resource consumption could increase considerably, providing a bad trade-off between efficiency and performance. In that regard, other model architectures than the SegFormer could also be used for this task, as the SegFormer is not the only model suitable for semantic segmentation. Tzepkenlis et al. (2023), for example, propose a U-Net with a temporal attention encoder, which outperforms the SegFormer in terms of performance and efficiency. In order to measure the efficiency of the according model effectively, Desislavov et al. (2023) and Getzner et al. (2023) propose using Floating Point Operations as a measure of computational complexity, independent from machine configuration. Mehlin et al. (2023) further propose multiple other metrics to measure carbon emission, energy usage, and computational complexity, in order to enhance the evaluation of the model's efficiency and sustainability, which should further be considered in future studies.

All in all, the methodology was suitable for the study and the results show that the integration of spatial contextual information into a semantic segmentation model for LULC-M is feasible. It enhances the model's performance and offers possible improvements to resource consumption and efficiency, if according optimization techniques are implemented. Therefore, the feasibility of the concept is proven.

Chapter 6

Conclusion

This study aimed at evaluating the integration of the OSM road network and two of its attributes, road types and speed limits, into the feature space of a semantic segmentation model for LULC-M. Hypothetically, by adding information about the spatial context of LULC classes using a road network, the model convergence should be supported and the overall resource consumption reduced alongside an increase of the performance, as the model can make more informed decisions about the LULC classes. The feature engineered attributes were encoded via OHE into the feature space of the model. The results revealed that this integration provides valuable spatial contextual information about adjacent LULC classes, primarily increasing the model's performance while needing less epochs for convergence. Therefore, the model can make more informed decisions about the prediction of LULC areas, partially confirming the hypothesis. In four out of six LULC classes and five out of six metric comparison cases, the extensions outperform the baseline. This improvement, however, comes at the cost of increased resource consumption and longer runtimes, highlighting a trade-off between enhanced model performance and resource efficiency. Still, the results showed that the model could potentially achieve similar performance with a higher efficiency and less resource consumption, if it is optimized accordingly. The study also highlighted that increasing the complexity of the model by adding more information does not guarantee better performance unless the information is optimized. This finding emphasizes the need for careful consideration of the information included in the model to balance performance gains and resource consumption. Future research should focus on optimizing the model to reduce resource consumption while maintaining or enhancing performance. Applying advanced optimization techniques and fine-tuning hyperparameters are expected to improve the model's efficiency and sustainability and keep the level of increased performance, if not even increased. Evaluating these optimizations with the full feature space of the vanilla model could further reveal the true potential of road network data as an enhancement for LULC-M models.

The proof-of-concept that this study aimed for was achieved, but the results are not generalizable due to the small dataset and the specific setup. The study should be repeated with a larger dataset, the omission of the LCBB, a further reduction of road network attributes to only tags with substantial relations to specific LULC classes, adapted LULC filters or other LULC classes, fine-tuned hyperparameters for the extended feature space, and according optimization techniques. This would allow for a more comprehensive evaluation of the integration of road network data into LULC-M models and provide a more reliable basis for future research in this field. However, it is expected that the tendencies regarding an increase in performance and a possible reduction of resource consumption will remain similar, as the results of this study are in line with previous research on the integration of contextual data into models for LULC-M. Also, the methodology is regarded feasible as it is based on established practices

and tools as well as prior studies, and the results are reproducible. As other studies have shown, OSM data is suitable for training and enhancing DL models for various tasks, including the integration of spatial contextual information for effective and efficient LULC-M. But, the data has to be preprocessed and feature engineered carefully to avoid anomalies and irrelevant information that could decrease the model's performance.

Alongside multiple other studies, this study has proven that enriching DL models with contextual data has the potential to increase the performance while making them more efficient and sustainable. In regard to climate change and the SDGs, this is a crucial step towards reducing their negative environmental impacts and utilizing their full potential sustainably. For future research, this should be the main aim, to develop sustainable DL models that can be applied to various tasks and domains, as they have major potential for human development in any regard. This study has shown that the integration of road network data into LULC-M models is a promising approach to achieve this goal in one small field of the large landscape of applications of DL.

Bibliography

Abdollahi, A., Pradhan, B., Shukla, N., Chakraborty, S., & Alamri, A. (2020). Deep Learning Approaches Applied to Remote Sensing Datasets for Road Extraction: A State-Of-The-Art Review. *Remote Sensing*, *12*(9), 1444. https://doi.org/10.3390/rs12091444

Abdullah, M., Madain, A., & Jararweh, Y. (2022). ChatGPT: Fundamentals, Applications and Social Impacts. *2022 Ninth International Conference on Social Networks Analysis, Management and Security (SNAMS)*, 1–8. https://doi.org/10.1109/SNAMS58071.2022.10062688

Ahmadzai, F. (2020). Analyses and modeling of urban land use and road network interactions using spatial-based disaggregate accessibility to land use. *Journal of Urban Management*, *9*(3), 298–315. https://doi.org/10.1016/j.jum.2020.06.003

Airbus. (2022). *Copernicus DEM* (Product Handbook No. GEO.2018-1988-2). Airbus Defence and Space GmbH. Retrieved June 16, 2024, from https://spacedata.copernicus.eu/documents/20123/122407/GEO1988-CopernicusDEM-SPE-002_ProductHandbook_I5.0+%281%29.pdf

Alexander, M., & Kusleika, D. (2022). *Microsoft Excel 365 bible*. John Wiley & Sons, Inc.

Alghanim, A., Jilani, M., Bertolotto, M., & McArdle, G. (2021). Leveraging Road Characteristics and Contributor Behaviour for Assessing Road Type Quality in OSM. *ISPRS International Journal of Geo-Information*, *10*(7), 436. https://doi.org/10.3390/ijgi10070436

Alhassan, V., Henry, C., Ramanna, S., & Storie, C. (2020). A deep learning framework for land-use/land-cover mapping and analysis using multispectral satellite imagery. *Neural Computing and Applications*, *32*(12), 8529–8544. https://doi.org/10.1007/s00521-019-04349-9

Alhichri, H., Alswayed, A. S., Bazi, Y., Ammour, N., & Alajlan, N. A. (2021). Classification of Remote Sensing Images Using EfficientNet-B3 CNN Model With Attention. *IEEE Access*, *9*, 14078–14094. https://doi.org/10.1109/ACCESS.2021.3051085

Al-Taei, A. I., Alesheikh, A. A., & Darvishi Boloorani, A. (2023). Land Use/Land Cover Change Analysis Using Multi-Temporal Remote Sensing Data: A Case Study of Tigris and Euphrates Rivers Basin. *Land*, *12*(5), 1101. https://doi.org/10.3390/land12051101

Alzubaidi, L., Zhang, J., Humaidi, A., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M., Al-Amidie, M., & Farhan, L. (2021). Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*, *8*. https://doi.org/10.1186/s40537-021-00444-8

Anderson, J., Sarkar, D., & Palen, L. (2019). Corporate Editors in the Evolving Landscape of OpenStreetMap. *ISPRS International Journal of Geo-Information*, *8*(5), 232. https://doi.org/10.3390/ijgi8050232

Atwal, K. S., Anderson, T., Pfoser, D., & Züfle, A. (2022). Predicting building types using OpenStreetMap. *Scientific Reports*, *12*(1), 19976. https://doi.org/10.1038/s41598-022-24263-w

Auer, M., Eckle, M., & Fendrich, S. (2018). Ohsome – eine plattform zur analyse raumzeitlicher entwicklungen von openstreetmap-daten für intrinsische qualitätsbewertungen. *Journal für Angewandte Geoinformatik*, *4*. https://doi.org/10.14627/537647020

Awuh, M. E., Japhets, P. O., Officha, M. C., Okolie, A. O., & Enete, I. C. (2019). A Correlation Analysis of the Relationship between Land Use and Land Cover/Land Surface Temperature in Abuja Municipal, FCT, Nigeria. *Journal of Geographic Information System*, *11*(01), 44. https://doi.org/10.4236/jgis.2019.111004

Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). *Layer Normalization*. Retrieved July 2, 2024, from http://arxiv.org/abs/1607.06450

Badrinarayanan, V., Kendall, A., & Cipolla, R. (2017). SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(12), 2481–2495. https://doi.org/10.1109/TPAMI.2016.2644615

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural Machine Translation by Jointly Learning to Align and Translate. *3rd International Conference for Learning Representations (ICLR 2015)*. https://doi.org/10.48550/arXiv.1409.0473

Barrington-Leigh, C., & Millard-Ball, A. (2017). The world's user-generated road map is more than 80% complete. *PLOS ONE*, *12*(8), e0180698. https://doi.org/10.1371/journal.pone.0180698

Basheer, S., Wang, X., Farooque, A. A., Nawaz, R. A., Liu, K., Adekanmbi, T., & Liu, S. (2022). Comparison of Land Use Land Cover Classifiers Using Different Satellite Imagery and Machine Learning Techniques. *Remote Sensing*, *14*(19), 4978. https://doi.org/10.3390/rs14194978

Bernard, E. (2021). *Introduction to Machine Learning*. Wolfram Media, Inc.

Bi, J., Zhu, Z., & Meng, Q. (2021). Transformer in Computer Vision. *2021 IEEE International Conference on Computer Science, Electronic Information Engineering and Intelligent Control Technology (CEI)*, 178–188. https://doi.org/10.1109/CEI52496.2021.9574462

Bini, S. A. (2018). Artificial Intelligence, Machine Learning, Deep Learning, and Cognitive Computing: What Do These Terms Mean and How Will They Impact Health Care? *The Journal of Arthroplasty*, *33*(8), 2358–2361. https://doi.org/10.1016/j.arth.2018.02.067

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Bjorck, N., Gomes, C. P., Selman, B., & Weinberger, K. Q. (2018). Understanding Batch Normalization. *31st Conference on Neural Information Processing Systems (NeurIPS 2018)*, *31*.

Boston, T., Van Dijk, A., Larraondo, P. R., & Thackway, R. (2022). Comparing CNNs and Random Forests for Landsat Image Segmentation Trained on a Large Proxy Land Cover Dataset. *Remote Sensing*, *14*(14), 3396. https://doi.org/10.3390/rs14143396

Brauwers, G., & Frasincar, F. (2023). A General Survey on Attention Mechanisms in Deep Learning. *IEEE Transactions on Knowledge and Data Engineering*, *35*(4), 3279–3298. https://doi.org/10.1109/TKDE.2021.3126456

Camargo, C. Q., Bright, J., McNeill, G., Raman, S., & Hale, S. A. (2020). Estimating Traffic Disruption Patterns with Volunteered Geographic Information. *Scientific Reports*, *10*(1), 1271. https://doi.org/10.1038/s41598-020-57882-2

Cao, G. (2022). Deep Learning of Big Geospatial Data: Challenges and Opportunities. In B. Li, X. Shi, A.-X. Zhu, C. Wang, & H. Lin (Eds.), *New Thinking in GIScience* (pp. 159–169). Springer Nature. https://doi.org/10.1007/978-981-19-3816-0_18

Cao, R., Zhu, J., Tu, W., Li, Q., Cao, J., Liu, B., Zhang, Q., & Qiu, G. (2018). Integrating Aerial and Street View Images for Urban Land Use Classification. *Remote Sensing*, *10*(10), 1553. https://doi.org/10.3390/rs10101553

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., & Yuille, A. L. (2015). Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFS. *3rd International Conference for Learning Representations (ICLR 2015)*. https://doi.org/10.48550/arXiv.1412.7062

Chen, L.-C., Yang, Y., Wang, J., Xu, W., & Yuille, A. L. (2016). Attention to Scale: Scale-aware Semantic Image Segmentation. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016)*. https://doi.org/10.48550/arXiv.1511.03339

Chen, X., Zhou, W., Pickett, S. T. A., Li, W., & Han, L. (2016). Spatial-Temporal Variations of Water Quality and Its Relationship to Land Use and Land Cover in Beijing, China. *International Journal of Environmental Research and Public Health*, *13*(5), 449. https://doi.org/10.3390/ijerph13050449

Chen, Y., Liu, P., Zhao, J., Huang, K., & Yan, Q. (2023). Shallow-Guided Transformer for Semantic Segmentation of Hyperspectral Remote Sensing Imagery. *Remote Sensing*, *15*(13), 3366. https://doi.org/10.3390/rs15133366

Chen, Z., Deng, L., Luo, Y., Li, D., Marcato Junior, J., Nunes Gonçalves, W., Awal Md Nurunnabi, A., Li, J., Wang, C., & Li, D. (2022). Road extraction in remote sensing data: A survey. *International Journal of Applied Earth Observation and Geoinformation*, *112*, 102833. https://doi.org/10.1016/j.jag.2022.102833

Chicco, D., & Jurman, G. (2020). The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC Genomics*, *21*(1), 6. https://doi.org/10.1186/s12864-019-6413-7

City of Heidelberg. (2024a). *Die stadt*. Retrieved June 4, 2024, from https://heidelberg.de/HD/Leben/Die+Stadt.html

City of Heidelberg. (2024b). *Heidelberg in zahlen*. Retrieved June 4, 2024, from https://heidelberg.de/HD/Rathaus/Heidelberg+in+Zahlen.html

City of Mannheim. (2024). *Mannheim in zahlen*. Retrieved June 4, 2024, from https://www.mannheim.de/de/wirtschaft-entwickeln/wirtschaftsstandort/mannheim-in-zahlen

Comber, A., & Wulder, M. (2019). Considering spatiotemporal processes in big data analysis: Insights from remote sensing of land cover and land use. *Transactions in GIS*, *23*(5), 879–891. https://doi.org/10.1111/tgis.12559

Cordonnier, J.-B., Loukas, A., & Jaggi, M. (2020). On the Relationship between Self-Attention and Convolutional Layers. *International Conference on Learning Representations*. https://doi.org/10.48550/arXiv.1911.03584

Corteva Agriscience. (2024). *GeoCube*. Retrieved June 22, 2024, from https://corteva.github.io/geocube/stable/

Courtial, A., Touya, G., & Zhang, X. (2022). Representing Vector Geographic Information As a Tensor for Deep Learning Based Map Generalisation. *AGILE: GIScience Series*, *3*, 1–8. https://doi.org/10.5194/agile-giss-3-32-2022

Csurka, G., & Larlus, D. (2013). What is a good evaluation measure for semantic segmentation? *IEEE Trans. Pattern Anal. Mach. Intell.*, *26*. https://doi.org/10.5244/C.27.32

Demir, D. B., & Musaoglu, N. (2023). Automatic Classification of Selected Corine Classes Using Deep Learning Based Semantic Segmentation. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *XLVIII-M-3-2023*, 71–75. https://doi.org/10.5194/isprs-archives-XLVIII-M-3-2023-71-2023

de Santana Correia, A., & Colombini, E. L. (2022). Attention, please! A survey of neural attention models in deep learning. *Artificial Intelligence Review*, *55*(8), 6037–6124. https://doi.org/10.1007/s10462-022-10148-x

Desislavov, R., Martínez-Plumed, F., & Hernández-Orallo, J. (2023). Compute and Energy Consumption Trends in Deep Learning Inference. *Sustainable Computing: Informatics and Systems*, *38*, 100857. https://doi.org/10.1016/j.suscom.2023.100857

Dharani, M., & Sreenivasulu, G. (2019). Land use and land cover change detection by using principal component analysis and morphological operations in remote sensing applications. *International Journal of Computers and Applications*. https://doi.org/10.1080/1206212X.2019.1578068

Diakogiannis, F. I., Waldner, F., Caccetta, P., & Wu, C. (2020). ResUNet-a: A deep learning framework for semantic segmentation of remotely sensed data. *ISPRS Journal of Photogrammetry and Remote Sensing*, *162*, 94–114. https://doi.org/10.1016/j.isprsjprs.2020.01.013

Digra, M., Dhir, R., & Sharma, N. (2022). Land use land cover classification of remote sensing images based on the deep learning approaches: A statistical analysis and review. *Arabian Journal of Geosciences*, *15*(10), 1003. https://doi.org/10.1007/s12517-022-10246-8

Ding, H., Jiang, X., Shuai, B., Liu, A. Q., & Wang, G. (2020). Semantic Segmentation With Context Encoding and Multi-Path Decoding. *IEEE Transactions on Image Processing*, *29*, 3520–3533. https://doi.org/10.1109/TIP.2019.2962685

