A Domain Decomposition-based CNN Architecture for High-Resolution Image Segmentation

Master Thesis

C. Verburg

Delft University of Technology

TU Delft

A Domain Decomposition-based CNN Architecture for High-Resolution Image Segmentation

by

C. Verburg

Daily Supervisor:	A. Heinlein	Numerical Analysis
Daily Co-Supervisor:	E.C. Cyr	Sandia National Laboratories, US
Responsible Supervisor:	C. Vuik	Numerical Analysis
Supervisor:	D. Lahaye	Mathematical Physics

Project Duration: July, 2023 - February, 2024
Faculty of Electrical Engineering, Mathematics and Computer Science, Delft

Preface

This thesis project concludes my master's degree in Applied Mathematics at Delft University of Technology. Two years ago, fresh from completing my bachelor's degree, my interests were mostly in scientific computing, particularly in numerical methods and high-performance computing algorithms. I would not have anticipated then that my thesis project would focus on a machine-learning-related topic. However, this new direction has excited me greatly, and it was truly enjoyable and interesting to delve further into this area.

This thesis project has been made possible by the support of many different persons, whom I would like to thank personally here. Firstly, I want to thank my daily supervisors, Alexander and Eric. Without your valuable, prompt, and insightful thoughts and feedback, this thesis would not have been possible. Alexander, your interest in the results, enthusiasm for the project, and your initiative in inviting me to various scientific conferences, have been inspiring and pushed me beyond my limits. Eric, as my second daily supervisor, I want to extend my gratitude for your incisive questions, helpful insights, and constructive feedback. I would also like to thank Kees Vuik and Domenico Lahaye for being members of my thesis committee and providing their thoughts on my work during the literature review presentation.

To my fellow math students with whom I have shared many coffee and lunch sessions, thank you for making the last few months so enjoyable. A special thanks goes to my housemates from the Lindenburgh, who have likely heard more about implementation and neural network training issues than they ever will again. Lastly, I am very grateful to my friends and family, on whom I could always count during this, at some points, stressful period of my life.

C. Verburg
Delft, April 2024

Abstract

This thesis addresses the challenge of segmenting ultra-high-resolution images. Limitations of current approaches to segment these are that either detailed spatial contextual information is lost or many redundant computations are necessary. To overcome these issues, we propose a novel approach combining the U-Net architecture with domain decomposition strategies to balance incorporating spatial context and maintaining computational efficiency. Our proposed method partitions input images into non-overlapping patches, each processed independently on a separate device. A communication network facilitates the exchange of information between patches, enhancing the model's understanding of the spatial context.

Through theoretical analysis and practical experimentation on device memory usage during training, we demonstrate that our approach incurs minimal additional memory overhead for inter-device communication. Evaluation on synthetic and realistic datasets, including the Inria Aerial Image and DeepGlobe Satellite Segmentation datasets, demonstrates the effectiveness of our approach. Our model achieves competitive performance compared to the baseline U-Net model, with consistent class predictions around boundaries. Visualization of feature maps highlights the role of the communication network in transferring contextual information. Furthermore, it is shown that our approach remains scalable even when trained on limited subdomains.

In conclusion, our proposed model offers an intuitive solution for segmenting ultra-high-resolution images by effectively incorporating spatial context. Future research could explore variations of our model, such as overlapping subdomains or communication on different levels of the U-Net, to further enhance boundary consistency and information transfer.

Contents

List of Tables

1 Introduction

In recent years, the vast majority of deep learning models in computer vision have focused on low-resolution images, typically 256×256 pixels or smaller. However, the advancement of high-resolution image datasets brings forth new challenges related to the memory constraints of a single graphics processing unit (GPU), especially for memory-intensive tasks and deep learning models.

To illustrate this, consider, for example, CT scans with sub-millimeter resolution, resulting in voxel image data, typically sized at $512 \times 512 \times 512$ voxels. Even with half-precision floating-point numbers and a modest batch size of 8, processing such images with a 1-layer convolutional neural network with 64 filters demands over 137GB of GPU/TPU memory, as highlighted in [28]. Dealing with such high-resolution inputs using conventional strategies like downsampling or patch cropping often leads to the loss of detailed information or spatial contextual information [9, 28, 66].

A form of deep learning for image vision where this memory constraint is especially relevant is the field of *(semantic) image segmentation.* Semantic segmentation is the computer vision task of classifying the pixels in the input into distinct, non-overlapping semantic categories. Examples include identifying organs like lungs, heart, and stomach in biomedical images or differentiating land, water, and buildings in satellite images. Ultra-high-resolution image segmentation holds significance in diverse fields such as self-driving vehicles [42], metallic surface defect detection [63], and computer-aided medical diagnosis [3]. While deep convolutional neural networks (CNNs) have achieved remarkable success in segmentation [41], many of these models are designed to process full-resolution input images, posing memory challenges for high-resolution image model training and inference.

To address this issue, we approach this problem from a perspective inspired by the mathematical field of Domain Decomposition Methods (DDMs). Traditionally, domain decomposition methods [61] are numerical methods employed to solve partial differential equations. These methods decompose a global problem on a large domain into smaller subproblems on sub-domains, where the local subproblems are addressed and solved for a large part independently. Domain decomposition methods provide ways to decompose the problem and *couple* the different subproblems using different strategies, such as overlapping subdomains, transmission conditions, or introducing a coarse problem. Domain decomposition is known as a convenient paradigm for solving PDEs on parallel computers; the common factor in all domain decomposition methods is that they rely on the fact that each processor can do the most significant part of the work independently of the other processors.

This parallel nature inherent to many DDMs is very relevant for the memory-distribution problem of deep learning models for images. Consequently, combining insights, strategies, and approaches from domain decomposition with the design and training procedure of neural networks appears promising.

In this thesis, we present a novel architecture that is based on the U-Net [55], an image segmentation network (for more details, see Chapter 3). The U-Net has been applied in many (biomedical) segmentation tasks, from CT scans and MRI to X-rays and microscopy. Although the U-Net and U-Net-based architectures have been very successful for semantic segmentation tasks, a remaining constraint for all variants of the U-Net is the large memory requirements, making the model unsuitable for high-resolution applications with limited computational devices [4]. Our novel approach to semantic image segmentation tasks integrates the established U-Net architecture with a divide-and-conquer spatial domain decomposition strategy to use the combined memory of multiple devices for successful high-resolution segmentation. Unlike previous U-Net parallelization methods such as [25, 28, 57] that involve communicating margins before each convolution or entail redundant computations, our approach strategically incorporates communication only at specific layers within the U-Net. This minimizes communication overhead while preserving essential contextual information.

This thesis is organized as follows. In Chapter 2, we formally introduce the task of image segmentation and provide a theoretical background on (convolutional) neural networks. In Chapter 3, the U-Net is discussed in more detail, and we consider existing memory parallelization strategies of the U-Net with their up- and downsides. Next, we discuss existing approaches using ideas from the field of domain decomposition to speed up and parallelize convolutional neural networks in Chapter 4. In Chapter 5, the proposed model is introduced, as well as a detailed description of the datasets used for testing. Some preliminary founding results are shown in Chapter 6. These preliminary results include an analysis of the encoder network used and an analysis of a baseline U-Net model compared to our approach. Finally, We test and evaluate our proposed network model and the related training strategy in Chapter 7. We conclude and present possible future research directions in Chapter 8. The code repository with the implementations used for this project can be found at `https://github.com/corne00/HiRes-Seg-CNN`.

2 Image Segmentation and Neural Networks

In this chapter, we start by describing the task of semantic image segmentation. We introduce neural networks and more specifically CNNs. This type of network is one of the major achievements in image vision techniques in the field of deep learning. Computer vision using CNNs can be applied in a broad range of fields, such as face recognition, image enhancement, content generation, autonomous vehicles, and intelligent medical treatment. To better understand state-of-the-art techniques applied in CNNs, in this chapter, we first present some fundamentals of (convolutional) neural networks, then we introduce the standard architectural components of CNNs and discuss training and optimization procedures.

2.1. Image Segmentation

The machine learning task considered in this thesis is image segmentation, which has a broad range of applications, such as self-driving vehicles [42], metallic surface defect detection [63], and computer-aided medical diagnosis [3]. A formal definition of semantic segmentation is given in Definition 2.1 [58].

Definition 2.1 (Semantic Segmentation). *Semantic (image) segmentation is the process of assigning each pixel of the received image to one of the predefined classes.*

Compared to image classification, where entire images are assigned one or more class labels, and object detection, which identifies both the class and location of objects within an image, semantic segmentation poses greater challenges. It requires accurately distinguishing object regions from background regions, necessitating the integration of low-level and high-level features. This aspect forms a particularly demanding aspect of semantic segmentation. [23]. An example image and corresponding segmented mask is shown in

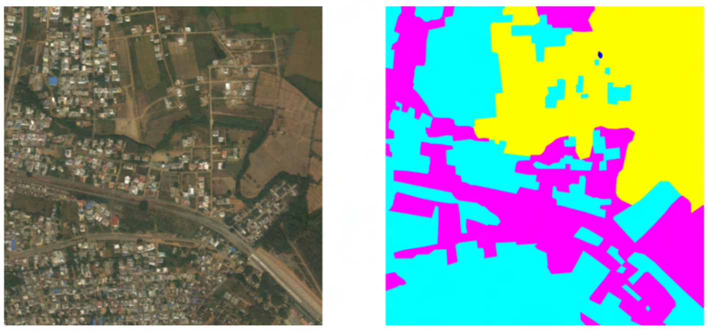


Figure 2.1: An example of a segmentation task for a satellite photo. The segmentation classes correspond to different land types (urban land, agricultural land, rangeland, forest land, water, barren land, and unknown). The figure is constructed using samples from the DeepGlobe dataset [14]

To measure and compare the performance of different approaches for image segmentation, several metrics have been used that take into account the number of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) predictions for each class. We describe these metrics, assuming that there are in total C classes, including the background. The total number of pixels in the image is denoted as P. The number of pixels in class i predicted as class j is defined as P_{ij}, with $i, j \in \{1, \dots, C\}$.