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2020). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *International Conference on Learning Representations*. https://doi.org/10.48550/arXiv.2010.11929

Dumoulin, V., & Visin, F. (2018). *A guide to convolution arithmetic for deep learning*. https://doi.org/10.48550/arXiv.1603.07285

ESA. (2024). *S2 Mission*. Retrieved June 16, 2024, from https://sentiwiki.copernicus.eu/web/s2-mission

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, *27*(8), 861–874. https://doi.org/10.1016/j.patrec.2005.10.010

Feng, X., & Myint, S. W. (2016). Exploring the effect of neighboring land cover pattern on land surface temperature of central building objects. *Building and Environment*, *95*, 346–354. https://doi.org/10.1016/j.buildenv.2015.09.019

Forget, Y., Linard, C., & Gilbert, M. (2018). Supervised Classification of Built-Up Areas in Sub-Saharan African Cities Using Landsat Imagery and OpenStreetMap. *Remote Sensing*, *10*(7), 1145. https://doi.org/10.3390/rs10071145

Getzner, J., Charpentier, B., & Günnemann, S. (2023). Accuracy is not the only Metric that matters: Estimating the Energy Consumption of Deep Learning Models. *International Conference on Learning Representations*. https://doi.org/10.48550/arXiv.2304.00897

Geva, M., Schuster, R., Berant, J., & Levy, O. (2021). Transformer Feed-Forward Layers Are Key-Value Memories. In M.-F. Moens, X. Huang, L. Specia, & S. W.-t. Yih (Eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing* (pp. 5484–5495). Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.emnlp-main.446

Gholamalinezhad, H., & Khosravi, H. (2020). *Pooling Methods in Deep Neural Networks, a Review*. https://doi.org/10.48550/arXiv.2009.07485

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., & He, K. (2018). *Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour*. https://doi.org/10.48550/arXiv.1706.02677

Guo, Y., Liu, Y., Georgiou, T., & Lew, M. S. (2018). A review of semantic segmentation using deep neural networks. *International Journal of Multimedia Information Retrieval*, *7*(2), 87–93. https://doi.org/10.1007/s13735-017-0141-z

Hacar, M., Kılıç, B., & Şahbaz, K. (2018). Analyzing OpenStreetMap Road Data and Characterizing the Behavior of Contributors in Ankara, Turkey. *ISPRS International Journal of Geo-Information*, *7*(10), 400. https://doi.org/10.3390/ijgi7100400

Hamilton, M., Zhang, Z., Hariharan, B., Snavely, N., & Freeman, W. T. (2022). Unsupervised Semantic Segmentation by Distilling Feature Correspondences. *International Conference on Learning Representations*. https://doi.org/10.48550/arXiv.2203.08414

Han, K., Wang, Y., Chen, H., Chen, X., Guo, J., Liu, Z., Tang, Y., Xiao, A., Xu, C., Xu, Y., Yang, Z., Zhang, Y., & Tao, D. (2023). A Survey on Vision Transformer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *45*(1), 87–110. https://doi.org/10.1109/TPAMI.2022.3152247

Hancock, J. T., & Khoshgoftaar, T. M. (2020). Survey on categorical data for neural networks. *Journal of Big Data*, *7*(1), 28. https://doi.org/10.1186/s40537-020-00305-w

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016)*, 770–778. https://doi.org/10.1109/CVPR.2016.90

HeiGIT. (2024a). *Heidelberg Institute for Geoinformation Technology*. Retrieved June 17, 2024, from https://heigit.org/

HeiGIT. (2024b). *LULC Utility*. Retrieved May 25, 2024, from https://gitlab.gistools.geog.uni-heidelberg.de/climate-action/utilities/lulc-utility

HeiGIT. (2024c). *Ohsome Quality API*. Retrieved June 10, 2024, from https://heigit.org/big-spatial-data-analytics-en/ohsome/ohsome-quality-analyst-oqt/

Herfort, B., Lautenbach, S., Porto de Albuquerque, J., Anderson, J., & Zipf, A. (2021). The evolution of humanitarian mapping within the OpenStreetMap community. *Scientific Reports*, *11*(1), 3037. https://doi.org/10.1038/s41598-021-82404-z

HOT OSM. (2024). *Humanitarian OpenStreetMap Team*. Retrieved June 17, 2024, from https://www.hotosm.org/

Hu, H., Li, Z., Li, L., Yang, H., & Zhu, H. (2020). Classification of Very High-Resolution Remote Sensing Imagery Using a Fully Convolutional Network With Global and Local Context Information Enhancements. *IEEE Access*, *8*, 14606–14619. https://doi.org/10.1109/ACCESS.2020.2964760

Hugging Face. (2024). *SegFormer*. Retrieved January 30, 2024, from https://huggingface.co/docs/transformers/model_doc/segformer

IPCC. (2023). *Climate Change 2023: Synthesis Report. Contribution of Working Groups I, II and III to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change [Core Writing Team, H. Lee and J. Romero (eds.)]. IPCC, Geneva, Switzerland.* Intergovernmental Panel on Climate Change (IPCC). https://doi.org/10.59327/IPCC/AR6-9789291691647

Janocha, K., & Czarnecki, W. M. (2017). On Loss Functions for Deep Neural Networks in Classification. *Theoretical Foundations of Machine Learning 2017 (TFML 2017)*. https://doi.org/10.48550/arXiv.1702.05659

Johnson, S. C. (1967). Hierarchical clustering schemes. *Psychometrika*, *32*(3), 241–254. https://doi.org/10.1007/BF02289588

Jokar Arsanjani, J., Mooney, P., Zipf, A., & Schauss, A. (2015). Quality Assessment of the Contributed Land Use Information from OpenStreetMap Versus Authoritative Datasets. In J. Jokar Arsanjani, A. Zipf, P. Mooney, & M. Helbich (Eds.), *OpenStreetMap in GIScience: Experiences, Research, and Applications* (pp. 37–58). Springer International Publishing. https://doi.org/10.1007/978-3-319-14280-7_3

Kar, A. K., Choudhary, S. K., & Singh, V. K. (2022). How can artificial intelligence impact sustainability: A systematic literature review. *Journal of Cleaner Production*, *376*, 134120. https://doi.org/10.1016/j.jclepro.2022.134120

Keller, S., Gabriel, R., & Guth, J. (2020). Machine Learning Framework for the Estimation of Average Speed in Rural Road Networks with OpenStreetMap Data. *ISPRS International Journal of Geo-Information*, *9*(11), 638. https://doi.org/10.3390/ijgi9110638

Khamchiangta, D., & Dhakal, S. (2020). Time series analysis of land use and land cover changes related to urban heat island intensity: Case of Bangkok Metropolitan Area in Thailand. *Journal of Urban Management*, *9*(4), 383–395. https://doi.org/10.1016/j.jum.2020.09.001

Khan, S., Naseer, M., Hayat, M., Zamir, S. W., Khan, F. S., & Shah, M. (2021). Transformers in Vision: A Survey. *ACM Computing Surveys*, *54*, 1–41. https://doi.org/10.1145/3505244

Kibena, J., Nhapi, I., & Gumindoga, W. (2014). Assessing the relationship between water quality parameters and changes in landuse patterns in the Upper Manyame River, Zimbabwe. *Physics and Chemistry of the Earth, Parts A/B/C*, *67–69*, 153–163. https://doi.org/10.1016/j.pce.2013.09.017

Kingma, D. P., & Ba, J. (2015). Adam: A Method for Stochastic Optimization. *3rd International Conference for Learning Representations (ICLR 2015)*. https://doi.org/10.48550/arXiv.1412.6980

Kirillov, A., He, K., Girshick, R., Rother, C., & Dollár, P. (2019). Panoptic Segmentation. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2019)*. https://doi.org/10.48550/arXiv.1801.00868

Kussul, N., Lavreniuk, M., Skakun, S., & Shelestov, A. (2017). Deep Learning Classification of Land Cover and Crop Types Using Remote Sensing Data. *IEEE Geoscience and Remote Sensing Letters*, *14*(5), 778–782. https://doi.org/10.1109/LGRS.2017.2681128

Lazzaro, D., Cinà, A. E., Pintor, M., Demontis, A., Biggio, B., Roli, F., & Pelillo, M. (2023). Minimizing Energy Consumption of Deep Learning Models by Energy-Aware Training. *22nd International Conference on Image Analysis and Processing (ICIAP 2023)*. https://doi.org/10.48550/arXiv.2307.00368

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

LeCun, Y., Kavukcuoglu, K., & Farabet, C. (2010). Convolutional networks and applications in vision. *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, 253–256. https://doi.org/10.1109/ISCAS.2010.5537907

LeCun, Y. A., Bottou, L., Orr, G. B., & Müller, K.-R. (2012). Efficient BackProp. In G. Montavon, G. B. Orr, & K.-R. Müller (Eds.), *Neural Networks: Tricks of the Trade: Second Edition* (pp. 9–48). Springer. https://doi.org/10.1007/978-3-642-35289-8_3

Levinson, D. M., Xie, F., & Zhu, S. (2007). The Co-Evolution of Land Use and Road Networks. In *Transportation and Traffic Theory* (pp. 839–859). Emerald Group.

Levinson, D. M., & Yerra, B. (2005). *How Land Use Shapes the Evolution of Road Networks*. https://doi.org/10.2139/ssrn.1736160

Li, B., Jiang, X., Bai, D., Zhang, Y., Zheng, N., Dong, X., Liu, L., Yang, Y., & Li, D. (2021). Full-Cycle Energy Consumption Benchmark for Low-Carbon Computer Vision. *35th Conference on Neural Information Processing Systems (NeurIPS 2021)*. https://doi.org/10.48550/arXiv.2108.13465

Li, J., Cai, Y., Li, Q., Kou, M., & Zhang, T. (2024). A review of remote sensing image segmentation by deep learning methods. *International Journal of Digital Earth*, *17*(1), 2328827. https://doi.org/10.1080/17538947.2024.2328827

Lightning AI. (2024). *PyTorch Lightning*. Retrieved June 25, 2024, from https://lightning.ai/pytorch-lightning

Lin, X., Cheng, Y., Chen, G., Chen, W., Chen, R., Gao, D., Zhang, Y., & Wu, Y. (2023). Semantic Segmentation of China's Coastal Wetlands Based on Sentinel-2 and Segformer. *Remote Sensing*, *15*(15), 3714. https://doi.org/10.3390/rs15153714

Liu, Y., Zhang, Y., Wang, Y., Hou, F., Yuan, J., Tian, J., Zhang, Y., Shi, Z., Fan, J., & He, Z. (2024). A Survey of Visual Transformers. *IEEE Transactions on Neural Networks and Learning Systems*, *35*(6), 7478–7498. https://doi.org/10.1109/TNNLS.2022.3227717

Liu, Y., Yu, J., & Han, Y. (2018). Understanding the effective receptive field in semantic image segmentation. *Multimedia Tools and Applications*, *77*(17), 22159–22171. https://doi.org/10.1007/s11042-018-5704-3

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., & Guo, B. (2021). Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 9992–10002. https://doi.org/10.1109/ICCV48922.2021.00986

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2015)*, 3431–3440. https://doi.org/10.1109/CVPR.2015.7298965

Ludwig, C., Psotta, J., Buch, A., Kolaxidis, N., Fendrich, S., Zia, M., Fürle, J., Rousell, A., & Zipf, A. (2023). Traffic Speed Modelling to Improve Travel Time Estimation in OpenRouteService. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *XLVIII-4-W7-2023*, 109–116. https://doi.org/10.5194/isprs-archives-XLVIII-4-W7-2023-109-2023

Ludwig, C., Fendrich, S., & Zipf, A. (2021). Regional variations of context-based association rules in OpenStreetMap. *Transactions in GIS*, *25*(2), 602–621. https://doi.org/10.1111/tgis.12694

Ludwig, C., Hecht, R., Lautenbach, S., Schorcht, M., & Zipf, A. (2021). Mapping Public Urban Green Spaces Based on OpenStreetMap and Sentinel-2 Imagery Using Belief Functions. *ISPRS International Journal of Geo-Information*, *10*(4), 251. https://doi.org/10.3390/ijgi10040251

Luo, W., Li, Y., Urtasun, R., & Zemel, R. (2017). Understanding the Effective Receptive Field in Deep Convolutional Neural Networks. *29th Conference on Neural Information Processing Systems (NIPS 2016)*. Retrieved July 15, 2024, from http://arxiv.org/abs/1701.04128

Ma, L., Liu, Y., Zhang, X., Ye, Y., Yin, G., & Johnson, B. A. (2019). Deep learning in remote sensing applications: A meta-analysis and review. *ISPRS Journal of Photogrammetry and Remote Sensing*, *152*, 166–177. https://doi.org/10.1016/j.isprsjprs.2019.04.015

Majic, I., Winter, S., & Tomko, M. (2017). Finding equivalent keys in openstreetmap: Semantic similarity computation based on extensional definitions. *Proceedings of the 1st Workshop on Artificial Intelligence and Deep Learning for Geographic Knowledge Discovery*, 24–32. https://doi.org/10.1145/3149808.3149813

Mapzen. (2024). *Terrain Tiles - Registry of Open Data on AWS*. Retrieved June 16, 2024, from https://registry.opendata.aws/terrain-tiles/

Marshall, S., Gil, J., Kropf, K., Tomko, M., & Figueiredo, L. (2018). Street Network Studies: From Networks to Models and their Representations. *Networks and Spatial Economics*, *18*(3), 735–749. https://doi.org/10.1007/s11067-018-9427-9

Mehlin, V., Schacht, S., & Lanquillon, C. (2023). *Towards energy-efficient Deep Learning: An overview of energy-efficient approaches along the Deep Learning Lifecycle*. https://doi.org/10.48550/arXiv.2303.01980

Mi, L., & Chen, Z. (2020). Superpixel-enhanced deep neural forest for remote sensing image semantic segmentation. *ISPRS Journal of Photogrammetry and Remote Sensing*, *159*, 140–152. https://doi.org/10.1016/j.isprsjprs.2019.11.006

Microsoft. (2024). *Road Detections*. Retrieved May 24, 2024, from https://github.com/microsoft/RoadDetections

Minaee, S., Boykov, Y., Porikli, F., Plaza, A., Kehtarnavaz, N., & Terzopoulos, D. (2022). Image Segmentation Using Deep Learning: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *44*(7), 3523–3542. https://doi.org/10.1109/TPAMI.2021.3059968

Minghini, M., & Frassinelli, F. (2019). OpenStreetMap history for intrinsic quality assessment: Is OSM up-to-date? *Open Geospatial Data, Software and Standards*, *4*(1), 9. https://doi.org/10.1186/s40965-019-0067-x

Missing Maps. (2024). *Missing Maps*. Retrieved June 17, 2024, from https://www.missingmaps.org/

Mocnik, F.-B., Zipf, A., & Raifer, M. (2017). The OpenStreetMap folksonomy and its evolution. *Geo-spatial Information Science*, *20*(3), 219–230. https://doi.org/10.1080/10095020.2017.1368193

Mohammed, A., & Kora, R. (2023). A comprehensive review on ensemble deep learning: Opportunities and challenges. *Journal of King Saud University - Computer and Information Sciences*, *35*(2), 757–774. https://doi.org/10.1016/j.jksuci.2023.01.014

Moharram, M. A., & Sundaram, D. M. (2023). Land use and land cover classification with hyperspectral data: A comprehensive review of methods, challenges and future directions. *Neurocomputing*, *536*, 90–113. https://doi.org/10.1016/j.neucom.2023.03.025

Montazeri, A., Lilienthal, A. J., & Albertson, J. D. (2021). A spatial land use clustering framework for investigating the role of land use in mediating the effect of meteorology on urban air quality. *Atmospheric Environment: X*, *12*, 100126. https://doi.org/10.1016/j.aeaoa.2021.100126

Moradi, M., Roche, S., & Mostafavi, M. A. (2021). Exploring five indicators for the quality of OpenStreetMap road networks: A case study of Québec, Canada. *Geomatica*, *75*(4), 178–208. https://doi.org/10.1139/geomat-2021-0012

Murtagh, F., & Legendre, P. (2014). Ward's Hierarchical Agglomerative Clustering Method: Which Algorithms Implement Ward's Criterion? *Journal of Classification*, *31*(3), 274–295. https://doi.org/10.1007/s00357-014-9161-z

Neis, P., Goetz, M., & Zipf, A. (2012). Towards Automatic Vandalism Detection in OpenStreetMap. *ISPRS International Journal of Geo-Information*, *1*(3), 315–332. https://doi.org/10.3390/ijgi1030315

Neis, P., Zielstra, D., & Zipf, A. (2012). The Street Network Evolution of Crowdsourced Maps: OpenStreetMap in Germany 2007–2011. *Future Internet*, *4*, 1–21. https://doi.org/10.3390/fi4010001

Neptune Labs. (2024). *Neptune.ai - The MLOps stack component for experiment tracking*. neptune.ai. Retrieved June 17, 2024, from https://neptune.ai/

Netrapalli, P. (2019). Stochastic Gradient Descent and Its Variants in Machine Learning. *Journal of the Indian Institute of Science*, *99*(2), 201–213. https://doi.org/10.1007/s41745-019-0098-4

Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press.

Noh, H., Hong, S., & Han, B. (2015). Learning Deconvolution Network for Semantic Segmentation. *Proceedings of the IEEE International Conference on Computer Vision (ICCV 2015)*, 1520–1528. https://doi.org/10.48550/arXiv.1505.04366

NVIDIA Corporation. (2024). *CUDA Zone*. Retrieved June 24, 2024, from https://developer.nvidia.com/cuda-zone

Ongsulee, P. (2017). Artificial intelligence, machine learning and deep learning. *2017 15th International Conference on ICT and Knowledge Engineering (ICT&KE)*, 1–6. https://doi.org/10.1109/ICTKE.2017.8259629

Onim, M. S. H., Ehtesham, A. R. B., Anbar, A., Nazrul Islam, A. K. M., & Mahbubur Rahman, A. K. M. (2020). LULC classification by semantic segmentation of satellite images using FastFCN. *2020 2nd International Conference on Advanced Information and Communication Technology (ICAICT)*, 471–475. https://doi.org/10.1109/ICAICT51780.2020.9333522

ONNX.ai. (2024). *ONNX Concepts*. Retrieved January 29, 2024, from https://onnx.ai/onnx/intro/concepts.html

OpenAI. (2022). *ChatGPT*. Retrieved May 24, 2024, from https://chatgpt.com

OSM Foundation. (2024). *OSM Taginfo*. Retrieved June 12, 2024, from https://taginfo.openstreetmap.org/about

OSM Taginfo. (2024). *Highway key*. Retrieved May 5, 2024, from https://taginfo.openstreetmap.org/keys/highway?filter=ways#values

OSM Wiki. (2024a). *Default speed limits*. Retrieved June 15, 2024, from https://wiki.openstreetmap.org/wiki/Default_speed_limits

OSM Wiki. (2024b). *Elements*. Retrieved June 10, 2024, from https://wiki.openstreetmap.org/wiki/Elements

OSM Wiki. (2024c). *Key: Maxspeed*. Retrieved May 29, 2024, from https://wiki.openstreetmap.org/wiki/Key:maxspeed

Pan, X., Ge, C., Lu, R., Song, S., Chen, G., Huang, Z., & Huang, G. (2022). On the Integration of Self-Attention and Convolution. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2022)*, 815–825. https://doi.org/10.48550/arXiv.2111.14556

Park, N., & Kim, S. (2022). How Do Vision Transformers Work? *10th International Conference for Learning Representations (ICLR 2022)*. https://doi.org/10.48550/arXiv.2202.06709

Petrakis, R. E., Norman, L. M., Vaughn, K., Pritzlaff, R., Weaver, C., Rader, A., & Pulliam, H. R. (2021). Hierarchical Clustering for Paired Watershed Experiments: Case Study in Southeastern Arizona, U.S.A. *Water*, *13*(21), 2955. https://doi.org/10.3390/w13212955

Potdar, K., Pardawala, T., & Pai, C. (2017). A Comparative Study of Categorical Variable Encoding Techniques for Neural Network Classifiers. *International Journal of Computer Applications*, *175*, 7–9. https://doi.org/10.5120/ijca2017915495

PowerAPI. (2024). *pyJoules* (Version 0.5.1). Retrieved June 9, 2024, from https://pyjoules.readthedocs.io/en/latest/

Powers, D., & Ailab. (2011). Evaluation: From precision, recall and F-measure to ROC, informedness, markedness & correlation. *J. Mach. Learn. Technol*, *2*, 2229–3981. https://doi.org/10.9735/2229-3981

Qi, K., Wu, H., Shen, C., & Gong, J. (2015). Land-Use Scene Classification in High-Resolution Remote Sensing Images Using Improved Correlatons. *IEEE Geoscience and Remote Sensing Letters*, *12*(12), 2403–2407. https://doi.org/10.1109/LGRS.2015.2478966

Raifer, M., Troilo, R., Kowatsch, F., Auer, M., Loos, L., Marx, S., Przybill, K., Fendrich, S., Mocnik, F.-B., & Zipf, A. (2019). OSHDB: A framework for spatio-temporal analysis of OpenStreetMap history data. *Open Geospatial Data, Software and Standards*, *4*(1), 3. https://doi.org/10.1186/s40965-019-0061-3

Rangaraj, A. G., ShobanaDevi, A., Srinath, Y., Boopathi, K., & Balaraman, K. (2022). Efficient and Secure Storage for Renewable Energy Resource Data Using Parquet for Data Analytics. In N. Sharma, A. Chakrabarti, V. E. Balas, & A. M. Bruckstein (Eds.), *Data Management, Analytics and Innovation* (pp. 263–292). Springer. https://doi.org/10.1007/978-981-16-2937-2_19

Rangel, A., Terven, J., Cordova-Esparza, D. M., & Chavez-Urbiola, E. A. (2024). Land Cover Image Classification. *12th International Conference for Learning Representations (ICLR 2024)*. https://doi.org/10.48550/arXiv.2401.09607

Reichstein, M., Camps-Valls, G., Stevens, B., Jung, M., Denzler, J., Carvalhais, N., & Prabhat. (2019). Deep learning and process understanding for data-driven Earth system science. *Nature*, *566*(7743), 195–204. https://doi.org/10.1038/s41586-019-0912-1

Rhein-Neckar-Kreis. (2024). *Landkreis*. Retrieved June 11, 2024, from https://www.rhein-neckar-kreis.de/start/landkreis.html

Rhine-Neckar Metropolitan Region. (2024). *Zahlen und fakten über rhein-neckar*. Retrieved June 4, 2024, from https://www.m-r-n.com/zahlen-und-fakten

Road and Transportation Research Association. (2011). *Guidelines for the Design of Motorways (RAA)*. Committee: Motorways.

Rolf, E., Klemmer, K., Robinson, C., & Kerner, H. (2024). Mission Critical – Satellite Data is a Distinct Modality in Machine Learning. *International Conference on Machine Learning (ICML)(Opens in New Tab)*. https://doi.org/10.48550/arXiv.2402.01444

Rosenblatt, F. (1957). *The Perceptron, a Perceiving and Recognizing Automaton Project Para*. Cornell Aeronautical Laboratory.

Roumeliotis, K. I., & Tselikas, N. D. (2023). ChatGPT and Open-AI Models: A Preliminary Review. *Future Internet*, *15*(6), 192. https://doi.org/10.3390/fi15060192

Sarker, I. H. (2021). Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions. *SN Computer Science*, *2*(6), 420. https://doi.org/10.1007/s42979-021-00815-1

Schott, M., Zell, A., Lautenbach, S., Sumbul, G., Schultz, M., Zipf, A., & Demir, B. (2024). Analyzing and Improving the Quality and Fitness for Purpose of OpenStreetMap as Labels in Remote Sensing Applications. In D. Burghardt, E. Demidova, & D. A. Keim (Eds.), *Volunteered Geographic Information: Interpretation, Visualization and Social Context* (pp. 21–42). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-35374-1_2

Schultz, M., Voss, J., Auer, M., Carter, S., & Zipf, A. (2017). Open land cover from OpenStreetMap and remote sensing. *International Journal of Applied Earth Observation and Geoinformation*, *63*, 206–213. https://doi.org/10.1016/j.jag.2017.07.014

Sentinel Hub. (2024). *Sentinel Hub Process API*. Retrieved June 14, 2024, from https://sentinelhub-py.readthedocs.io/en/latest/examples/process_request.html

Shanmugamani, R. (2018). *Deep Learning for Computer Vision: Expert techniques to train advanced neural networks using TensorFlow and Keras*. Packt Publishing Ltd.

Shinde, P. P., & Shah, S. (2018). A Review of Machine Learning and Deep Learning Applications. *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, 1–6. https://doi.org/10.1109/ICCUBEA.2018.8697857

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of machine learning research*, *15*, 1929–1958. https://dl.acm.org/doi/abs/10.5555/2627435.2670313

Srivastava, S., Vargas Muñoz, J., & Tuia, D. (2019). Understanding urban landuse from the above and ground perspectives: A deep learning, multimodal solution. *Remote Sensing of Environment*, *228*, 129–143. https://doi.org/10.1016/j.rse.2019.04.014

Strubell, E., Ganesh, A., & McCallum, A. (2019). Energy and Policy Considerations for Deep Learning in NLP. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 3645–3650. https://doi.org/10.48550/arXiv.1906.02243

Szeliski, R. (2022). *Computer Vision: Algorithms and Applications*. Springer International Publishing. https://doi.org/10.1007/978-3-030-34372-9

Tao, C., Meng, Y., Li, J., Yang, B., Hu, F., Li, Y., Cui, C., & Zhang, W. (2022). MSNet: Multispectral semantic segmentation network for remote sensing images. *GIScience & Remote Sensing*, *59*(1), 1177–1198. https://doi.org/10.1080/15481603.2022.2101728

Thanh Noi, P., & Kappas, M. (2018). Comparison of Random Forest, k-Nearest Neighbor, and Support Vector Machine Classifiers for Land Cover Classification Using Sentinel-2 Imagery. *Sensors*, *18*(1), 18. https://doi.org/10.3390/s18010018

Thompson, N. C., Greenewald, K., Lee, K., & Manso, G. F. (2022). The Computational Limits of Deep Learning. *9th Computing within Limits (LIMITS 2023)*. https://doi.org/10.48550/arXiv.2007.05558

Tobler, W. R. (1970). A Computer Movie Simulating Urban Growth in the Detroit Region. *Economic Geography*, *46*, 234–240. https://doi.org/10.2307/143141

Torchmetrics. (2024a). *Accuracy*. Retrieved May 30, 2024, from https://lightning.ai/docs/torchmetrics/stable/classification/accuracy.html

Torchmetrics. (2024b). *Jaccard Index*. Retrieved May 30, 2024, from https://lightning.ai/docs/torchmetrics/stable/classification/jaccard_index.html

Torchmetrics. (2024c). *TorchMetrics Package*. Retrieved June 4, 2024, from https://lightning.ai/docs/torchmetrics/stable/

Torchvision. (2024). *Illustration of transforms*. Retrieved June 25, 2024, from https://pytorch.org/vision/stable/auto_examples/transforms/plot_transforms_illustrations.html#sphx-glr-auto-examples-transforms-plot-transforms-illustrations-py

Tzepkenlis, A., Marthoglou, K., & Grammalidis, N. (2023). Efficient Deep Semantic Segmentation for Land Cover Classification Using Sentinel Imagery. *Remote Sensing*, *15*(8), 2027. https://doi.org/10.3390/rs15082027

United Nations. (2023). *The Sustainable Development Goals Report 2023: Special Edition*. https://doi.org/10.18356/9789210024914

University of Michigan Center for Sustainable Systems. (2023). *Carbon Footprint Factsheet*. Retrieved May 22, 2024, from https://css.umich.edu/publications/factsheets/sustainability-indicators/carbon-footprint-factsheet

Usmani, M., Napolitano, M., & Bovolo, F. (2023). Towards global scale segmentation with OpenStreetMap and remote sensing. *ISPRS Open Journal of Photogrammetry and Remote Sensing*, *8*, 100031. https://doi.org/10.1016/j.ophoto.2023.100031

Ustin, S. L., & Middleton, E. M. (2021). Current and near-term advances in Earth observation for ecological applications. *Ecological Processes*, *10*(1), 1. https://doi.org/10.1186/s13717-020-00255-4

Vargas-Munoz, J. E., Srivastava, S., Tuia, D., & Falcão, A. X. (2021). OpenStreetMap: Challenges and Opportunities in Machine Learning and Remote Sensing. *IEEE Geoscience and Remote Sensing Magazine*, *9*(1), 184–199. https://doi.org/10.1109/MGRS.2020.2994107

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention Is All You Need. *31st Conference on Neural Information Processing Systems (NIPS 2017)*. https://doi.org/10.48550/arXiv.1706.03762

Vitkienė, J., Puodžiukas, V., & Zilioniene, D. (2017). New Approach to the Lithuanian Road Classification Based on Worldwide Experience. *"Environmental Engineering" 10th International Conference*. https://doi.org/10.3846/enviro.2017.155

Wang, W., Xie, E., Li, X., Fan, D.-P., Song, K., Liang, D., Lu, T., Luo, P., & Shao, L. (2021). Pyramid Vision Transformer: A Versatile Backbone for Dense Prediction without Convolutions. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 548–558. https://doi.org/10.1109/ICCV48922.2021.00061

Wang, Y., Sun, Y., Cao, X., Wang, Y., Zhang, W., & Cheng, X. (2023). A review of regional and Global scale Land Use/Land Cover (LULC) mapping products generated from satellite remote sensing. *ISPRS Journal of Photogrammetry and Remote Sensing*, *206*, 311–334. https://doi.org/10.1016/j.isprsjprs.2023.11.014

Wang, Z., Wang, P., Liu, K., Wang, P., Fu, Y., Lu, C.-T., Aggarwal, C. C., Pei, J., & Zhou, Y. (2024). *A Comprehensive Survey on Data Augmentation*. https://doi.org/10.48550/arXiv.2405.09591

Wang, Z., Wang, Q., Yang, Y., Liu, N., Chen, Y., & Gao, J. (2023). Seismic Facies Segmentation via a Segformer-Based Specific Encoder–Decoder–Hypercolumns Scheme. *IEEE Transactions on Geoscience and Remote Sensing*, *61*, 1–11. https://doi.org/10.1109/TGRS.2023.3244037

Ward, J. H. (1963). Hierarchical Grouping to Optimize an Objective Function. *Journal of the American Statistical Association*, *58*(301), 236–244. https://doi.org/10.2307/2282967

Xie, E., Wang, W., Yu, Z., Anandkumar, A., Alvarez, J. M., & Luo, P. (2021). SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers. *35th Conference on Neural Information Processing Systems (NeurIPS 2021)*, *34*, 12077–12090. https://doi.org/10.48550/arXiv.2105.15203

Xu, J., Zhou, W., Fu, Z., Zhou, H., & Li, L. (2021). *A Survey on Green Deep Learning*.

Xu, Y., Zhou, B., Jin, S., Xie, X., Chen, Z., Hu, S., & He, N. (2022). A framework for urban land use classification by integrating the spatial context of points of interest and graph convolutional neural network method. *Computers, Environment and Urban Systems*, *95*, 101807. https://doi.org/10.1016/j.compenvurbsys.2022.101807

Yang, D., Fu, C.-S., Smith, A. C., & Yu, Q. (2017). Open land-use map: A regional land-use mapping strategy for incorporating OpenStreetMap with earth observations. *Geo-spatial Information Science*, *20*(3), 269–281. https://doi.org/10.1080/10095020.2017.1371385

Yao, S., Chen, C., He, M., Cui, Z., Mo, K., Pang, R., & Chen, Q. (2023). Land use as an important indicator for water quality prediction in a region under rapid urbanization. *Ecological Indicators*, *146*, 109768. https://doi.org/10.1016/j.ecolind.2022.109768

Yu, L., Liang, L., Wang, J., Zhao, Y., Cheng, Q., Hu, L., Liu, S., Yu, L., Wang, X., Zhu, P., Li, X., Xu, Y., Li, C., Fu, W., Li, X., Li, W., Liu, C., Cong, N., Zhang, H., … Gong, P. (2014). Meta-discoveries from a synthesis of satellite-based land-cover mapping research. *International Journal of Remote Sensing*, *35*(13), 4573–4588. https://doi.org/10.1080/01431161.2014.930206

Zaitsev, I. (2019, March 29). *The Best Format to Save Pandas Data*. Medium. Retrieved May 4, 2024, from https://towardsdatascience.com/the-best-format-to-save-pandas-data-414dca023e0d

Zeng, C., Zhao, Z., Wen, C., Yang, J., & Lv, T. (2020). Effect of Complex Road Networks on Intensive Land Use in China's Beijing-Tianjin-Hebei Urban Agglomeration. *Land*, *9*(12), 532. https://doi.org/10.3390/land9120532

Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2023). *Dive into Deep Learning* (5th ed.). Cambridge University Press.