Definition 2.2 (Pixel Accuracy (PA)). *The pixel accuracy denotes the ratio between the number of correctly classified pixels and the total number of pixels (equivalent to (TP / (TP+FN)). It is given in Equation* (2.1)*:*

$$PA = \frac{\sum_{i=1}^{i=C} P_{ii}}{\sum_{i=1}^{C} \sum_{j=1}^{C} P_{ij}} \tag{2.1}$$

Definition 2.3 (Class-wise Intersection over Union (IoU)). *The class-wise IoU is the ratio between the intersection and the union of correctly classified pixels and wrongly classified pixels for a specific class (equivalent to (TP/(TP+FP+FN)), which is computed as in Equation* (2.2)*:*

$$IoU_i = \frac{P_{ii}}{\sum_{j=0}^{k} P_{ij} + \sum_{j=0}^{k} P_{ji} - P_{ii}}, \qquad i = 1, \dots, C \tag{2.2}$$

Definition 2.4 (Mean Intersection over Union (mIoU)). *The mIoU denotes the average of the class-wise IoU. Using Definition 2.3, the mean IoU score (mIoU) can be computed based on Equation* (2.3)*:*

$$mIoU = \frac{1}{C} \sum_{i=1}^{C} IoU_i \tag{2.3}$$

Finally, two other valuable metrics for image segmentation are precision and recall. Though subtle, their distinction is crucial, as both metrics describe another aspect of the segmentation results. *Recall* measures the completeness of predictions (i.e., how much of a class is accurately predicted by the segmentation model) relative to the ground truth, while *precision* describes the accuracy of the predictions made (which part of the predictions is predicted in the correct class).

Definition 2.5 (Recall). *The Recall metric, also known as sensitivity or true positive rate, represents the number of correctly classified pixels divided by the total number of true pixels in a class. It can be computed separately for each class and is defined as shown in Equation* (2.4)*:*

$$Recall_i = \frac{P_{ii}}{\sum_{j=1}^{C} P_{ij}}, \qquad i = 1, \dots, C, \tag{2.4}$$

Definition 2.6 (Precision). *The precision metric, also known as positive predictive value, denotes the number of correctly classified pixels divided by the total number of predicted pixels in a class. It can be computed separately for each class and is defined as shown in Equation* (2.5)*:*

$$Precision_i = \frac{P_{ii}}{\sum_{j=1}^{C} P_{ji}}, \qquad i = 1, \dots, C, \tag{2.5}$$

Note that all these metrics measure different aspects of the quality of the prediction. The choice of a specific segmentation method often means a trade-off between recall and precision. Therefore, it is essential to consider both when evaluating the results.

2.2. Fundamentals of Neural Networks

Before we introduce CNNs, we first briefly introduce the more general concept of (feedforward) neural networks. Neural networks are inspired by the human brain and initially started as an attempt to model the neurons in the human brain [46]. Intuitively, a neural network can be thought of as a collection of neurons, organized in layers, where the neurons of different subsequent layers are interconnected in some way. Each connection has a certain weight, indicating how important the information flowing through that connection is. Only if a neuron receives enough stimulating information from neurons in previous layers, it will activate and send out information itself.

To define neural networks more formally, we consider a supervised learning task, i.e., the approximation of a function $F : \mathbb{R}^n \to \mathbb{R}^m$ mapping given input $I = \{x_1, \dots, x_N\}$ to corresponding output data $O = \{y_1, \dots, y_N\}$, with $x_i \in \mathbb{R}^n$ and $y_i \in \mathbb{R}^m$, for $i = 1, \dots, N$. In other words: we want the function F to satisfy the relation

$$F(x_i) = y_i, \qquad i = 1, \dots, N, \tag{2.6}$$

or at least a function that is very close to this function, concerning some error norm, for example the L_2-cost function:

$$L_2(x_i, y_i; F) = \sum_{i=1}^{N} (F(x_i) - y_i)^2) \tag{2.7}$$

NNs are designed to perform this task of function approximation by taking advantage of some useful linear algebra properties. To understand this mechanism, we first consider the following equation.

$$P(x) = A\alpha(Bx + c). \tag{2.8}$$

In this equation, x, c, and $P(x)$ are vectors and A and B are matrices. The *activation function* $\alpha : \mathbb{R} \to \mathbb{R}$ is a scalar function applied component-wise to the vector $Bx + c$. Furthermore, we have that $x \in \mathbb{R}^n, c \in \mathbb{R}^k, P(x) \in \mathbb{R}^m, A \in \mathbb{R}^{m \times k}, B \in \mathbb{R}^{k \times n}$, with k a freely chosen parameter and m and n the output and input dimensions, respectively. Note that the function α is essential for the model to be nonlinear since all other operations in this equation are linear. A model that only includes linear functions is only able to make linear transformations, making it incapable of handling more complex, nonlinear real-world data. Nonlinear activation functions allow the model to approximate these more complex mappings as well.

If we want to approximate the function F, the most general assumption that we can make is that this function is nonlinear. Therefore, to obtain nonlinearity in our model, it is mostly the most suitable option to choose for α a nonlinear function such as, for example, the hyperbolic tangent function, or the ReLU function.

It has been proven that neural networks that have the form of Equation 2.8 are universal function approximators. For example, for sigmoid activations, in [27] we find the proof of the following theorem.

Theorem 2.1 (Universal Approximation Theorem (Sigmoid)). *Let I_n denote the n-dimensional unit cube $[0, 1]^n$, and let α be the sigmoid activation function. Then, finite sums of the form*

$$P(x) = \sum_{i=1}^{M} a_i \alpha(Bx + c) \tag{2.9}$$

are dense in the space of continuous functions $\mathcal{C}(I_n)$. In other words, given any $f \in \mathcal{C}(I_n)$ and $\epsilon > 0$, there is a sum $P(x)$ of the above form for which holds that

$$|P(x) - f(x)| < \epsilon, \quad \forall x \in I_n. \tag{2.10}$$

Intuitively, this universal approximation states that, given that the number M is large enough, a neural network with layers of the form Equation (2.8) can approximate any continuous function to arbitrary accuracy. This theorem has been extended and proven for a more general class of activation functions in the paper with the title *Multilayer feedforward networks are universal approximators* [27]. This shows us that it is worthwhile to look for approximations of this form and brings us to the definition of general neural networks. We define $x \in \mathbb{R}^n$ as the input of our neural network. The number of hidden layers, intermediate layers between the input and output layers of our network, is denoted by L. Note that x can represent many different data points, varying from images to voxel images or time series as well as unstructured data. Then, the neural network is described by the set of equations

$$\begin{aligned} h_1 &= \alpha_1(W_1 x + b_1), \\ h_{i+1} &= \alpha_{i+1}(W_{i+1} h_i + b_{i+1}), \quad \text{for } i = 1, \dots, L-1, \\ y &= W_{L+1} h_L \end{aligned} \tag{2.11}$$

The vector $y \in R^m$ is the output of the neural network, while the other vectors $h_{i+1} \in \mathbb{R}^{n_i}$ with $i = 0, \ldots, L-1$ are the states of the neurons in the L hidden layers of the network. Each layer is definded by a weight matrix $W_i \in \mathbb{R}^{n_i \times n_{i-1}}$ that contains the *weights* of the particular layer, and a bias vector $b_i \in R^{n_i}$. Also, note that the activation function α_i can be chosen to be a different function for different layers.

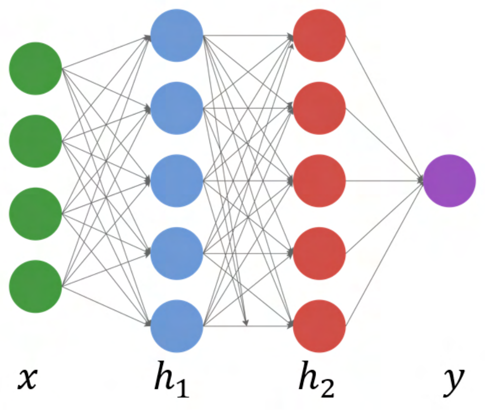


Figure 2.2: Visualization of a very simplistic neural network. The green circles denote the input vector x. The network consists of two hidden layers h_1 and h_2 (the red and blue circles) and gives as output one value y, denoted by the purple circle. Note that the arrows between the different neurons represent the weights stored in the weight matrices W_i: each arrow represents another weight $w \in W_i$.

If the weight matrices $W_i, i = 1, \ldots, L+1$ are dense, the neural network defined above is a *dense* network. However, in many architectures, to limit the number of computations, these matrices are sparse. The number of layers in a neural network is also denoted as its **depth**, and the number of neurons n_i in a layer is often referred to as the **width** of a specific layer. The term *deep learning* is derived from networks with many layers (generally 4 or more layers). An example of a very simple dense neural network is shown in Figure 2.2.

2.2.1. Activation functions

In this subsection, we explore some important activation functions, based on the survey on activation functions of Dubey et al. [19]. Note that all activation functions operate element-wise on their argument: therefore, all activation functions are defined as scalar functions. The following activation functions are often encountered in machine-learning tasks.

- **The linear activation function** $\alpha(x) = x$ is one of the simplest activation functions. However, this activation function is normally not used since it does not introduce non-linearity in the model, leading to a network that only can estimate simple linear problems. However, this activation function can be used in combination with nonlinear activation functions.
- **The sigmoid activation function**, defined as $\alpha(x) = \frac{1}{1+\exp(-x)}$ is one of the classical activation functions. This function results in a value in the interval $(0, 1)$. A disadvantage of this activation function is that the function is saturated for higher and lower inputs, leading to a low gradient, which results in very slow learning. This problem is also known as the *vanishing gradient problem.*
- **The tanh activation function** shares a lot of properties with the sigmoid function, but it shows the zero-centric property, i.e. the mean activation value is zero. This function is defined as $\alpha(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$ and has the output range $[-1, 1]$.
- **The Rectified Linear Unit Activation Functions (ReLU)**, defined as $\alpha(x) = \max(0, x)$ is the most widely used loss function, providing a solution for the vanishing gradient problem of the sigmoid function. Its simplicity also makes it fast and easy to implement. The disadvantages of this loss function are that negative values are not utilized, which can lead to *deactivated* neurons, and that the output is unbounded, which may lead to exploding gradients. Furthermore, the gradient is zero for negative inputs.

- **The leaky ReLU activation function** was proposed to use the advantages of the ReLU function, but also use the negative values by giving them a small linear weight. This function is defined as $\alpha(x) = \beta \min(0, x) + \max(0, x)$, where β is either set by the user or included in the model as a learnable parameter.
- **The Exponential Linear Unit (ELU)** $\alpha(x) = x, x > 0$ and $\alpha(x) = \beta(\exp(x) - 1), x \leq 0$ can be seen as a shifted ReLU function with a smoother, such that it is everywhere continuously differentiable.
- **The Swish activation function** is defined as $\alpha(x) = xf(\beta x)$, where $f(x)$ denotes the sigmoid activation function and $\beta \in \mathbb{R}$ is a learnable parameter. This activation function can help in solving the vanishing gradient problem since it returns larger gradients during back-propagation compared to other activation functions like the Sigmoid or Tanh functions.
- **The Softmax activation function** is often used in the output layer of classification problems: it is designed to convert a vector $x \in \mathbb{R}^n$ to a certain probabilistic distribution with n classes. Therefore, its output is a vector of n probabilities, where the components are defined as $(\alpha(x))_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)}$. Note that all components in the vector $\alpha(x)$ will be in the range $(0, 1)$ and their sum is equal to 1. Larger input values will lead to a larger probability.

Figure 2.3 shows some of the discussed activation functions.

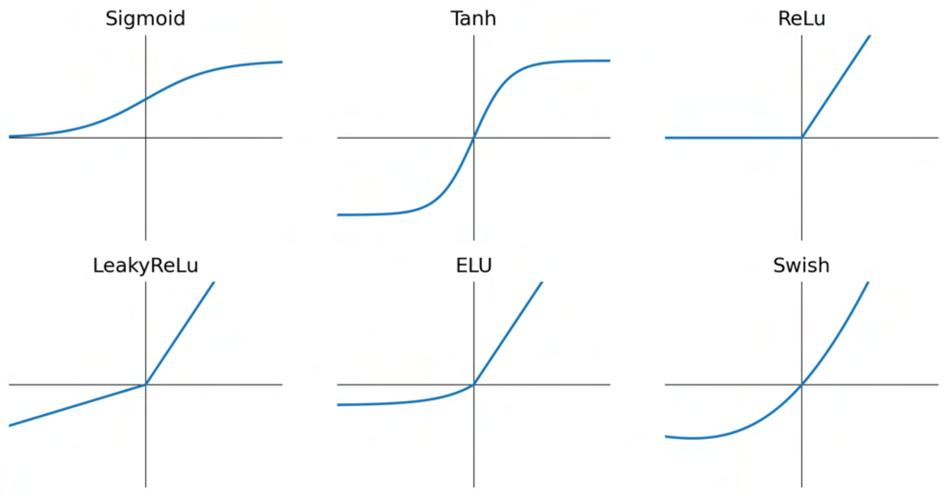


Figure 2.3: Some important activation functions. Note the similarities and differences between the ReLU, Leaky ReLU, and ELU activation functions: for $x > 0$ they are the same, but they are different for negative x. The first row shows the sigmoid, tanh, and yReLU respectively, whereas the second row shows the LeakyReLU, ELU, and Swish activation function respectively.

In this study, we make use of established activation functions to maintain consistency and reliability, employing ReLU for intermediate layers and Softmax for the output layer to ensure the model output is in the correct range. However, it's worth noting that adjusting activation functions can significantly enhance performance in terms of both speed and accuracy, as discussed in, among many other papers, [19, 24].

2.3. Training of Neural Networks

Given a neural network with the right parameters, this network will be able to be a good estimator of many non-linear functions. However, therefore it is crucial to find those parameters. In this section, we explore the training of neural networks. Suppose we have a neural network $\mathcal{NN}^{\alpha}_{W,b}$ that uses the same activation function α in each layer and is parameterized by weights $W = \{W_i\}_{i=1}^{L+1}$ and bias vectors $b = \{b_i\}_{i=1}^{L}$. Then, we compute the output of the i-th layer with layer input x, denoted by $F_i(x)$ with the following formula:

$$F_i(x) = \alpha(W_i x + b_i). \tag{2.12}$$

where W_i is the weight matrix and b_i the bias vector. But then, the output of the neural network is simply defined as

$$\mathcal{NN}^{\alpha}_{W,b}(x) = W_{L+1}F_L \circ \ldots \circ F_1(x). \tag{2.13}$$

i.e. a composition of functions. Now, suppose we have a training data-set with input data $I = \{x_1, \ldots, x_N\}$ and corresponding output data $O = \{y_1, \ldots, y_N\}$. We want to train the neural network to solve the optimization problem:

$$\min_{W,b} \sum_{i=1}^{N} \mathcal{L}(\mathcal{NN}^{\alpha}_{W,b}(x_i), y_i). \tag{2.14}$$

In this equation, the loss function $\mathcal{L}$ should penalize for deviations of the model output $\mathcal{NN}^{\alpha}_{W,b}(x_i)$ from the correct label y_i. Several options are possible and will be presented in the next subsection. For now, let the loss function be the mean squared error, defined as:

$$\mathcal{L}_{L2}((\mathcal{NN}^{\alpha}_{W,b}(x_i), y_i) = ||(\mathcal{NN}^{\alpha}_{W,b}(x_i) - y_i||_2^2 \tag{2.15}$$

To solve this minimization problem, several approaches can be used, mostly variants of the stochastic gradient descent (SGD) method or quasi-Newton methods or combinations of both. In this literature study, we limit ourselves to a discussion of the most common form of optimization: mini-batch optimization for SGD. Furthermore, we discuss the Adam optimization algorithm as this is further employed throughout this paper.

Mini-batch optimization is a variant of the stochastic gradient descent method (SGD). The SGD method selects at each gradient step one random term from the sum

$$\sum_{i=1}^{N} \mathcal{L}(\mathcal{NN}^{\alpha}_{W,b}(x_i), y_i) \tag{2.16}$$

to compute the gradient with respect to the weights W and the bias b in all the layers of the neural network,

$$\nabla_{W,b}\mathcal{L}(\mathcal{NN}^{\alpha}_{W,b}(x_i), y_i), \tag{2.17}$$

and then updates the weights matrices W and bias vectors b using this gradient:

$$W_{k+1} = W_k - \alpha\nabla_W\mathcal{L}(\mathcal{NN}^{\alpha}_{W,b}(x_i), y_i), \tag{2.18}$$
$$b_{k+1} = b_k - \alpha\nabla_b\mathcal{L}(\mathcal{NN}^{\alpha}_{W,b}(x_i), y_i), \tag{2.19}$$

where $\alpha > 0$ is the learning rate, a parameter determining how fast the weights and bias are updated.

In the next iteration, one of the remaining data points is chosen to perform the next gradient step. The number of iterations needed to go through all data points is denoted as one *epoch*. The advantage of this approach is that the computation of the gradients is much cheaper in a gradient step compared to the case where the gradient is computed for all samples, and furthermore, this approach is robust against getting stuck in local minima.

To combine the advantages of the classical gradient descent method and the SGD method, the mini-batch optimization approach partitions the data index set $\{0, \ldots, N\}$ into K disjoint subsets with size k. Then, in the j-th step of the mini-batch SGD, we use the gradient

$$\sum_{i \in B_j} \nabla_{W,b}\mathcal{L}(\mathcal{NN}^{\alpha}_{W,b}(x_i), y_i), \qquad j = 1, \ldots, K \tag{2.20}$$

to update the weights and bias vectors. Since the sets are disjoint and form a partition of the set of data indices, K iterations of mini-batch SGD correspond to one epoch. Note that a partition of only one set $K = 1$ corresponds to the classical gradient descent method, whereas a partition into sets with only one elements $K = N$ corresponds to the SGD method.

The Adam (ADAptive Moment) optimization algorithm is an algorithm that can be used as an alternative to the stochastic gradient descent procedure to update neural network weights. It was first introduced in [39]. Whereas classical SGD methods maintain a fixed, global learning rate α that is used for all weight updates, the Adam algorithm computes individual, adaptive learning rates for different parameters from estimates of the first moments (means) and second moments (the variance) of the gradients.

To describe the Adam algorithm mathematically, we let θ represent the parameters of the neural network, consisting of weights W and bias b in our case. The algorithm maintains two moving averages: the first moment m_t (mean) and the second moment v_t of the gradients (for all parameters). These are initialized as zero vectors at time step $t = 0$:

$$m_0 = 0, \ v_0 = 0. \tag{2.21}$$

At each iteration, the algorithm computes the gradient g_t (computed as in Equation (2.20) of the loss function concerning the parameters θ based on a mini-batch of training data. Then, it updates the moving averages:

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \tag{2.22}$$
$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \tag{2.23}$$

Here, β_1 and β_2 are decay rates (typically close to 1, the default initialization is recommended to $\beta_1 = 0.9, \beta_2 = 0.999$ in [39]), controlling the exponential decay of the moving averages.

Next, Adam computes bias-corrected estimates of the first and second moments:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.24}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{2.25}$$

Finally, it updates the parameters θ using the following rule:

$$\theta_t = \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{2.26}$$

Here, ϵ is a small constant (often on the order of 10^{-8}) to prevent division by zero.

The adaptive learning rates computed by Adam improve training efficiency by adjusting the step sizes based on the magnitudes of the gradients and the history of gradient updates. This adaptive behavior allows Adam to converge faster and potentially find better solutions compared to traditional SGD [39].

Forward and backward propagation are the crucial steps in machine learning. We briefly describe those processes here. For a more extensive discussion, we refer to [7, 12]. Forward propagation is the propagation of the input data through the neural network to compute the predicted output. This predicted output is used to compute the loss with respect to some loss metric, such as the L_2 cost function. Backward propagation, on the other hand, is the process of calculating gradients by propagating the computed loss backward from the output layer through the network to update the model's parameters (the weights W and biases b during training. Since the functions used in neural networks are mostly elementary compositions of linear and nonlinear functions, the gradients can be computed efficiently using the chain rule, product rule, and the property of linearity of derivatives - it is only an administratively challenging task to do so. Using *automatic differentiation*, the backpropagation can be done very efficiently and accurately. As the gradients of intermediate, hidden layers often depend on their output, the intermediate outputs of these hidden layers needs to be stored.