Zhang, C., & Li, X. (2022). Land Use and Land Cover Mapping in the Era of Big Data. *Land*, *11*(10), 1692. https://doi.org/10.3390/land11101692

Zhang, J., Zhang, Q., Ren, J., Zhao, Y., & Liu, J. (2022). Spatial-context-aware deep neural network for multi-class image classification. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2022)*. https://doi.org/10.48550/arXiv.2111.12296

Zhao, S., Tu, K., Ye, S., Tang, H., Hu, Y., & Xie, C. (2023). Land Use and Land Cover Classification Meets Deep Learning: A Review. *Sensors*, *23*(21), 8966. https://doi.org/10.3390/s23218966

Zhao, Y., Ning, Y., Li, H., Liao, Z., Liu, Y., & Li, F. (2023). MSC-DeepFM: OSM Road Type Prediction via Integrating Spatial Context Using DeepFM. *Sustainability*, *15*(24), 16671. https://doi.org/10.3390/su152416671

Zheng, S., Lu, J., Zhao, H., Zhu, X., Luo, Z., Wang, Y., Fu, Y., Feng, J., Xiang, T., Torr, P. H., & Zhang, L. (2021). Rethinking Semantic Segmentation from a Sequence-to-Sequence Perspective with Transformers. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2021)*, 6877–6886. https://doi.org/10.1109/CVPR46437.2021.00681

Zong, L., He, S., Lian, J., Bie, Q., Wang, X., Dong, J., & Xie, Y. (2020). Detailed Mapping of Urban Land Use Based on Multi-Source Data: A Case Study of Lanzhou. *Remote Sensing*, *12*(12), 1987. https://doi.org/10.3390/rs12121987

199 entries.

Chapter A

Appendix

A.1 Code Availability & Repositories

Files created for this study can be found in the GitHub repository at *https://github.com/GrHalbgott/university/tree/main/6_thesis_M.Sc.* The folder "relationship_analysis" contains the scripts used for exploring the road network attributes according to the used attributes as well as the relationship analysis between roads and adjacent LULC classes. Additionally, the repository contains scripts for code verification and testing, comparison of speeds as shown in appendix A.2 further below, and the calculation and plotting of the evaluation metrics derived from Neptune.ai. Also, the main additions to the LULC Utility (mentioned below) are included as well. This repository is publicly accessible.

The adapted LULC Utility can be found in the GitLab repository at *https://gitlab.gistools.geog.uni-heidelberg.de/nkolaxidis/lulc-utility.* The main script where the extensions are called and integrated into the feature space is *lulc/data/ dataset.py.* The handling of the road network and the different functions required for querying, preprocessing, feature engineering, encoding, and rasterizing the road network data, are implemented in the *lulc/ops/ road_network_operator.py* script, which is the main addition to the LULC Utility in context of this thesis. The configuration variables specifically for the road network and its feature engineering process are in the file *assets/ road_network.yaml.* The configuration files for the baseline model and each extension can be found in the *conf/model* folder, their normalization values in the *conf/data* folder. Other parts of the code have been adapted to enable an easier training workflow and enhanced user experience, but they are not related to the thesis itself, therefore they are not specifically mentioned here. All changes and additions to the LULC Utility are summarized in the merge request under *https://gitlab.gistools.geog.uni-heidelberg.de/nkolaxidis/lulc-utility/-/merge_requests/3.* The project is only accessible with a login.

Results of the model runs can be found in the Neptune.ai project at *https://app.neptune.ai/o/HeiGIT/org/RoadNetwork-integration.* The project contains the results of all model runs, including the evaluation metrics and confusion matrices, as well as segmentation samples. The project is only accessible with a login and an invitation.

A.2 Default and Country-Specific Speed Limits

The file used for getting country-specific and default speed limits per road type is found here: *https://raw.githubusercontent.com/GIScience/openrouteservice/master/ors-engine/src/main/resources/resources/services/routing/speed_limits/car.json.* It is part of the OpenRouteService, a routing engine developed by the HeiGIT, and the speeds apply for cars only.

A.3 Maps of Mannheim - Heidelberg - Rhein-Neckar-Kreis

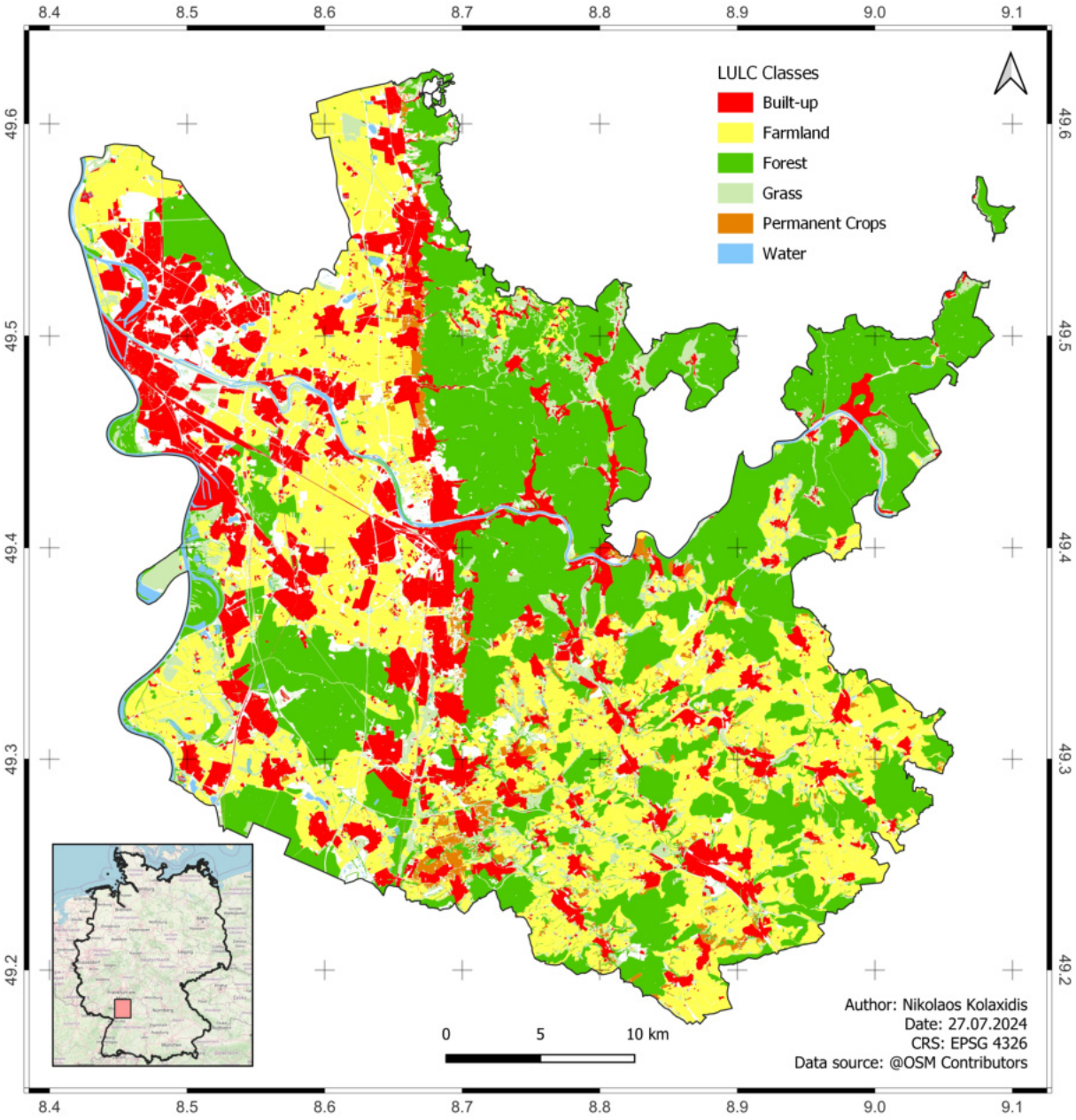


FIGURE A.1: Aggregated LULC Classes of MA-HD-RNK. The large share of the classes "forest" and "farmland" is easily observable as well as the two larger and many spatially distributed smaller cities aggregated in the "built-up" class.

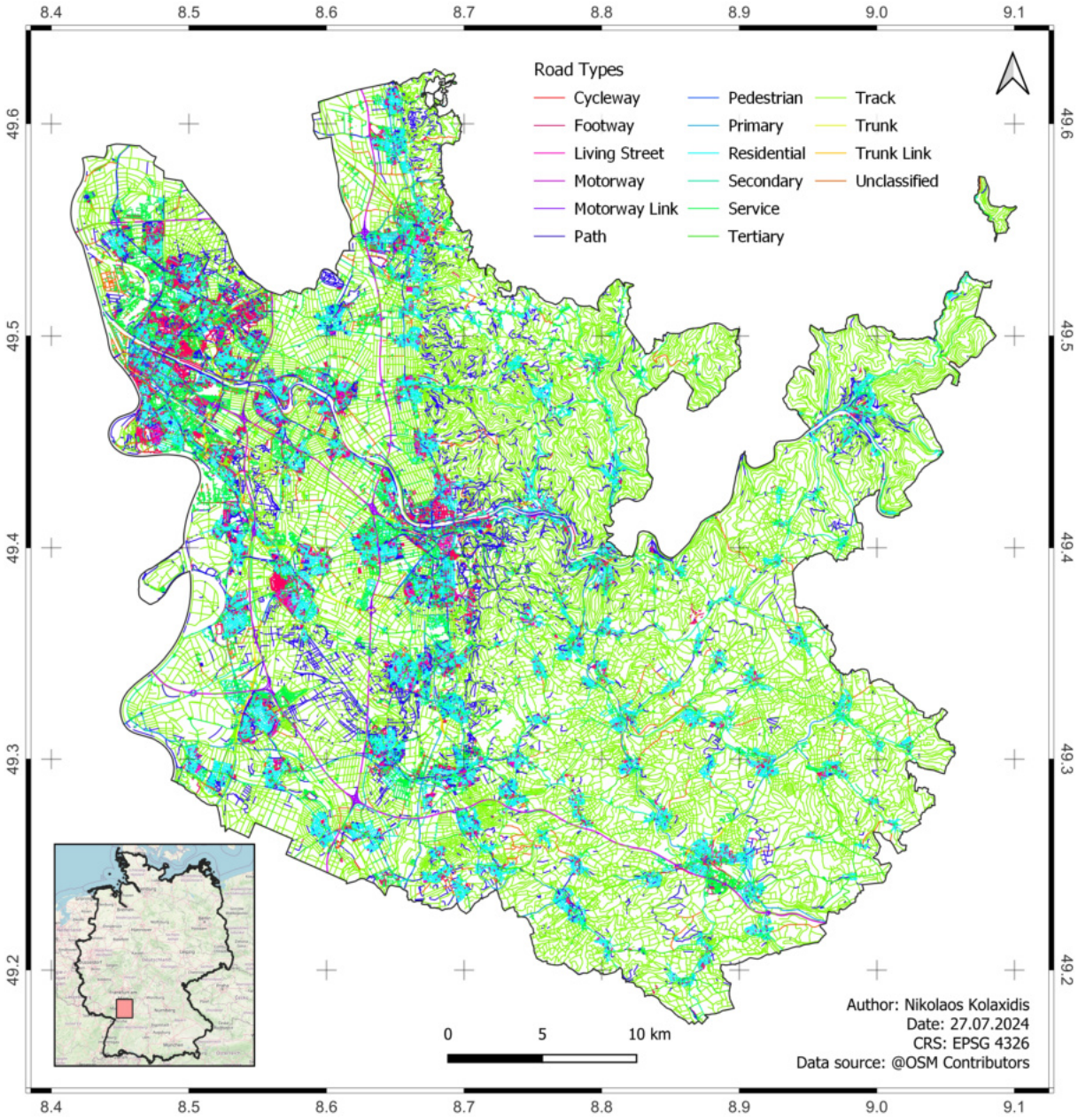


FIGURE A.2: Road types of the AOI MA-HD-RNK after feature engineering. The high prevalence of *track* is easily observable.

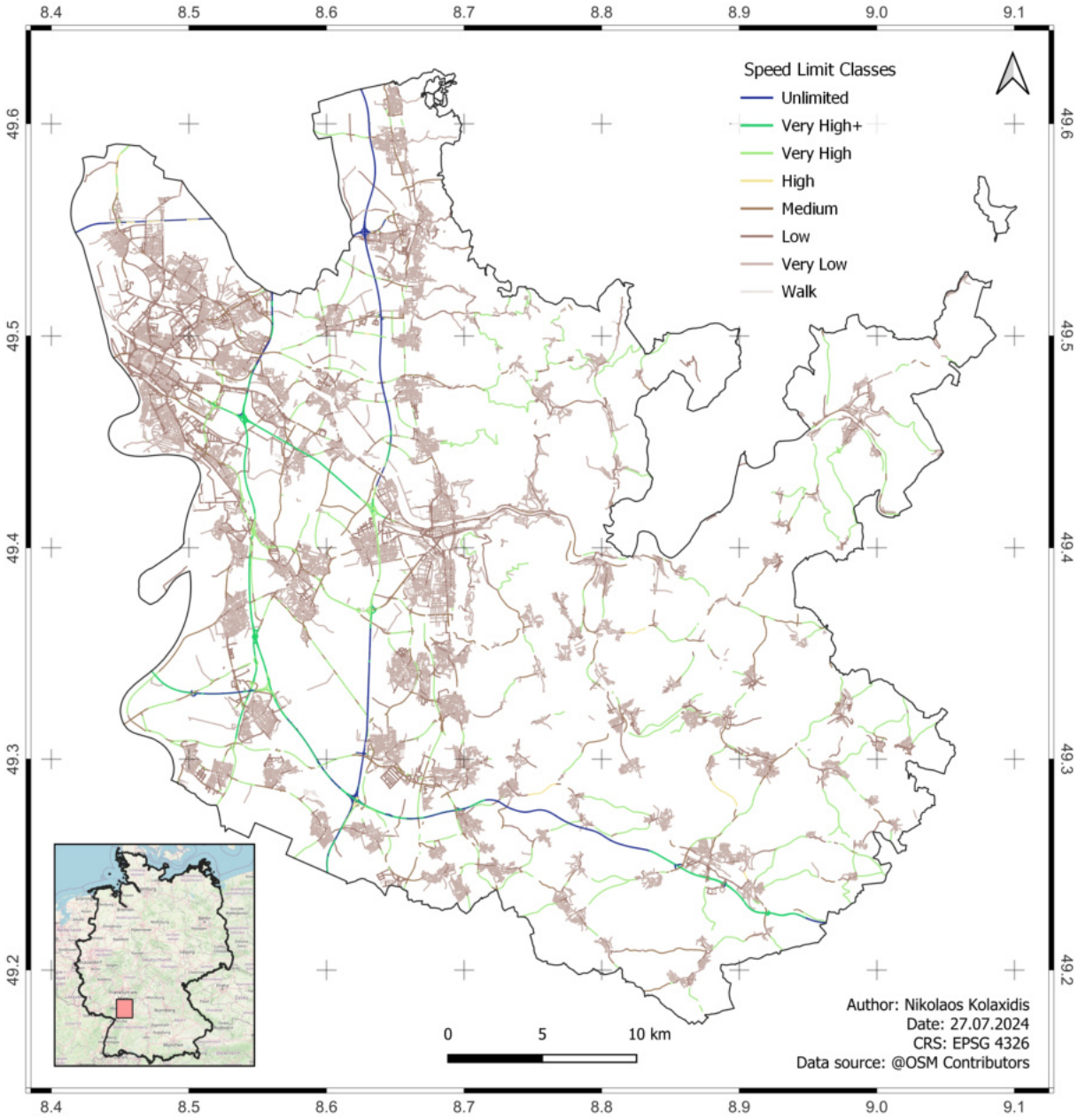


FIGURE A.3: Speed limit classes of the AOI MA-HD-RNK after feature engineering. The whitespace in comparison to the other road networks is due to the high share of missing values in the speed tags.