Data and batch normalization. For many datasets, it is beneficial to normalize the input data, since this helps to ensure that different input features contribute equally to the model's results [62]. Note that normalization is an invertible operation so that the outputs can be transformed back to their original range.

For the output of the different layers, it can also be beneficial to apply normalization. However, this can only happen during training, since the outputs of different layers are dependent on each other. This approach is called *batch-normalization*, and was first proposed in [31]. In this approach, the normalization takes place during the forward propagation. To compute the mean and value of a certain layer output, only the values of the batch itself are used during training. These batch mean and variance are used to update a global mean and variance, which is used during inference.

2.4. Convolutional Neural Networks

Having defined a general (feedforward) neural network, we now focus on a more specific kind of neural network that has become very popular in the field of image vision: the convolutional neural network [67]. As the term already says, this type of network uses convolutional operations which turn out to be very suitable for imaging tasks. To gain a better understanding of this type of network, we introduce in the following subsections the *convolutional layer*, *pooling layers*, and *downsampling layers*.

2.4.1. Convolutional layers

The idea of CNNs is inspired by visual perception: the human eye contains several receptors that respond to different features. The convolutional layer plays a vital role here. A *convolution* is a mathematical operation that involves the element-wise multiplication and summation of a filter (kernel) with local regions of the input data, formally defined as

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v] \cdot F[i-u, j-v]. \tag{2.27}$$

for a 2D image (an image with only one channel, for example gray-scale). Here, G denotes the feature map obtained by convolution, H and $F \in \mathbb{R}^{(2k+1)\times(2k+1)}$ denote the input image/feature map and the filter/kernel, respectively. $2k+1$ is the size of the convolutional filter/kernel.

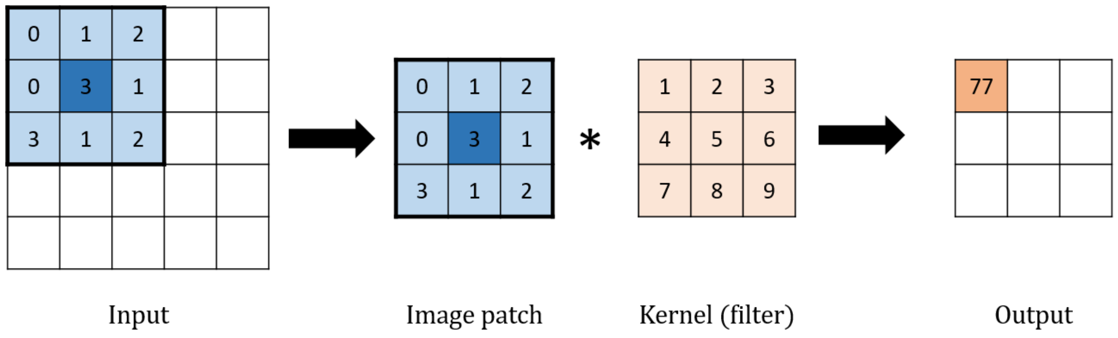


Figure 2.4: Visualization of the operations performed in the convolutional layer. The asterisk $*$ in this figure denotes a component-wise multiplication and summation operation. The stride parameter in this example is 1, the padding parameter is set to 0. Note that this leads to an output that has a smaller size than the original input.

This operation has proven to be successful in producing mappings of input data that capture relevant patterns, such as line segments, circles or other shapes. Each convolutional layer in a CNN consists of a number of learnable **kernels**. Typically, the kernels are small in spatial dimension - 3×3 and 5×5 kernels are often used - but they spread along the full depth of the input. When the data input reaches a convolutional layer, the kernels *slide* over the data input and perform a convolutional operation for each slide. The result of this convolution is stored in a *feature map*. This convolutional operation is visualized in Figure 2.4.

Note that this setup drastically reduces the number of learnable parameters in the network. For example, if we have a $64\times64\times3$ image as input and ignore the bias, one kernel of spatial size 3×3 pixels would only contain 3 pixels $\times$ 3 pixels $\times$ 3 channels $=$ 27 learnable parameters. However, if we flatten the image and use it as input to a fully connected neural network layer with only one layer of one neuron, this would already contain $64\times64\times3 = 12,288$ learnable parameters.

The depth of the output of each layer (i.e., the number of feature maps) can be chosen by modifying the number of kernels in a layer: each kernel produces one *feature map*. Furthermore, we can also change the stride and padding parameters of the convolution.

- **Stride** is the parameter that determines how the kernel moves over the input data. If the stride is 1, the kernel moves 1 pixel for each convolution operation. If we set it to a higher number $s \in \mathbb{N}$, then $s - 1$ pixels are skipped, resulting in a lower spatial dimension feature map.
- **Padding** is the parameter that determines the behavior of the kernel at the boundaries of the data. The padding denotes the number of pixels added to the sides of an image when being processed by a CNN. The values for these pixels are often chosen to be zero, but some studies use non-zero pixels, for example, obtained by interpolation of neighboring pixels. Note that a padding of 0 will lead to a smaller size of the feature maps compared to the original image.

Stride and padding are visualized in Figure 2.5.

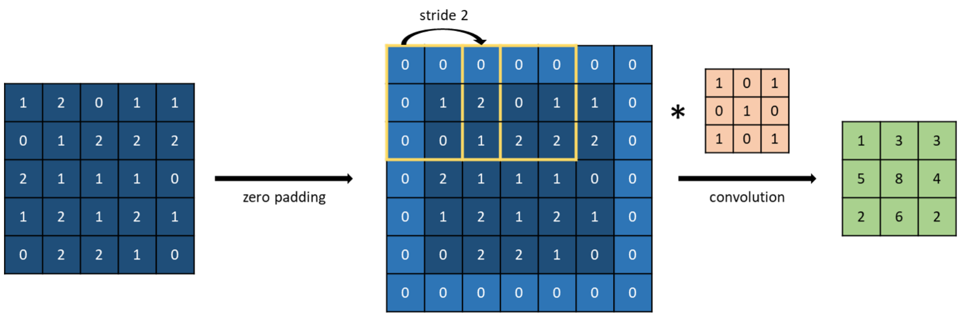


Figure 2.5: Visualization of padding and stride in a convolutional layer of a CNN. A convolutional operation with (zero) padding of 1 and a stride of 2 is depicted.

2.4.2. Pooling layers

Pooling layers are designed to reduce the dimension of the data. This is done by combining groups of neurons in a feature map into a single neuron in the next layer. The most common forms of pooling are *max pooling*, which uses the maximum value of a cluster of neurons, and *average pooling*, which uses the average value of the cluster of neurons to generate the value of the new single neuron. The max pooling operation, which will be employed in this thesis, is defined as

$$\text{MaxPooling}(x, p, q) = \max_{(i,j)\in R_{p,q}} x[i, j] \tag{2.28}$$

where x respresents the input feature map, $R_{p,q}$ denotes the pooling region of size $p \times q$.

Pooling layers help extract features invariant to translation shifts and minor distortions [38]. They can also play a role in preventing overfitting and reducing the computational complexity of the network.

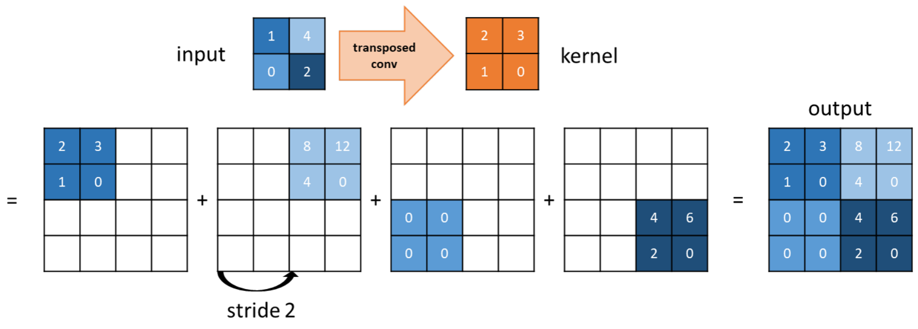


Figure 2.6: Visualization of transposed convolution (with stride 2). Note that the stride parameter in transposed convolution is applied to the output, in contrast to normal convolution, where this is applied to the input.

2.4.3. Upsampling layers

Pooling layers, striding, and padding can be used to reduce the spatial dimension of an image. However, besides downsampling, it is necessary to increase the spatial dimensions of the feature maps for many use cases. Upsampling layers can be used to do so. The simplest type of upsampling layer is the so-called *unpooling* layer, the reverse of pooling: generating a new feature map in which a cluster of neurons replaces one neuron.

Often, this operation is followed by a convolutional layer. Another option is *transposed convolution*, also known as up-convolution, de-convolution, or backward-strided convolution, in which the spatial dimensions of the feature map are increased. Instead of reducing input elements via the kernel to a single neuron, as we saw for convolution, transposed convolution broadcasts input elements via the (learnable) kernel to a larger output feature map. This process is visualized in Figure 2.6.

2.4.4. Receptive Field of CNNs

In fully connected neural networks, each input value of a layer connects to every output value. In contrast, convolutional layers in CNNs restrict connections between values in input and output to local regions, determined by the limited kernel size, see Figure 2.7 This restriction defines the receptive field, also known as the field of view [2].

Definition 2.7 (Receptive Field). *The receptive field is the region's size in the input that produces the feature in the output.*

This concept is essential for assessing CNN performance, especially in tasks like image segmentation. For instance, a pixel's class prediction in the output segmentation mask depends on the CNN's receptive field. Pixels beyond this field do not impact the prediction, although distant pixels might contain relevant information. Therefore, each output pixel must have a receptive field large enough to include all relevant information. The optimal receptive field size might differ for different datasets and tasks.

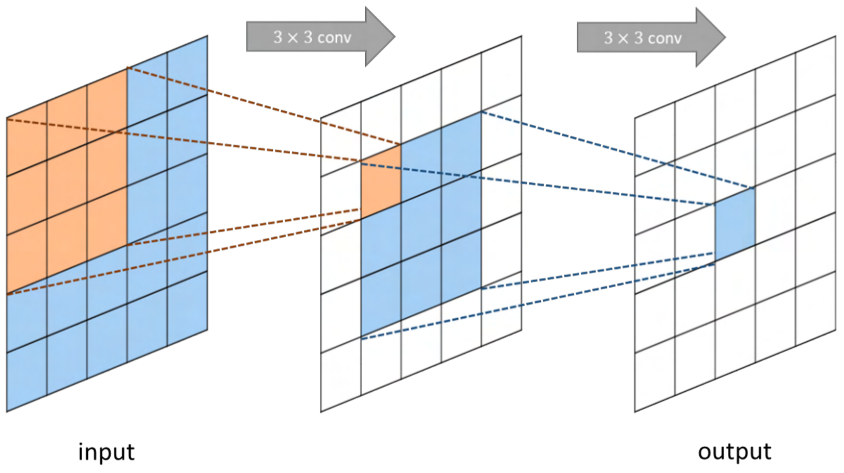


Figure 2.7: Visualization of the receptive field for a simple CNN architecture consisting of two subsequent 3×3 convolutional layers. Observe that this yields a receptive field of 5×5 pixels.

Numerous methods are available to obtain a large receptive field. Effective ways to do so are, among others, using larger kernel sizes, making the CNN deeper, adding pooling layers, and using strided convolutions (convolutions with stride parameter greater than 1) [44]. As shown clearly in [44], not all pixels in a receptive field contribute equally to the output features, as pixels in the center of the receptive field have many different paths to propagate information through to the output, in contrast to pixels in the outer area of the receptive field. The effective area in the receptive field, called the *effective receptive field*, only occupies a fraction of the *theoretical receptive field*.

3 Unraveling and Optimizing the U-Net: Introduction and Parallelization Approaches

In this chapter, we discuss one of the most successful and famous [4] CNN architectures for medical image segmentation: the U-Net architecture [55] and its most important properties. The chapter is split into two parts: first, we introduce the U-Net architecture and discuss some methods derived from it. Then, we describe the memory limitations of the classical U-Net architecture and briefly discuss some approaches proposed to solve these and their limitations.

3.1. The U-Net Architecture

The U-Net, introduced by Ronneberger *et al.* in [55], pictured in Figure 3.1, consists of two pathways: the *contraction path* or encoder path, and the *expansive path* or decoder path, which are connected by *skip connections.* In the initial version of this model, the contraction path involves repeated blocks, each consisting of two successive 3×3 convolutions, followed by a ReLU activation unit and a max-pooling layer. Conversely, the expansion path employs blocks that up-sample the feature map using a 2×2 transposed convolution. Subsequently, the feature maps from the corresponding layer in the contracting path are cropped (if the up-sampling path discards the boundary pixels) and concatenated onto the up-sampled feature maps. This concatenation is followed by two successive 3×3 convolutions and a ReLU activation unit.

In the final stage, the number of feature maps is reduced to the desired number of classes by a 1×1 convolution (with depth equal to the number of classes) to produce the segmented image. The contraction path captures (global) context and features, as well as the expansive path, especially in the deeper layers of the U-Net. The *skip connections*, which are this network's main characteristic, help transfer the fine-grained localized details between the encoder and decoder path. The classical U-Net returns a cropped segmentation mask (due to the loss of border pixels in every convolutional operation). However, more recent implementations of the U-Net often use zero padding at the borders to obtain output spatial dimensions equal to the input dimensions.

Many variants of the U-Net have been proposed. The number of convolutions, the size of the convolutional kernels, and the number of feature maps in each layer are model hyperparameters that have been varied for different applications.

The architecture of the U-Net is shown in Figure 3.1. In [55], it is shown that the U-Net achieves very good performance on different biomedical segmentation tasks. Also, it is noted that the model can work satisfyingly with only a few annotated images if data augmentation techniques are used. This last property is especially useful in the medical landscape since properly annotated images are often limited available [60]. The U-Net architecture is not rigid but rather adaptable to various applications.

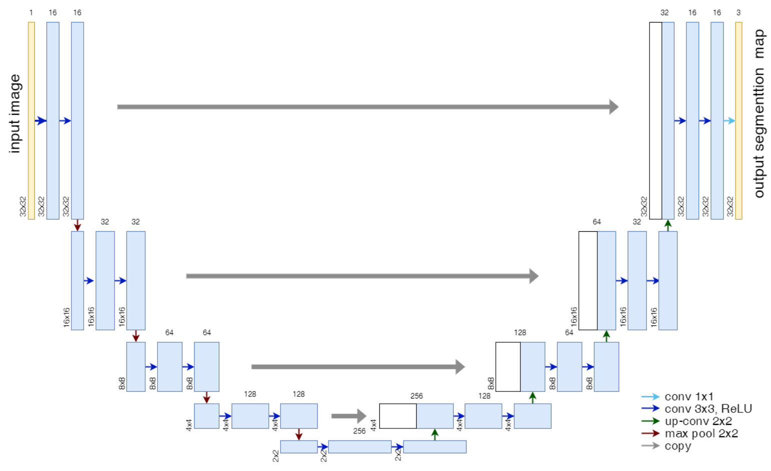


Figure 3.1: Schematic representation of the U-Net Architecture. Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. White boxes represent copied feature maps. Arrows denote the different operations. We note here that the number of channels and the image dimensions in each layer differ from the original U-Net as presented in [55].

Researchers have made adjustments to accommodate factors such as task complexity, dataset size, and device limitations. These modifications include for example changes in the number of up-sampling and downsampling blocks [30], the configuration of convolutional layers [10, 35, 53], and the use of a pretrained encoder network [56].

Since its publication, many architectures have been based on the U-Net, such as, for example, 3D U-Net [11], UNet++ [71], Attention U-Net [50] and ResUNet-a [16]. In [4], an overview is given of different models based on the original U-Net architecture. Challenges and opportunities are addressed for the U-Net family. It is concluded that, although the U-Net and U-Net-based architectures have been very successful for image segmentation tasks, a remaining constraint for most variants of the U-Net is the large memory requirements, making the model unsuitable for high-resolution applications with memory-limited computational devices [4]. These high memory requirements arise from the fact that intermediate feature maps, produced by the hidden convolutional and pooling layers, are stored during training, as those are needed for backpropagation - the gradients depend on them. The memory requirements of the U-Net for processing different image resolutions will be further explored in Chapter 7.

3.2. Optimizing Memory of CNNs

To address memory distribution challenges like those discussed in the previous section, in the training and inference of (deep) neural networks that process images, conventional strategies like downsampling and patch cropping are commonly employed, see Figure 3.2.

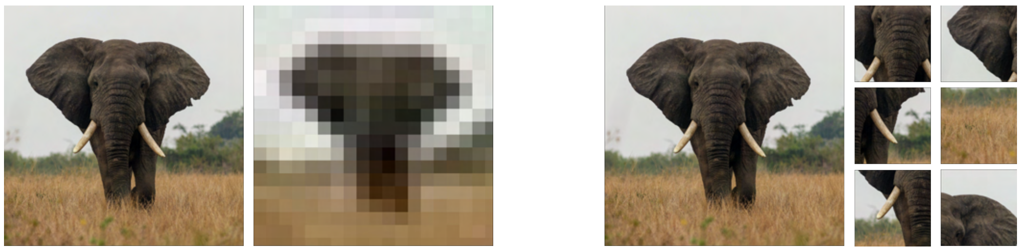


Figure 3.2: Visualization of downsampling/resizing (left) and patch cropping (right) for an example image.

Down-sampling entails resizing images, often through interpolation, to reduce memory demands. The problem with this approach is that it leads to a lower model performance if any important fine-grained details for the problem are lost in the resizing [5]. Patch cropping involves extracting smaller image patches from the training dataset, either randomly or systematically, for model training instead of using full-resolution images. This approach has two problems. First, multiple passes of inference are necessary to pass the whole image through the model. Secondly, global features of the whole image, such as perspective, image noise properties, and illumination may be split between multiple patches, leading to a worse accuracy [5, 9, 28, 66].

Another common approach to dividing the memory load of training is the use of parallelism. In machine learning, parallelism usually refers to the approach that splits the computational and memory workload into smaller parts and distributes those among several worker devices. Three of the most prevalent partitioning strategies are **data parallelism**, **model parallelism** and **pipelining** [6].

Data parallelism is the form of workload partitioning that involves a partitioning of the data set, for example when working with the mini-batch SGD method, where data is processed in batches of $k \in \mathbb{N}$ samples. Data parallelism distributes the work on the dataset among multiple computational devices. This approach can also be used to parallelize the work in one computational device such as a GPU. Data parallelism can be applied successfully when the work on samples can be done independently from each other. However, many tasks involve some global communication tasks (for example the backward propagation weights update in NN training). The balance between independent computations and the amount of communications determines how much time can be gained by data parallelism.