A.4 Relationship Between Roads and LULC Classes

The following pages contain additional material regarding the explorative relationship analysis between roads and LULC classes.

A.4.1 Visual Relationship

The following figures are a continuation of figure 4.1 in the results section 4.1.1.

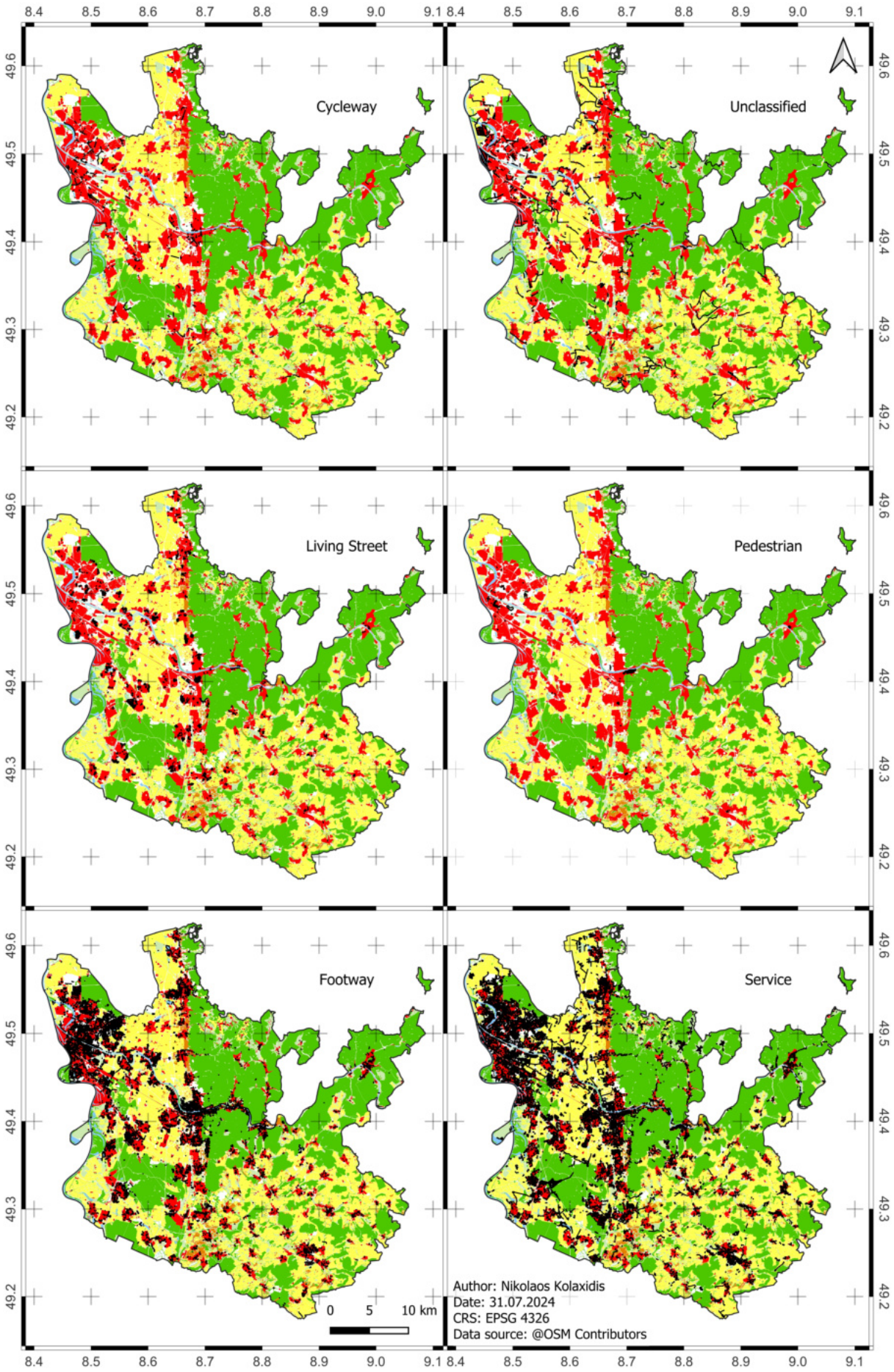


FIGURE A.4: Visual Relationship between road types and LULC classes for MA-HD-RNK, first continuation of figure 4.1. The background is figure A.1 in appendix section A.3. Roads are black.

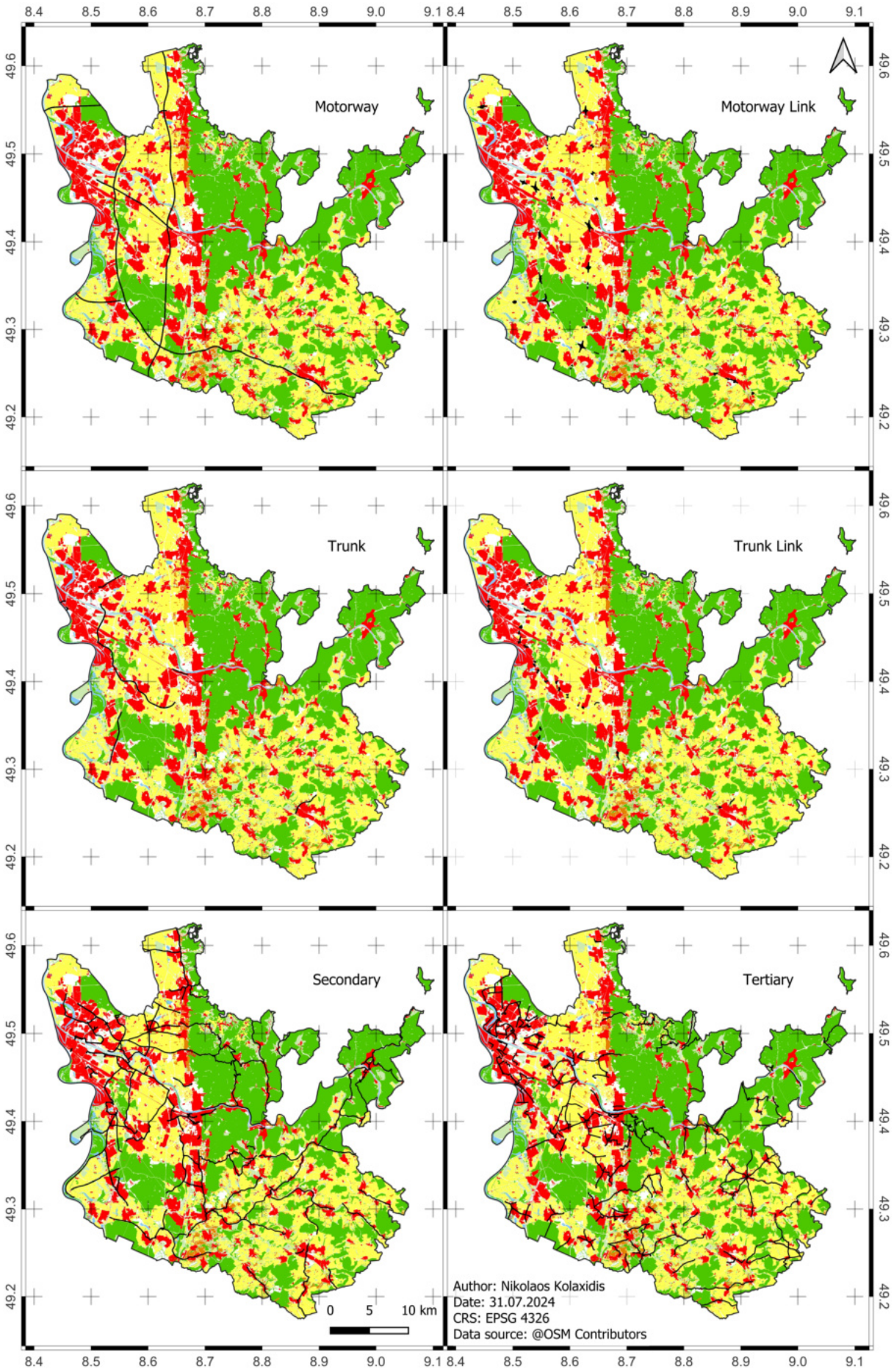


FIGURE A.5: Visual Relationship between road types and LULC classes for MA-HD-RNK, second continuation of figure 4.1. The background is figure A.1 in appendix section A.3. Roads are black.

A.4.2 Buffered Road Types vs. LULC Classes

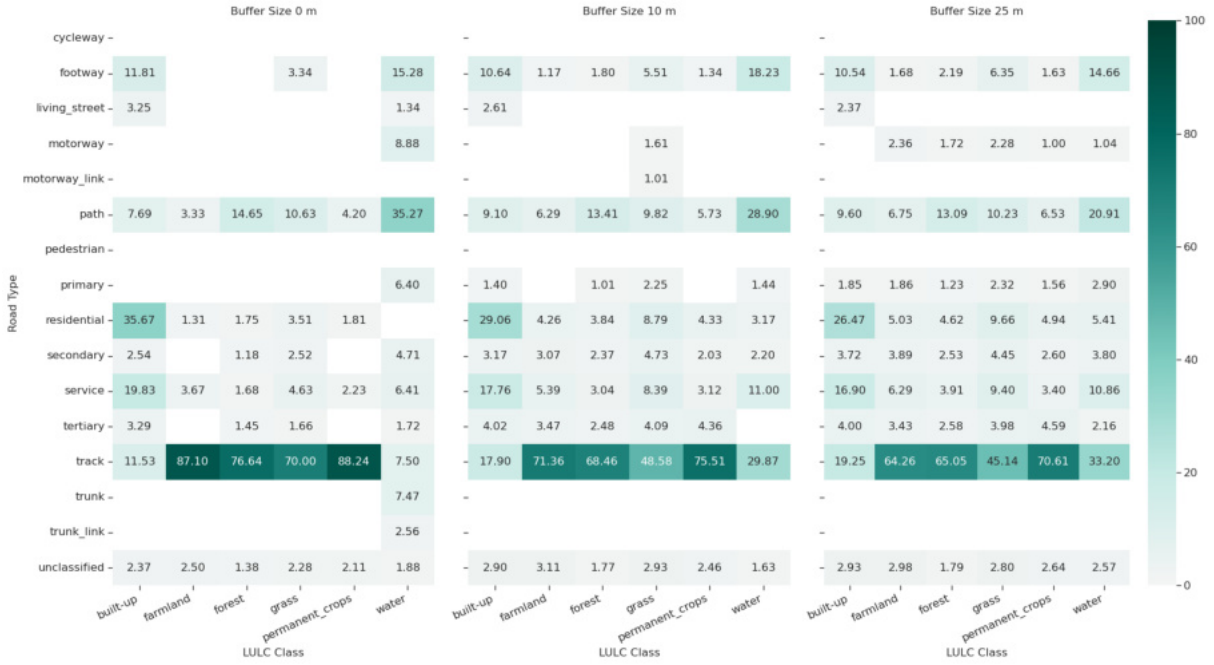


FIGURE A.6: Heatmaps showing the relationship between road types and LULC classes for the buffer sizes 0, 10, and 25 m. Summation to 100 % vertically. All values under 1 % are omitted for clarity, summation includes the omitted values.

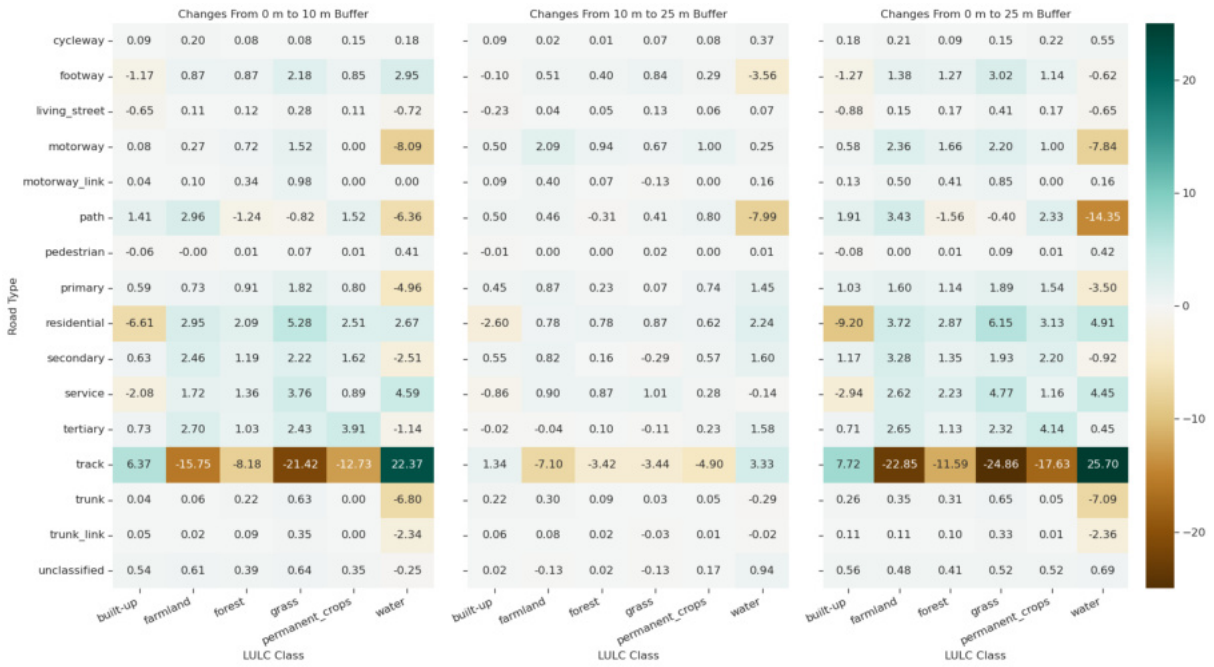
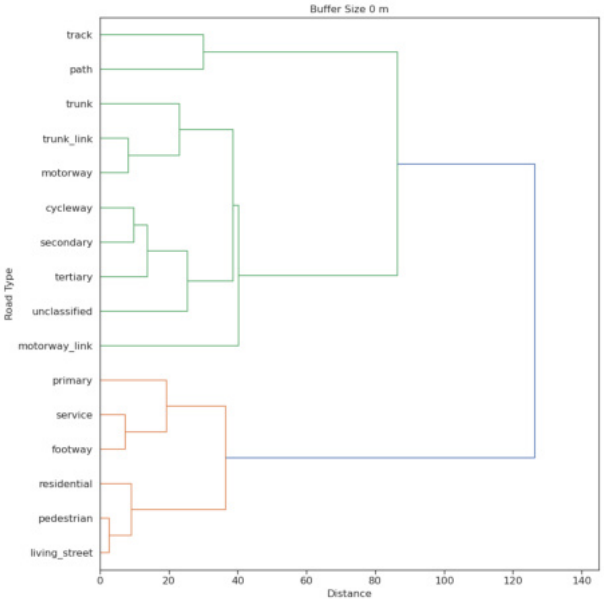


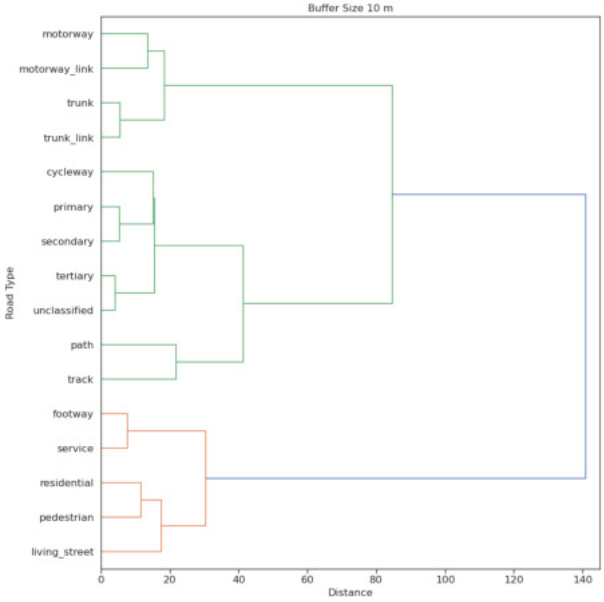
FIGURE A.7: Heatmaps showing the changes between the different buffer sizes 0, 10, and 25 m of the relationship between road types and LULC classes. Summation to 100 % vertically.

FIGURE A.8: Heatmaps showing the relationship between road types and LULC classes for the buffer sizes 0, 10, and 25 m. Summation to 100 % horizontally. All values under 1 % are omitted for clarity, summation includes the omitted values.

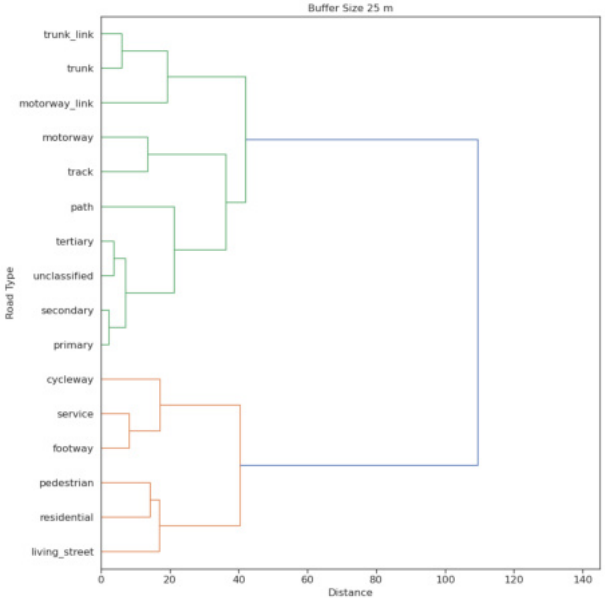
FIGURE A.9: Heatmaps showing the changes between the different buffer sizes 0, 10, and 25 m of the relationship between road types and LULC classes. Summation to 100 % horizontally.



(A) No applied buffer.



(B) Buffered Road Network with 10 m.



(C) Buffered Road Network with 25 m.

FIGURE A.10: Dendrograms showing the HCA of road types based on their distribution similarity over LULC classes for no applied buffer (A), 10 m buffer (B), and 25 m buffer (C). Road types *residential, pedestrian*, and *living_street* have been clustered together based on this result to decrease the number of channels for extension 2 (see section 3.5.3).

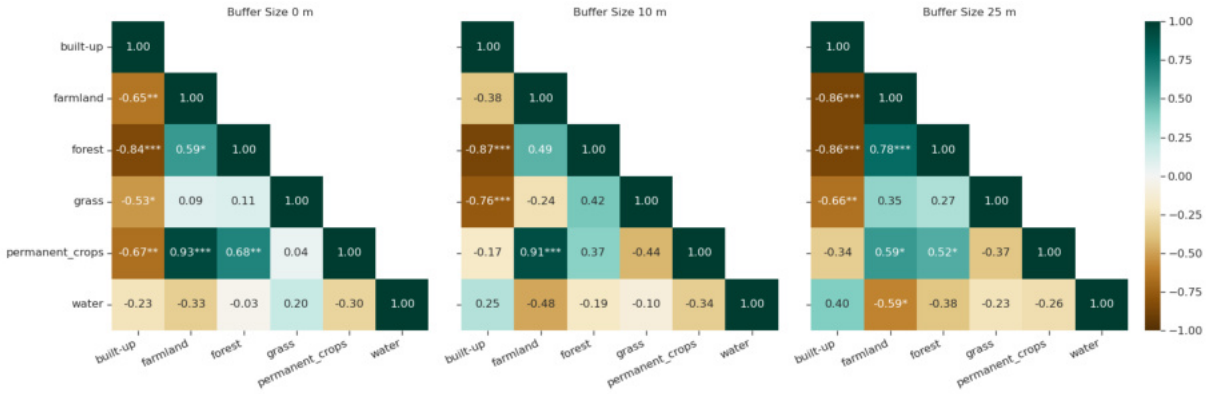


FIGURE A.11: PPMC of LULC classes based on road types distribution, for the buffer sizes 0, 10, and 25 m. Asterisks show the according significance level at 0.05 (*), 0.01 (**), and 0.001 (***), respectively. The upper triangle is omitted for clarity.

A.4.3 Buffered Speed Limit Classes vs. LULC Classes

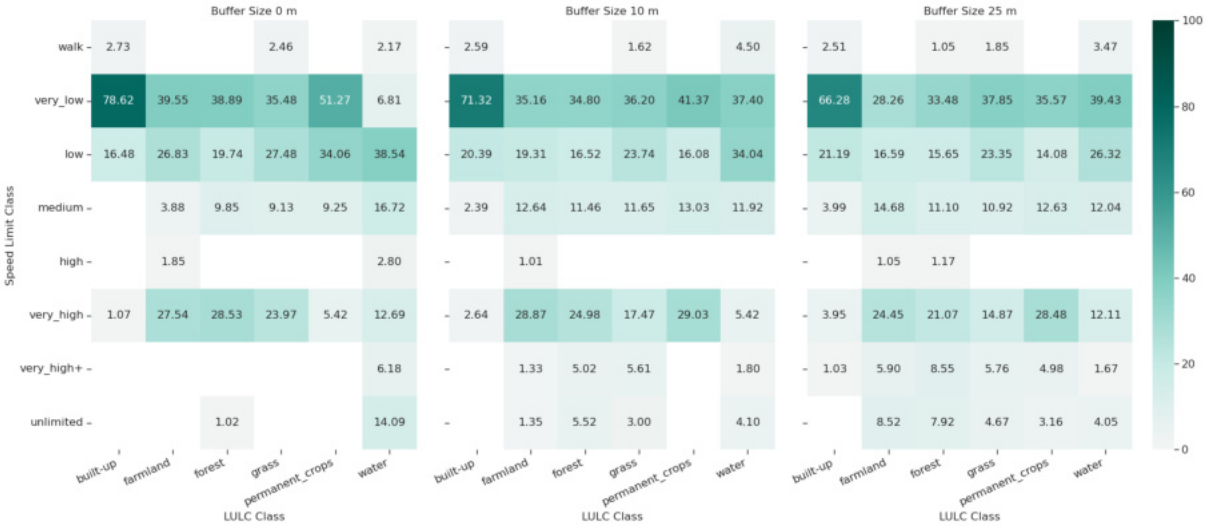


FIGURE A.12: Heatmaps showing the relationship between speed limit and LULC classes for the buffer sizes 0, 10, and 25 m. Summation to 100 % vertically. All values under 1 % are omitted for clarity, summation includes the omitted values.

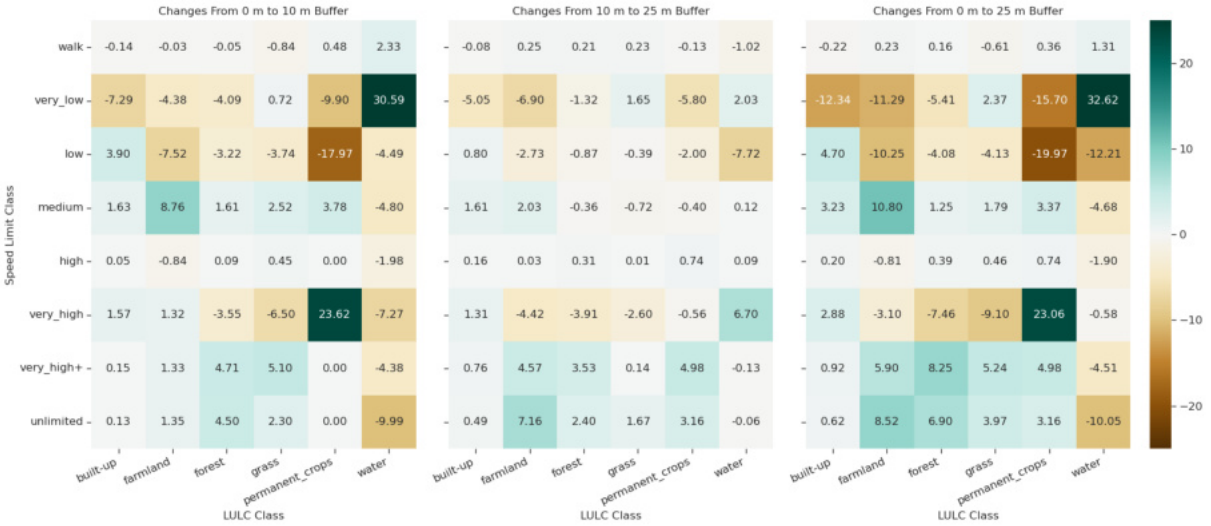


FIGURE A.13: Heatmaps showing the changes between the different buffer sizes 0, 10, and 25 m of the relationship between speed limit and LULC classes. Summation to 100 % vertically.

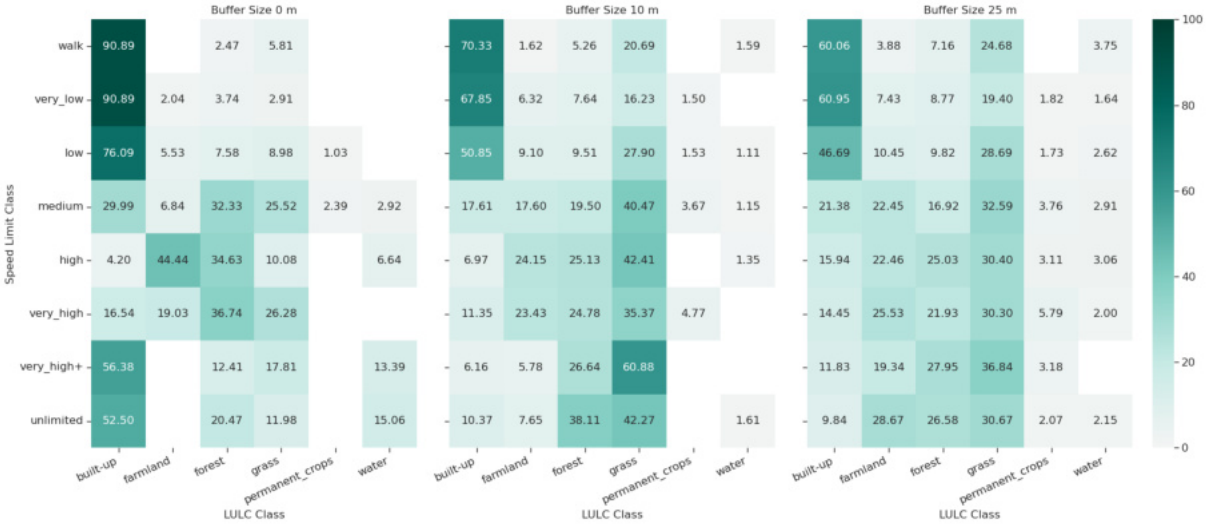


FIGURE A.14: Heatmaps showing the relationship between speed limit and LULC classes for the buffer sizes 0, 10, and 25 m. Summation to 100 % horizontally. All values under 1 % are omitted for clarity, summation includes the omitted values.

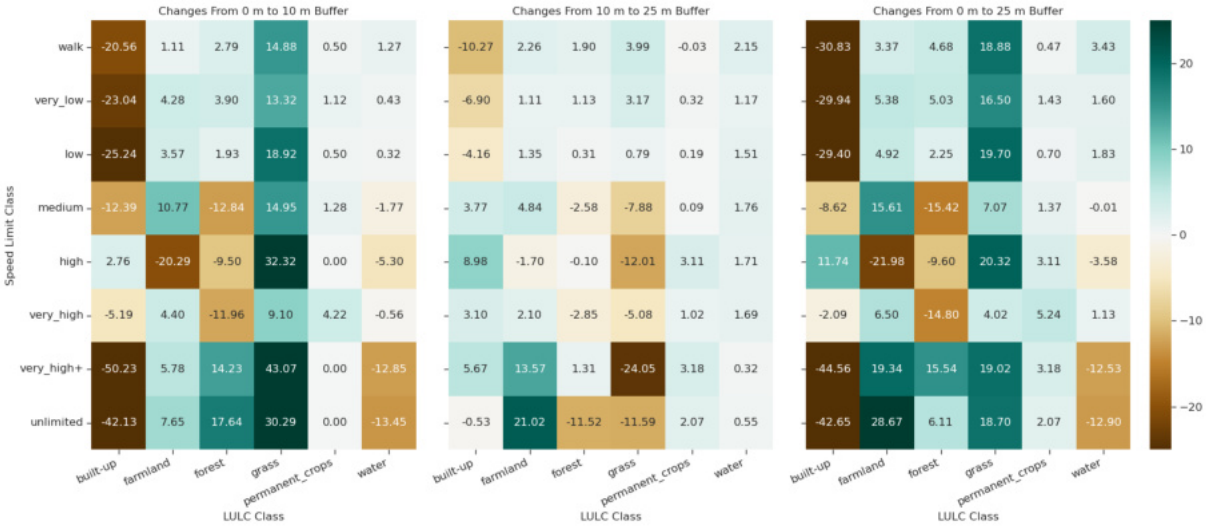
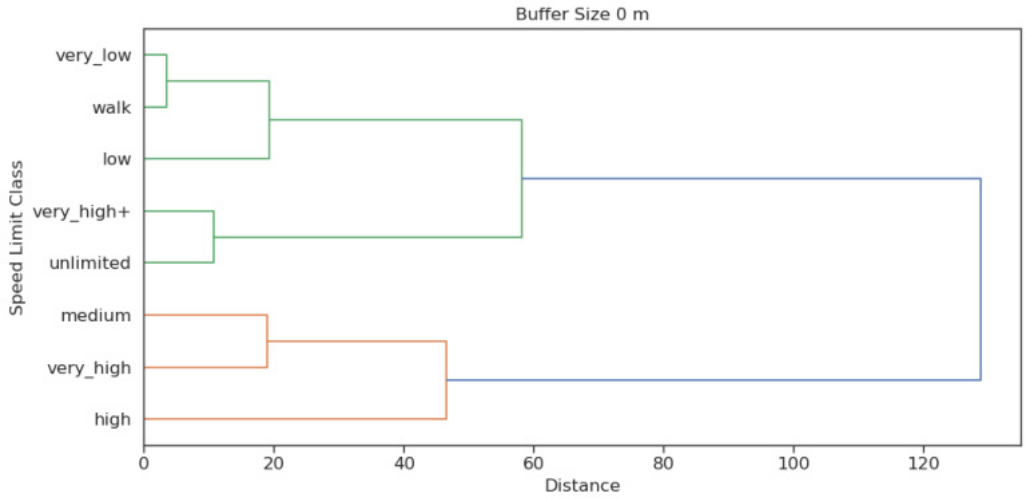
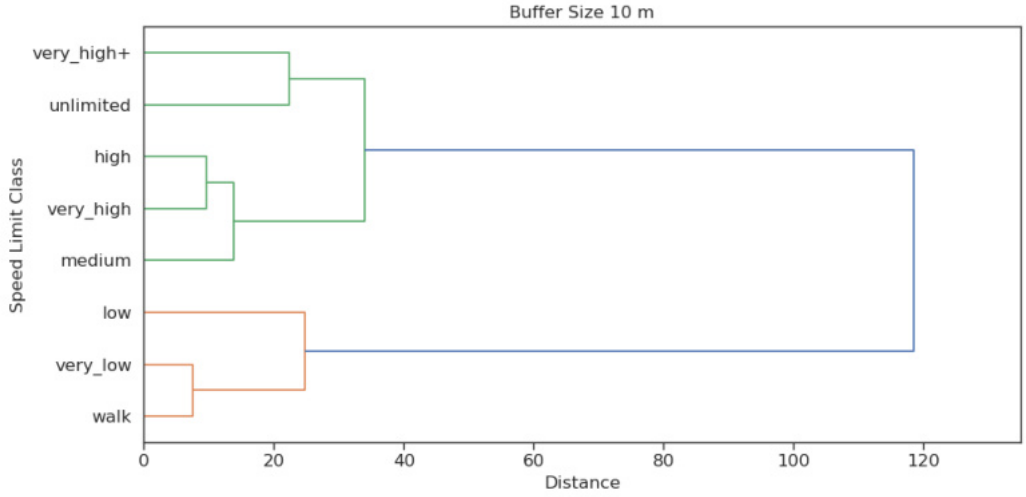


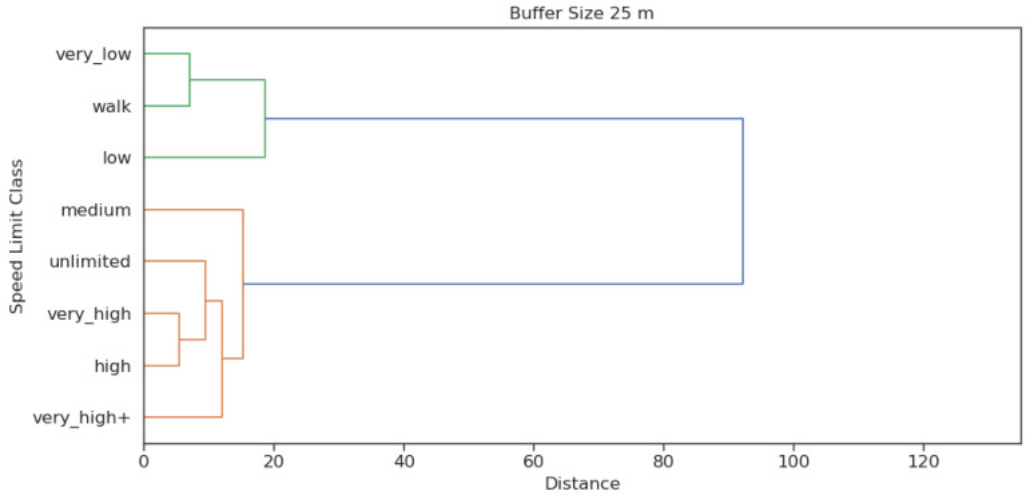
FIGURE A.15: Heatmaps showing the changes between the different buffer sizes 0, 10, and 25 m of the relationship between speed limit and LULC classes. Summation to 100 % horizontally.