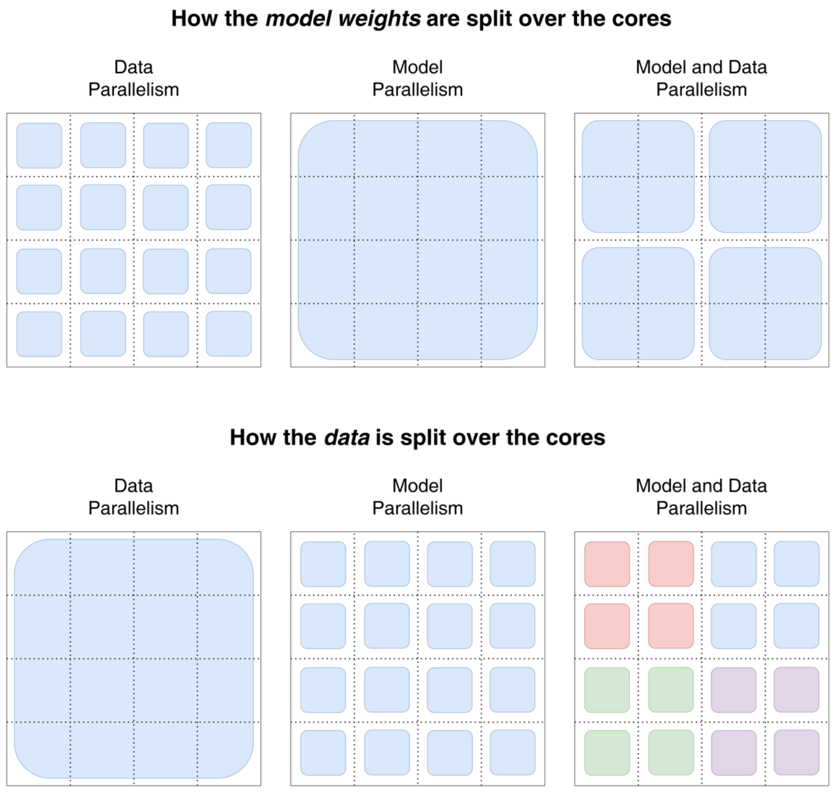

How the *model weights* are split over the cores
Data Parallelism
Model Parallelism
Model and Data Parallelism
How the *data* is split over the cores
Data Parallelism
Model Parallelism
Model and Data Parallelism

Figure 3.3: Visualization of model parallelism, data parallelism, and a combination of both classes of parallelism for neural networks. For data parallelism, the same model weights are sent to all cores, whereas the data is distributed over the different cores (non-overlapping). For model parallelism, on the other hand, the weights of the neural network are distributed over different cores, whereas the same dataset is sent to all cores.
Model and Data parallelism combines both approaches. The idea for this visualization was inspired by [20].

Model parallelism (network parallelism for NNs) on the other hand divides the workload corresponding to the neurons in each layer. This means the sample mini-batch is copied to all worker nodes and each node computes a different part of the neural network. Therefore, this also includes a partition of the network itself, which can be beneficial when the network is very large [6]. Note that this approach might need much more communication than data parallelism approaches between different processors since successive layers can be on different worker nodes.

Pipelining refers to either overlapping computations (if a sample is processed by a layer, it can start processing the next sample) or to partitioning the layers of a model over different worker nodes. Note that in a strict sense, pipelining can be viewed as data parallelism (samples are processed by the network in parallel), but also as a form of model parallelism (the length of the pipeline determines the DNN structure).

In Figure 3.3, the different forms of parallelism are visualized. As will become clear later in this thesis, there exist similarities between domain decomposition approaches and data or model parallelism: in domain decomposition, the domain on which the solution is computed is split into two or more subdomains, whereas in parallelism either the model domain or the data domain is split into smaller parts, see Chapter 4.

Much research has been done on data parallelism, model parallelism, and pipelining in neural networks. As a starting point for exploring existing literature and a more extended overview of these classes of parallelism, we refer to [22, 68].

3.3. Parallelization of the U-Net architecture

Several parallelized forms of the U-Net have been proposed to overcome the memory constraints of the U-Net. Both in [28] and in [65], the authors employ a spatial partitioning technique, that partitions the input and output of the convolutional layers into smaller, non-overlapping sub-images. Before each convolution operation, devices exchange patch margins of the feature maps of half the convolution kernel size with each other, as those are necessary to do the convolutional operation. While this approach may introduce a large communication overhead, especially for deep CNNs with large receptive fields, it is an effective approach for distributing memory across devices.

The authors of [57], on the other hand, partition the image into overlapping subdomains, with the overlap size determined by the receptive field size. Their strategy is to choose the overlap in such a way that the full receptive field of the output segmentation mask is included in the input image subdomain. This approach is successful in avoiding communication before each convolution, a large portion of the input subdomains is overlapping, leading to increased memory requirements and redundant computations, (approximately $\sim \mathcal{O}(1\frac{4R}{P})$) for an $P \times P$ pixel image partitioned into $q \times q$ subdomains and a U-Net with receptive field size $R \times R$). This approach allows for a fully parallelized execution of both forward and backward passes where no communication is needed during training. However, for large receptive field sizes, much extra memory is required for segmenting only a small image patch. A similar strategy is adopted by [66], but with an application on image classification using ResNet [25] rather than image segmentation. Although these approaches are very efficient in terms of communication, they require many redundant computations.

These approaches have been successful in partitioning the U-Net in such a way that the model can be trained and evaluated, even for ultra-high-resolution image datasets. However, they either involve communicating margins before each convolutional operation (for [28, 65]), which easily leads to communication overhead, or they entail many redundant computations (for [25, 66]).

3.4. The U-Net: Challenges and Opportunities

In this chapter, we explored the U-Net architecture, a successful CNN model for medical image segmentation. Despite its effectiveness, the U-Net faces challenges due to its high memory requirements, particularly in processing ultra-high-resolution images. These high memory requirements stem from the fact that intermediate layer outputs need to be stored for backpropagation as well as for skip connections. Various parallelization approaches have been proposed to address these challenges, such as spatial partitioning subdomain techniques. However, these methods often introduce communication overhead or entail redundant computations.

4 Domain Decomposition and (Convolutional) Neural Networks

This chapter further explores the relationship between domain decomposition and neural networks from a literature perspective. First, we start by giving a short introduction to domain decomposition. As the focus of this thesis is on applying strategies from domain decomposition methods to machine learning, we will not go into these methods in too much detail. Next, we discuss some relevant work that shows how neural network architecture and training can be combined with ideas from domain decomposition. We give a concise overview of the ideas presented in these papers and discuss how these are relevant to this thesis.

4.1. Domain Decomposition Methods

The presentation of domain decomposition methods (DDMs) in this thesis is mainly based on the textbook of Toselli and Widlund [64]. First, we describe the motivation and general idea behind domain decomposition methods and introduce a model problem that will be further used throughout this chapter. Then, we discuss subsequently (1) some traditional overlapping domain decomposition methods and (2) some traditional non-overlapping methods. Lastly, we discuss what fundamental differences exist between non-overlapping and overlapping methods and give some thoughts on how the principles of DDM can be applied to the creation and training of convolutional neural networks.

4.1.1. Basic Idea of DDMs

The basic idea of domain decomposition is natural and simple: it refers to the splitting of a (discretized) partial differential boundary value problems into coupled boundary value problems on smaller subdomains that form a partition of the original domain [64, p. V]. By splitting the domain into smaller subdomains, the problem size is reduced in terms of the number of computations and variables and as such the time necessary to come to a solution and the memory necessary to store the variables for a specific domain. Furthermore, partitioning the global domain into subdomains helps to deal with regions in the global domain where different PDEs dictate the behavior of the solution in a natural way.

Since the emergence of (parallel) computing power, domain decomposition methods have become a matter of interest to be combined with discretization methods for PDEs (finite element methods, finite volume methods, and finite difference methods), to solve differential equations more efficiently on parallel computer platforms.

Now, we introduce the model problem that will be further used in this chapter. In Figure 4.1, two ways are shown to subdivide the computational domain: either with *disjoint* subdomains or with *overlapping* subdomains. These two ways to partition the domain lead to the two main classes of DDMs: (1) overlapping DDMs and (2) non-overlapping DDMs.

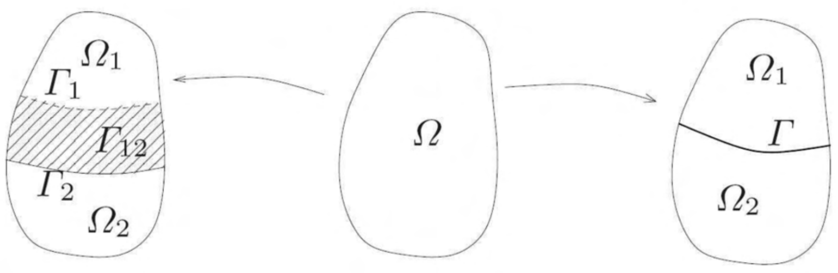


Figure 4.1: Overlapping subdomains (left) and non-overlapping subdomains (right) of the domain Ω (center). Figure reprinted from [54]

As a model problem, we define the following problem: find $u : \Omega \to \mathbb{R}$ s.t.:

$$\begin{cases} -\nabla^2 u = f & \text{in } \Omega, \\ u = g & \text{on } \partial\Omega, \end{cases} \tag{4.1}$$

where $\partial\Omega$ denotes the boundary of the domain Ω, $-\nabla^2$ denotes the negative Laplacian and g denotes the boundary condition on $\partial\Omega$.

4.1.2. Multiplicative Schwarz Method

We will show how a conventional, overlapping DDM solves this problem, namely the multiplicative Schwarz method. This method can be applied to an arbitrary number of (overlapping) subdomains.

We decompose the domain Ω into P different, overlapping subdomains. The set of subdomains is denoted by $\{\Omega_i\}_{i=1}^P$ and we construct the subdomains such that they satisfy the relation

$$\bigcup_{i=1}^{P} \Omega_i = \Omega.$$

By Γ_i, we denote the internal boundary of subdomain i: the part of the boundary of $\partial\Omega_i$ that is not a part of the original global boundary $\partial\Omega$:

$$\Gamma_i = \partial\Omega_i \setminus \partial\Omega.$$

Also, we denote by $\mathcal{N}_i$ the set of indices of neighboring subdomains of the i-th subdomain Ω_i. More formally, this means that:

$$j \in \mathcal{N}_i \iff \Omega_i \cap \Omega_j \neq \emptyset \qquad i, j = 1, \ldots, P.$$

This condition ensures that if j is in the set of neighboring indices $\mathcal{N}_i$ of subdomain Ω_i, then we know that the intersection of Ω_i and Ω_j is non-empty. Since we are dealing with an overlapping domain decomposition method, we require that there is an overlap between neighboring subdomains: every point on the artificial boundary Γ_i must also lie in the *interior* of at least one neighboring subdomain Ω_j, for $j \in \mathcal{N}_i$.