(A) No applied buffer.



(B) Buffered Road Network with 10 m.



(C) Buffered Road Network with 25 m.

FIGURE A.16: Dendrograms showing the HCA of speed limit classes based on their distribution similarity over LULC classes for no applied buffer (A), 10 m buffer (B), and 25 m buffer (C). The Speed Limit Classes are clusters based on speed limit ranges (see section 3.5.4).

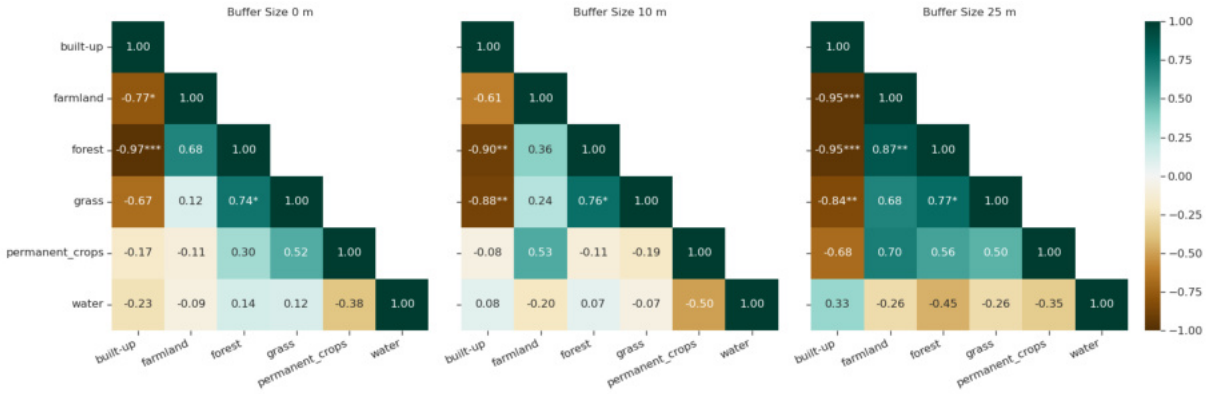


FIGURE A.17: PPMC of LULC classes based on speed limit classes distribution, for the buffer sizes 0, 10, and 25 m. Asterisks show the according significance level at 0.05 (*), 0.01 (**), and 0.001 (***), respectively. The upper triangle is omitted for clarity.

A.5 Model Evaluation

A.5.1 Performance of All Model Runs

TABLE A.1: Maximal achieved values for OA and IoU in all three phases of all model runs.

Model Run	Train OA [%]	Val OA [%]	Test OA [%]	Train IoU [%]	Val IoU [%]	Test IoU [%]
Baseline 1	67.38	74.84	76.69	36.86	40.48	36.42
Baseline 2	64.53	63.45	71.85	32.73	28.33	34.24
Baseline 3	67.48	67.29	66.34	35.69	37.29	29.68
Baseline	66.46	68.52	71.63	35.10	35.37	33.45
Extension 1.1	70.47 ↑	70.42 ↑	72.57 ↑	40.05 ↑	35.96 ↑	37.24 ↑
Extension 1.2	64.11 ↓	67.51 ↓	70.74 ↓	33.17 ↓	34.11 ↓	35.34 ↑
Extension 1.3	66.18 ↓	71.88 ↑	68.86 ↓	35.99 ↑	33.59 ↓	38.68 ↑
Extension 2.1	71.23 ↑	73.77 ↑	68.90 ↓	39.66 ↑	42.57 ↑	37.42 ↑
Extension 2.2	66.21 ↓	77.55 ↑	70.56 ↓	35.86 ↑	38.88 ↑	39.29 ↑
Extension 2.3	67.57 ↑	69.75 ↑	72.62 ↑	37.36 ↑	38.05 ↑	37.44 ↑
Extension 3.1	65.69 ↓	72.73 ↑	69.73 ↓	34.25 ↓	39.25 ↑	34.02 ↑
Extension 3.2	66.69 ↑	68.20 ↓	69.37 ↓	35.60 ↑	35.56 ↑	33.80 ↑
Extension 3.3	65.19 ↓	68.39 ↓	64.31 ↓	34.62 ↓	34.83 ↓	28.27 ↓
Extension 4.1	67.57 ↑	72.65 ↑	74.63 ↑	36.94 ↑	38.80 ↑	38.61 ↑
Extension 4.2	69.39 ↑	77.85 ↑	66.19 ↓	39.88 ↑	44.71 ↑	37.01 ↑
Extension 4.3	68.11 ↑	76.11 ↑	68.01 ↓	37.74 ↑	37.80 ↑	38.14 ↑

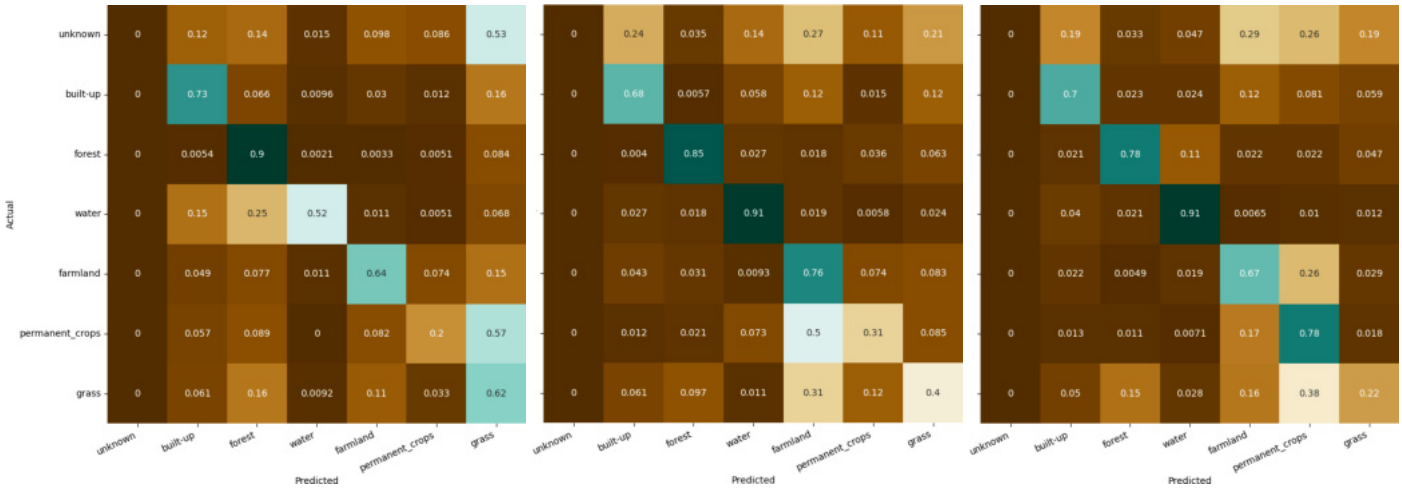
TABLE A.2: OA and IoU values at step 13 of all model runs.

Model Run	Train OA [%]	Val OA [%]	Train IoU [%]	Val IoU [%]
Baseline 1	60.97	52.55	30.41	23.83
Baseline 2	63.31	60.84	32.73	26.71
Baseline 3	61.66	58.34	31.98	31.23
Baseline	61.98	57.24	31.71	27.26
Extension 1.1	66.82 ↑	64.18 ↑	35.60 ↑	31.11 ↑
Extension 1.2	62.55 ↑	61.07 ↑	31.79 ↑	30.75 ↑
Extension 1.3	61.98	69.07 ↑	31.77 ↑	30.93 ↑
Extension 2.1	66.21 ↑	68.71 ↑	35.86 ↑	34.90 ↑
Extension 2.2	65.97 ↑	63.75 ↑	35.41 ↑	33.71 ↑
Extension 2.3	65.32 ↑	65.76 ↑	35.51 ↑	37.36 ↑
Extension 3.1	62.75 ↑	64.06 ↑	33.74 ↑	36.72 ↑
Extension 3.2	64.65 ↑	62.33 ↑	34.18 ↑	32.49 ↑
Extension 3.3 (12)	64.62 ↑	62.50 ↑	33.08 ↑	31.16 ↑
Extension 4.1	65.94 ↑	67.49 ↑	35.50 ↑	35.08 ↑
Extension 4.2	66.35 ↑	66.19 ↑	36.28 ↑	37.53 ↑
Extension 4.3	64.61 ↑	62.35 ↑	35.14 ↑	30.74 ↑

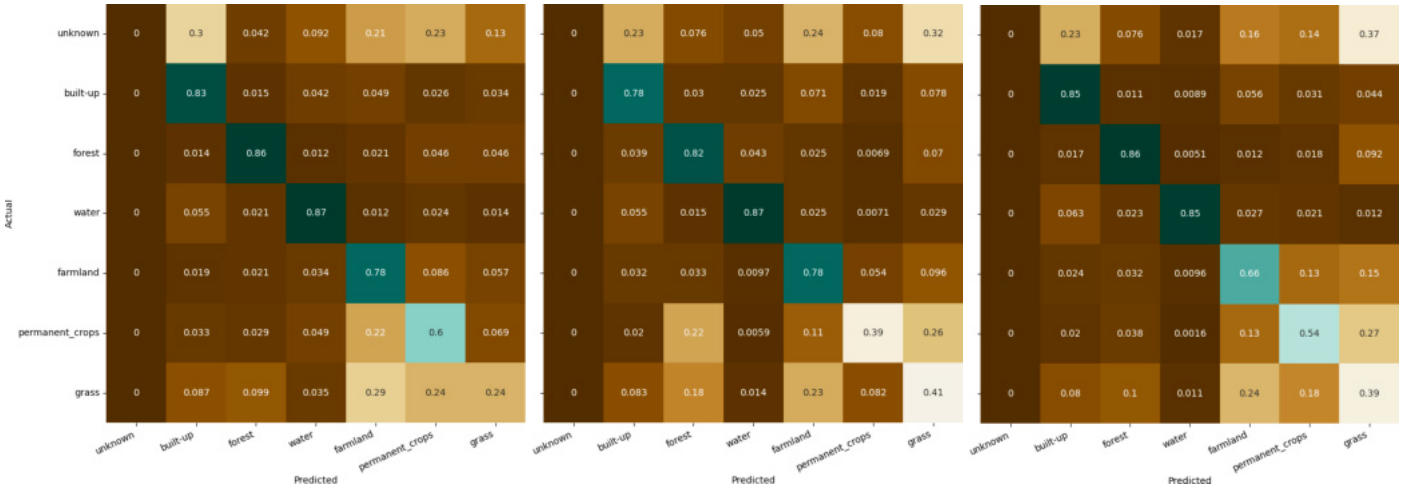
TABLE A.3: CA values of the test phases of all model runs. The according confusion matrices can be found in appendix section A.5.2.

Model Run	Built-up	Forest	Water	Farmland	Permanent Crops	Grass
Baseline 1	70	78	91	67	78	22
Baseline 2	68	85	91	76	31	40
Baseline 3	73	90	52	64	20	62
Baseline	70	84	78	69	43	41
Extension 1.1	83 ↑	86 ↑	87 ↑	78 ↑	60 ↑	24 ↓
Extension 1.2	78 ↑	82 ↓	87 ↑	78 ↑	39 ↓	41 ↓
Extension 1.3	85 ↑	86 ↑	85 ↑	66 ↓	54 ↑	39 ↓
Extension 2.1	90 ↑	84	92 ↑	63 ↓	47 ↑	51 ↑
Extension 2.2	87 ↑	84	93 ↑	68 ↓	66 ↑	27 ↓
Extension 2.3	81 ↑	84	90 ↑	74 ↑	57 ↑	40 ↓
Extension 3.1	88 ↑	88 ↑	91 ↑	55 ↓	70 ↑	16 ↓
Extension 3.2	81 ↑	90 ↑	89 ↑	57 ↓	44 ↑	26 ↓
Extension 3.3	68 ↓	83 ↓	72 ↓	59 ↓	46 ↑	19 ↓
Extension 4.1	87 ↑	87 ↑	91 ↑	79 ↑	46 ↑	39 ↓
Extension 4.2	87 ↑	84	88 ↑	48 ↓	87 ↑	35 ↓
Extension 4.3	89 ↑	77 ↓	93 ↑	66 ↓	68 ↑	38 ↓

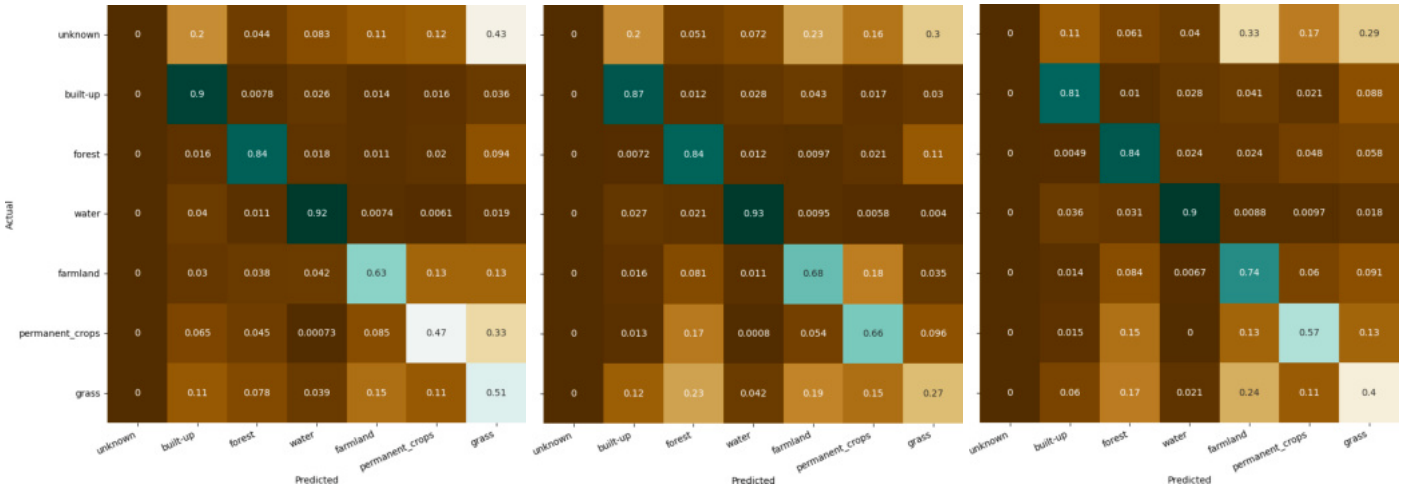
A.5.2 Confusion Matrices of All Model Runs



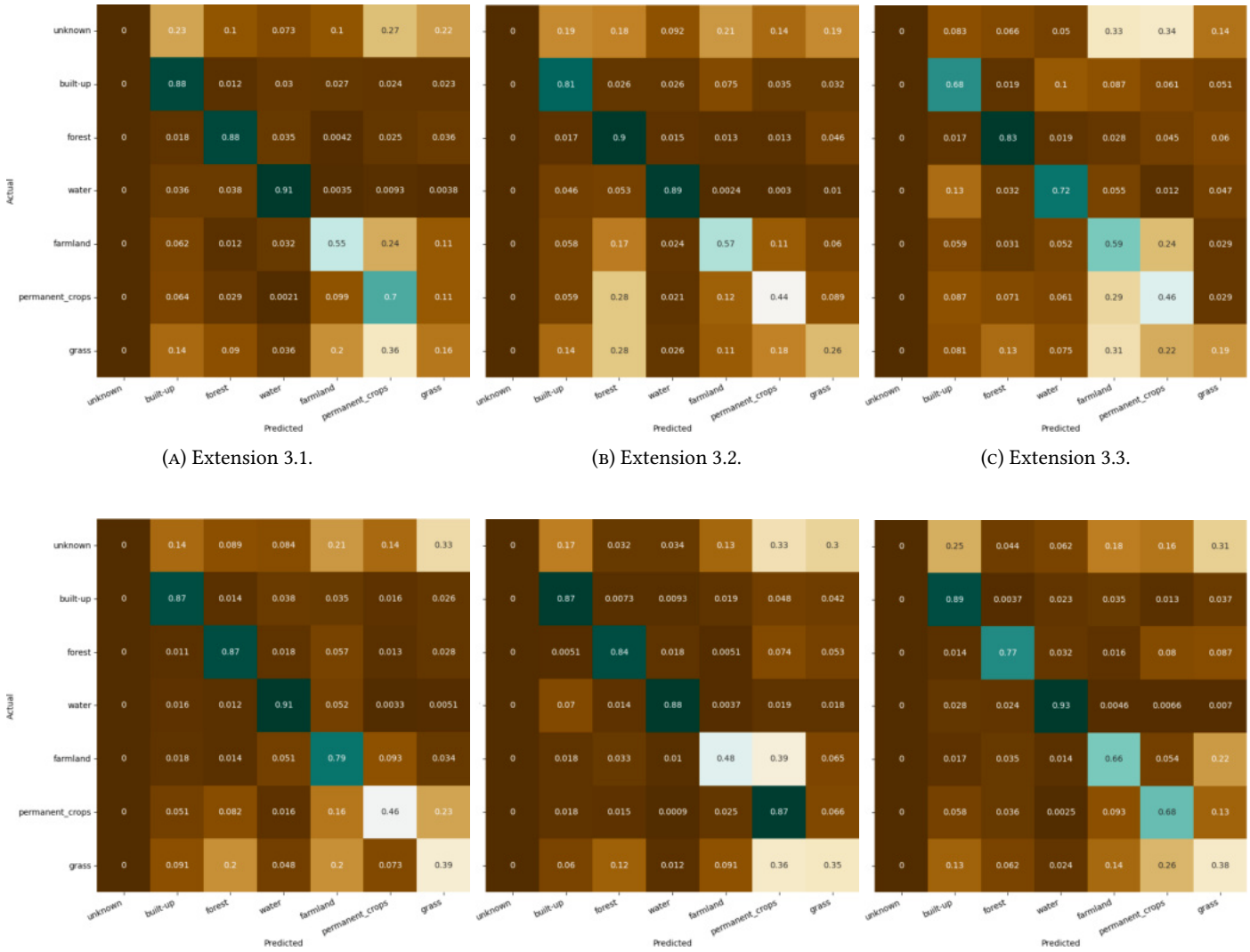
(A) Baseline 1. (B) Baseline 2. (C) Baseline 3.



(D) Extension 1.1. (E) Extension 1.2. (F) Extension 1.3.



(G) Extension 2.1. (H) Extension 2.2. (I) Extension 2.3.

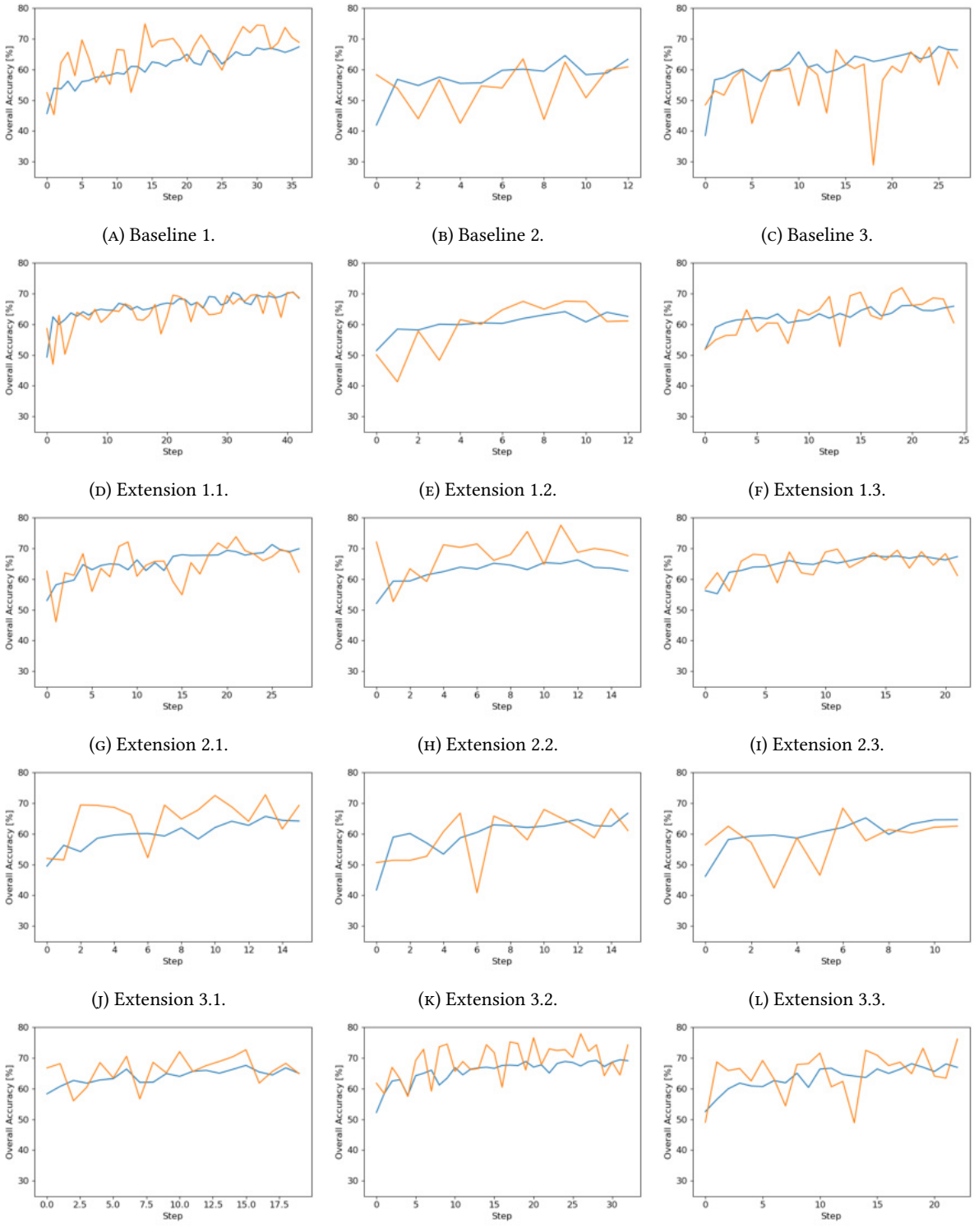


(A) Extension 3.1. (B) Extension 3.2. (C) Extension 3.3.

(D) Extension 4.1. (E) Extension 4.2. (F) Extension 4.3.

FIGURE A.19: Confusion matrices of all model runs for the test phase.

A.5.3 Training vs. Validation Overall Accuracy



(A) Baseline 1. (B) Baseline 2. (C) Baseline 3.

(D) Extension 1.1. (E) Extension 1.2. (F) Extension 1.3.

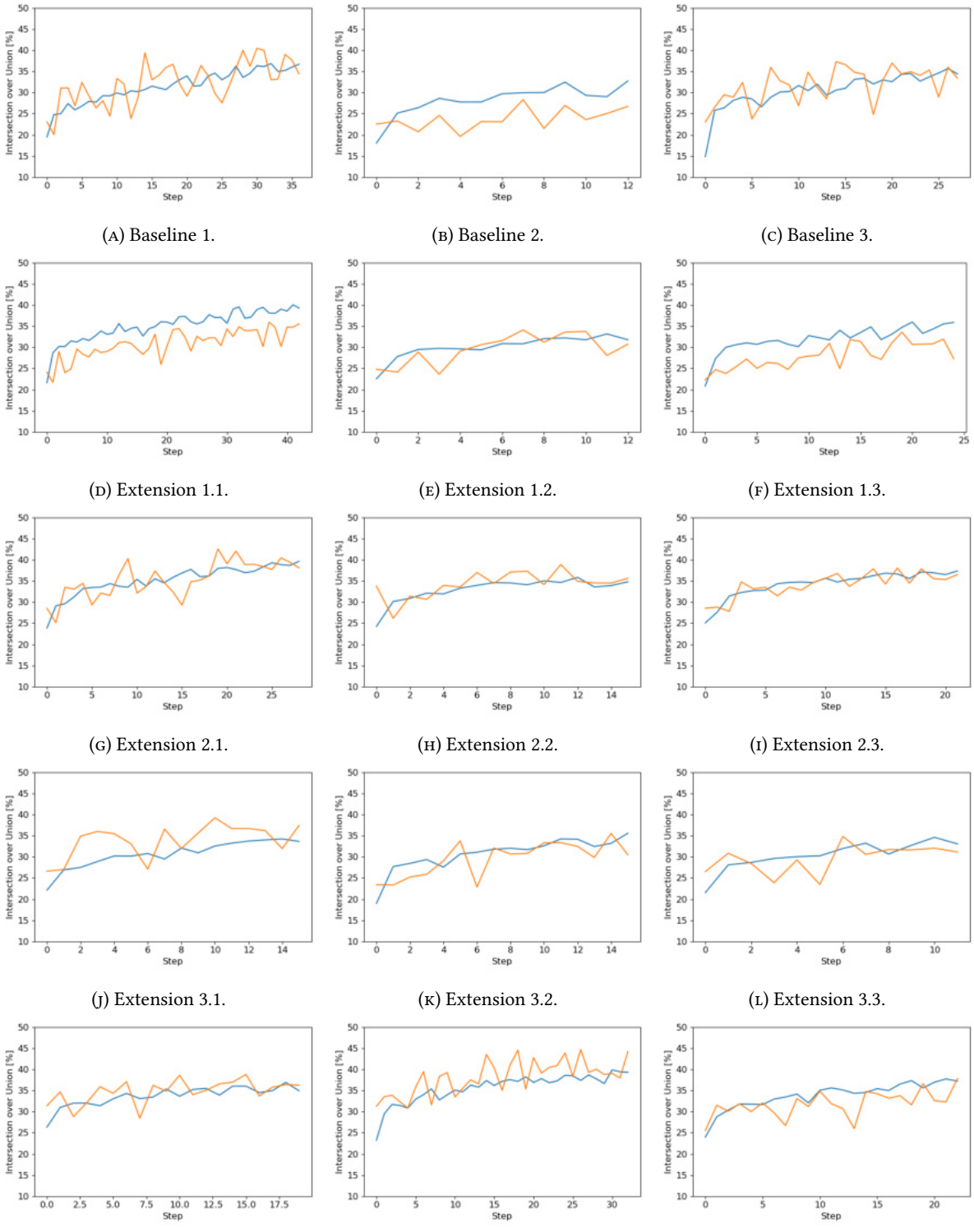
(G) Extension 2.1. (H) Extension 2.2. (I) Extension 2.3.

(J) Extension 3.1. (K) Extension 3.2. (L) Extension 3.3.

(M) Extension 4.1. (N) Extension 4.2. (O) Extension 4.3.

FIGURE A.20: OA of the training (blue) and validation (orange) phase for each model run. The number of epochs is not equalized.

A.5.4 Training vs. Validation Intersection over Union



(A) Baseline 1. (B) Baseline 2. (C) Baseline 3.

(D) Extension 1.1. (E) Extension 1.2. (F) Extension 1.3.

(G) Extension 2.1. (H) Extension 2.2. (I) Extension 2.3.

(J) Extension 3.1. (K) Extension 3.2. (L) Extension 3.3.

(M) Extension 4.1. (N) Extension 4.2. (O) Extension 4.3.

FIGURE A.21: IoU of the training (blue) and validation (orange) phase for each model run. The number of epochs is not equalized.

A.5.5 Hardware Utilization of All Model Runs

TABLE A.4: Averaged and maximal utilization of GPU and CPU of all model runs. Ø denotes the average value, and Max the maximum value.

Model Run	Channels	GPU Ø [%]	GPU Max [%]	CPU Ø [%]	CPU Max [%]
Baseline 1	5	23.75	68.00	37.18	74.80
Baseline 2	5	15.54	58.00	25.32	81.60
Baseline 3	5	18.92	67.00	34.39	79.40
Baseline	5	19.4	64.3	32.3	78.6
Extension 1.1	6	16.72 ↓	68.00 ↑	33.91 ↑	69.60 ↓
Extension 1.2	6	11.53 ↓	58.00 ↓	24.96 ↓	74.60 ↓
Extension 1.3	6	14.87 ↓	63.00 ↓	30.71 ↓	70.20 ↓
Extension 2.1	16	10.51 ↓	65.00 ↑	26.42 ↓	59.90 ↓
Extension 2.2	16	10.42 ↓	65.00 ↑	24.33 ↓	66.80 ↓
Extension 2.3	16	9.30 ↓	69.00 ↑	26.39 ↓	66.70 ↓
Extension 3.1	13	12.65 ↓	75.00 ↑	24.11 ↓	67.20 ↓
Extension 3.2	13	9.84 ↓	66.00 ↑	24.72 ↓	62.70 ↓
Extension 3.3	13	10.21 ↓	52.00 ↑	23.38 ↓	61.70 ↓
Extension 4.1	24	7.30 ↓	61.00 ↓	24.57 ↓	73.90 ↓
Extension 4.2	24	6.54 ↓	65.00 ↑	25.08 ↓	71.40 ↓
Extension 4.3	24	8.13 ↓	63.00 ↓	24.31 ↓	70.20 ↓

TABLE A.5: Averaged and maximal utilization of RAM and VRAM of all model runs. Ø denotes the average value, and Max the maximum value.

Model Run	Channels	RAM Ø [GB]	RAM Max [GB]	VRAM Ø [GB]	VRAM Max [GB]
Baseline 1	5	9.60	14.27	2.53	3.28
Baseline 2	5	7.08	13.92	2.00	3.29
Baseline 3	5	8.79	13.71	2.41	3.28
Baseline	5	8.49	13.97	2.31	3.28
Extension 1.1	6	13.57 ↑	18.51 ↑	2.69 ↑	3.28
Extension 1.2	6	9.87 ↑	18.60 ↑	2.12 ↓	3.28
Extension 1.3	6	11.96 ↑	18.79 ↑	2.44 ↑	3.28
Extension 2.1	16	15.27 ↑	22.60 ↑	3.14 ↑	3.67 ↑
Extension 2.2	16	13.95 ↑	24.69 ↑	2.86 ↑	3.65 ↑
Extension 2.3	16	14.78 ↑	25.49 ↑	3.02 ↑	3.65 ↑
Extension 3.1	13	12.83 ↑	21.40 ↑	2.65 ↑	3.49 ↑
Extension 3.2	13	12.52 ↑	20.23 ↑	2.66 ↑	3.50 ↑
Extension 3.3	13	11.61 ↑	19.99 ↑	2.54 ↑	3.50 ↑
Extension 4.1	24	16.98 ↑	28.25 ↑	3.26 ↑	3.90 ↑
Extension 4.2	24	18.07 ↑	28.72 ↑	3.47 ↑	3.90 ↑
Extension 4.3	24	17.77 ↑	27.94 ↑	3.32 ↑	3.91 ↑