We note here that there are many different ways to define the sub-domains, all yielding different partitions with different results. However, suppose we found a way to construct the set of P overlapping subdomains. Then, we define a set of initial guesses on each subdomain, denoted by $\{u_i^0\}_{i=1}^P$. Now, we need to perform the following iteration for all indices $i = 1, \ldots, P$, *precisely in this order*:

$$\begin{aligned} -\nabla^2 u_i^n &= f_i && \text{in } \Omega_i, \\ u_i^n &= g && \text{on } \partial\Omega_i \setminus \Gamma_i, \\ u_i^n &= \overline{g}_i^* && \text{on } \Gamma_i. \end{aligned} \tag{4.2}$$

In these equations, f_i denotes the restriction of f to Ω_i, and $\overline{g}^*$ denotes the artificial Dirichlet condition on the artificial boundary Γ_i. This boundary condition is constructed using the *latest* solution on Γ_i

from a *neighboring* subdomain. This can be formalized as follows. Suppose we want to find the value of g_i^* for some $x \in \Omega_i$. Then, since the domains are overlapping, we know that there is at least one subdomain, and possibly more, that contains x. We define the set of subdomains that contain x as $\mathcal{M}(x) = \{i : x \in \Omega_i\}$. Then, we can define the artificial Dirichlet boundary condition g_i^* for the i-th subdomain Ω_i as:

$$g_i^* = \begin{cases} u_J^n(x) & \text{where } J = \max\{j : j < i \text{ and } j \in \mathcal{M}(x)\} \text{ if this set is non-empty} \\ u_J^{n-1}(x) & \text{where } J = \max\{j : j < i \text{ and } j \in \mathcal{M}(x)\} \end{cases} \tag{4.3}$$

This iterative process leads to an iterative solution that converges to the solution of the original non-partitioned problem [64]. The resulting iterates are continuous as we require the artificial boundaries to be the same for different solutions. This is also why overlapping subdomains are necessary for this method: without overlapping, continuity around subdomain boundaries is not ensured. Also, the convergence speed of this algorithm is related to the overlap size: the convergence becomes faster as the ratio of the overlap size over the subdomain size becomes larger [17]. We note that this method is still fully sequential: the computations need to be performed in the correct order (starting from subdomain Ω_1 and finishing at subdomain Ω_P).

4.1.3. The Parallel Schwarz Method

Lions was the first researcher to realize the potential of the Schwarz method on parallel computers [43]. He proposed one minor change to the classical approach to make the multiplicative Schwarz method more suitable for parallel computing. The difference between the parallel Schwarz method of Lions and the multiplicative Schwarz method lies in the method for updating the boundary conditions Γ_i. Recall that for the Multiplicative Schwarz Method, we used

$$u_i^n = \overline{g}^* \qquad \text{on } \Gamma_i, \qquad i = 1, \dots, P, \tag{4.4}$$

where the boundary condition $\overline{g}^*$ was obtained using the most recent solution in the neighboring cells, so being a combination of the solutions at the current time step u^n and the previous time step u^{n-1}. For the parallel Schwarz method, the following artificial boundary condition is used instead:

$$u_i^n = \overline{g}^{n-1} \qquad \text{on } \Gamma_i, \qquad i = 1, \dots, P, \tag{4.5}$$

instead of Equation 4.4. So, to construct the boundary condition here, we use the results of the *previous iteration*. Therefore, the boundary conditions for all subdomains are known independently, and therefore, the PDEs on the different subdomains can be solved simultaneously.

This method is inherently parallel. After each iteration, the boundary values must be communicated to update the boundary conditions for the next iteration. However, its convergence is inferior to that of the multiplicative Schwarz Method [8].

4.1.4. Multi-level methods in general

Unfortunately, numerical results show that most domain decomposition methods based only on local subdomains do not scale with the number of subdomains. *Scalability* in this context refers to the number of iterations required for convergence. Two common notions of scalability are given here, based on the definitions from [17].

Definition 4.1 (Strong scalability). *Strong scalability refers to how the solution time or the number of iterations changes concerning the number of devices used while maintaining a fixed total problem size. Ideally, the elapsed time decreases inversely proportional to the number of devices.*

Definition 4.2 (Weak scalability). *Weak scalability refers to the variation in solution time of the used number of devices while maintaining a fixed problem size per device. Ideally, the elapsed time remains constant for a constant ratio between the total problem size and the number of devices.*

Problems with many subdomains often are not scalable: the number of iterations required for convergence remains constant or even increases when we partition the problem into more sub-problems. This is because the communication between subdomains only happens at the boundaries of two neighboring subdomains, meaning that information can travel at most one subdomain further per iteration [54]. Therefore, global information travels slowly through the system, which can lead to slow convergence.

The so-called two- or multi-level methods are used to speed up convergence. These approaches introduce a ”coarse” global problem spanning the entire domain, which helps to facilitate broader communication among all subdomains, not limited to neighboring ones. Such strategies enhance convergence rates by enabling efficient dissemination of global information.

4.2. DDM Approaches for (Convolutional) Neural Networks

Machine learning has been employed to solve PDE problems, combined with approaches from Domain Decomposition. For example, Physics-Informed Neural Networks (PINNs), have been used to replace a discretization and solver for the classical Schwarz method, for example, in [34, 59]. However, this literature study will focus on neural networks designed for image- and voxel tasks such as (pixel-wise) classification and segmentation. In the continuation of this chapter, we will consider some models that are designed to perform these tasks. We note here that, to our knowledge, the number of papers published on this topic is minimal.

4.2.1. Multi-layer segmentation of retina OCT images via advanced U-Net architecture

One of the most intuitive DDM-inspired convolutional neural networks is introduced in [48]. This research aims to generate a segmentation mask for retinal layers in the human eye, a computationally heavy task because of the high-resolution input images. The authors use that the retinal layers in the left half of the training samples behave differently than those on the right side of the training images. Therefore, they decompose the training data images into two sub-images and train two separate U-Nets on both. This approach allows both U-Nets to specialize in their part of the data and reduces the overall complexity of the model. Furthermore, it allows for entirely parallel training due to the independency between the two networks. The authors conclude that the domain decomposition approach reduces the complexity of the network architectures and the training time, while the testing errors are comparable [48, p. 198].

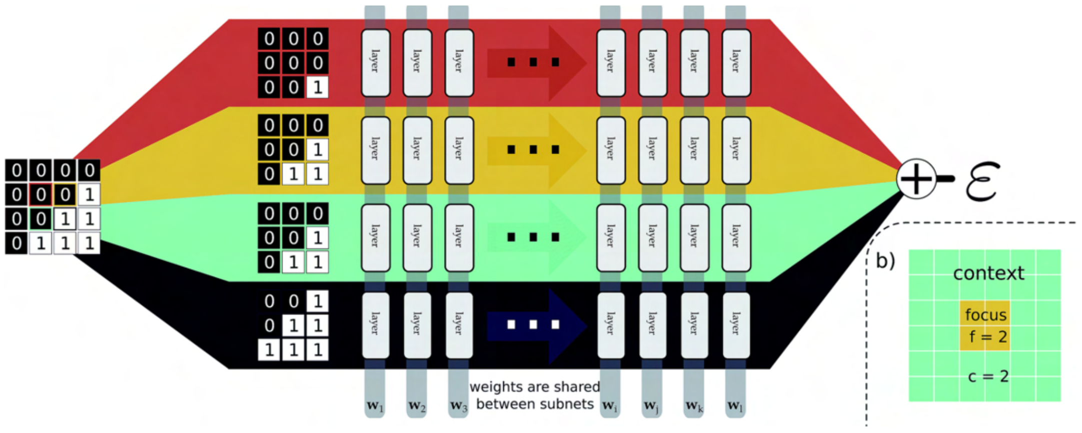

layer
weights are shared between subnets
w_1 w_2 w_3 w_i w_j w_k w_l
$\mathcal{E}$
b)
context
focus
f = 2
c = 2

Figure 4.2: Schematic visualization of the EDNN: An input example is decomposed into four tiles, each consisting of a focus (yellow) and a context (green) region. Tiles are simultaneously processed by the same, shared-weight, neural network. The individual outputs are summed to estimate ϵ, an extensive quantity. During training, the cost function is evaluated after this summation, ensuring weight updates consider all input tiles simultaneously. Colors denote different subdomains. (Figure reprinted from [49]).

4.2.2. Extensive Deep NNs for transferring small scale learning to large scale system

In [49], the authors propose a physically-motivated topology of a deep neural network, named EDNN (Extensive Deep NN), that is designed to estimate extensive parameters such as energy or entropy. An overlapping domain decomposition approach is used for training and inference. Each sub-domain consists of a *focus* region surrounded by an overlapping *context* region.

The network proposed in [49] is shown in Figure 4.2. The authors of this paper introduce the term *locality*: the spatial extent over which features in the configuration influence the value of an operator. Based on the locality of a certain application, an overlapping (context) size is chosen: the higher the locality is, the smaller the overlap size needs to be to obtain accurate results. The ML algorithm developed by the authors uses the same network to operate on the different data sub-samples. This is allowed since for extensive physical quantities there is no absolute spatial scale upon which the label of interest depends. The authors conclude that their model can learn extensive operators and can be used to make predictions for systems much larger than the system it was trained for due to the extensivity of the prediction. Compared to a network on the full image, evaluation times are much - up to a factor of one million - shorter.

4.2.3. A DD-based CNN-DNN Architecture for Model Parallel Training

Klawonn et al. [40] present a CNN-DNN architecture for image classification that naturally supports a model parallel training strategy. Their method is ”loosely inspired by two-level domain decomposition methods” [40, p. 1]. In their approach, first, the image is split into multiple sub-images. For each of the (either overlapping or non-overlapping) sub-images, a corresponding convolutional classification sub-network is trained purely on the local input data, making the training of these sub-networks completely parallelizable. The training target is a class label for the sub-images, the same as for the global image. Secondly, a global deep, fully connected neural network is trained, combining the outputs of the sub-networks, generating a final global prediction. The study interprets this DNN as a coarse problem that combines the finer-grained information from the local networks. The results of the study show that this approach can lead to a significant acceleration of the training procedure. Additionally, it can help to improve the accuracy of the classification problem.

4.2.4. Decomposition and composition of DCNNs and training acceleration via sub-network transfer learning

Gu *et al.* [21] propose another method to parallelize the training of CNNs. They divide a global classification network into several sub-networks by partitioning the width of the network (the channel dimension) while keeping the depth (the number of layers) constant, i.e., they keep the architecture of a baseline U-Net model that can be used for the global image, but they split the channel dimension over different U-Nets. They train the subnetworks individually, without inter-processor communication on the different sub-data samples. After this first training, the weights of the subnetworks are used to initialize a global network, which is then further trained to fine-tune the parameters. This approach assumes that there is an information redundancy when using global networks to deal with sub-samples. Fine-tuning is necessary to couple the different sub-samples into a global classification. Note that this approach is fundamentally different from the method in [40]: the method of Gu *et al.* aims to find a global network initialization using a DDM approach, whereas the approach of Klawonn does not further train the subnetworks for a global problem.

4.2.5. Additional literature

In [51], a DDM-inspired segmentation algorithm is inspired that uses several overlapping sub-images for training only *one* segmentation network. Experimental results show that this approach leads to better accuracy and accurate real-time segmentation of small objects. Duan et al. [18] use a non-overlapping image partitioning to generate a variational image segmentation model, which can cause inaccurate segmentation results around the boundaries of decomposed sub-domains.

4.3. Challenges and Opportunities for combining Image Segmentation, CNNs and DDMs

As discussed, many of the DDM-inspired approaches for machine learning show significant potential for speeding up training and/or evaluating extensive data sets with high-quality images. However, these methods are not developed for application on image segmentation tasks but rather on classification tasks. This thesis aims to expand these ideas and strategies from DDM to create a CNN model for image segmentation.

The following opportunities for DDM-inspired CNNs were identified:

1. **Potential for parallel training**. As research shows, DDM-inspired image processing offers the opportunity to process sub-images in parallel during training and evaluation. This can be advantageous when working with large or complex data to avoid memory issues. Additionally, this may help to speed up computations [21, 40, 49].
2. **Improved specialization**. By decomposing images and letting different sub-networks operate on different parts of the data set, overall performance can be improved. This can especially be advantageous for problems where different sub-domains have different physical properties.
3. **Scalability**. For complex data, DDM-inspired approaches appear to have the potential to be scalable to larger numbers of processors, allowing the segmentation of larger and more complex images.

Some of the most important challenges identified in this literature are listed below.

1. **Limited global communication**. The DDM-inspired algorithms described in the previous subsections mostly perform global communication by introducing overlapping sub-domains. However, this does not consider that relevant information for the segmentation can be distant or even global for the whole image. If a problem has more global features, this approach may not be sufficient, leading to lower accuracy. In [40], a coarse network is employed, but this is not trained to contain global information, as the training was purely local.
2. **Model complexity**. The introduction of sub-networks can easily increase the overall complexity of the CNN architecture since more weights must be stored.
3. **Communication overhead**. For many machine learning tasks on images, global communication is necessary between different sub-networks or sub-domains. However, this easily leads to a communication overhead.
4. **Generalization**. While DDM-inspired methods can improve accuracy or training speed, it is a challenging to ensure that the approach generalizes to diverse data sets and real-world scenarios.

These challenges need to be addressed when working on new DDM-inspired machine-learning approaches for image segmentation and classification.

5
Proposed Model and Datasets

In this chapter, we introduce a new CNN architecture that is based on the U-Net architecture but uses an advanced domain decomposition strategy to partition the image and transfer global information during training and inference. Furthermore, we present the datasets used in this thesis.

5.1. Proposed Model

In Figure 5.1, we provide a schematic representation of the network architecture proposed in this thesis. It illustrates the partitioning of an example input image into four subdomains. Starting with a dataset of high-resolution images and corresponding segmentation masks, the model takes a 2D-pixel image with dimensions $H \times W$ and outputs a probability distribution for $K \in \mathbb{N}_{\geq 1}$ classes, with shape $H \times W \times K$. Following DDM strategies, we decompose the input data into $N \times M$, with $N, M \in \mathbb{N}_{\geq 1}$ non-overlapping sub-images. It is important to note that we decompose images solely in the height and width dimensions, not in the channel dimension. Thus, each image is divided into $N \times M$ sub-images with widths W_i and heights H_j, where $i = 1, \ldots, N$ and $j = 1, \ldots, M$, and the following relation holds.

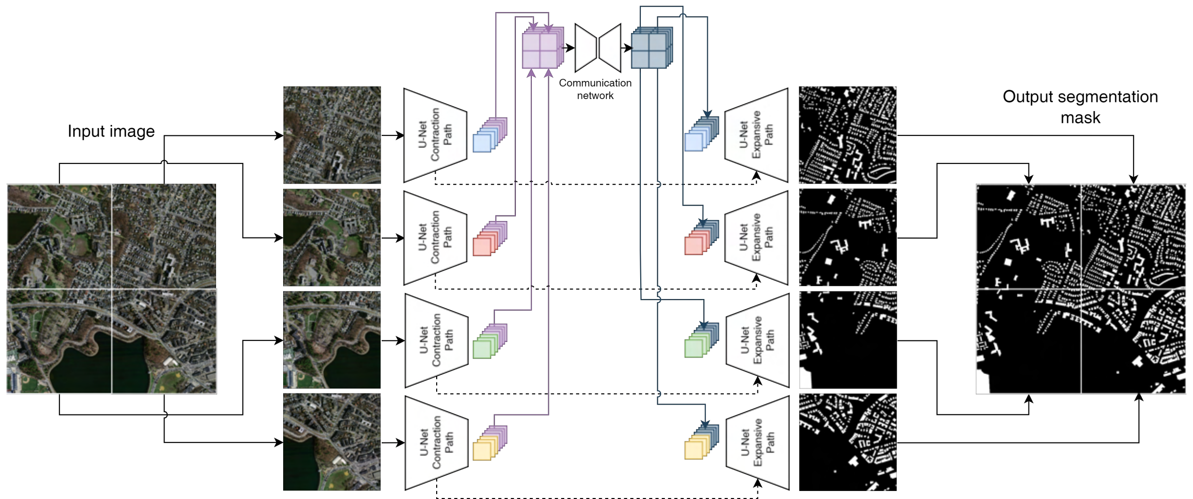


Figure 5.1: The proposed network architecture. The input image is partitioned into non-overlapping patches. Each computational device processes one or more patches. The contraction paths for each patch are processed independently. Before the expansive path operations are started, a selection of the feature maps of the bottleneck layer is communicated to one device and processed by a communication network. The modified feature maps replace the original feature maps. The expansive paths are again completely independent of each other. Dashed arrows denote the skip connections between the expansive and contraction path in the U-Net architecture. Please note that the number of subdomains can be changed depending on the size and complexity of the task.

$$\sum_{i=1}^{N} W_i = W, \qquad \sum_{i=1}^{M} H_j = H.$$

Each sub-image is sent to a distinct device (GPU/TPU), and every device hosts a local **copy** of the *subnetwork*. This means that the sub-networks have shared weights. Section 5.1.1 discusses more details about the sub-network architecture. Similar to the U-Net, these sub-networks include contraction and expansive paths.

The input sub-images are processed independently in the contraction paths during encoding. Subsequently, the last F deep feature maps generated by the last layer of the contraction path are selected and sent to the *communication module* (see Section 5.1.2). This module processes and modifies the feature maps, maintaining the spatial order of the input image. The modified feature maps are then returned to the devices, replacing the original ones. The expansive path of the sub-network is then utilized to resize the feature maps to the input dimensions and generate a segmentation mask.

The whole model is constructed so that subnetworks, consisting of an encoder and decoder, can be stored on separate GPUs. The communication module can be stored and utilized on a separate GPU or on the same device as one of the subnetworks.

5.1.1. The subnetwork

The subnetwork, depicted in Figure 5.2, handles local sub-images constituting the complete input image. Its architecture closely aligns with the classical U-Net architecture [55], but includes some adjustments in the deepest layer of the contraction path to facilitate communication between the local subdomains.

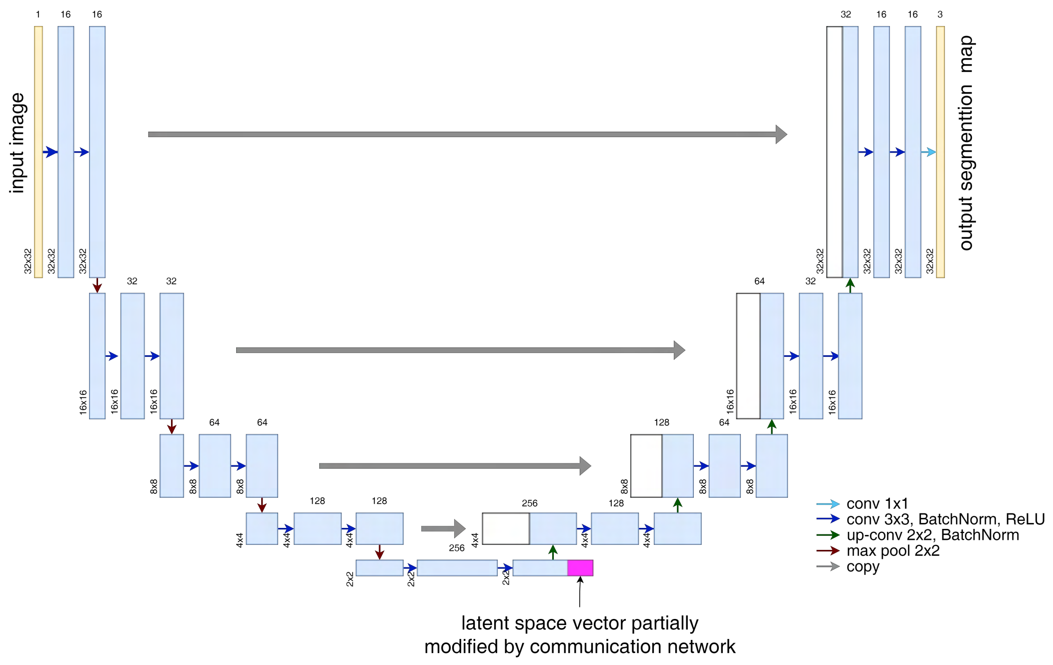


Figure 5.2: The proposed sub-network architecture. The architecture of the sub-network is almost the same as the architecture of U-Net [55], see also Figure 3.1. The only difference is located in the bottleneck feature maps, where the communication network modifies a number of the feature maps.

The subnetwork initiates the subimage processing with several blocks of 3×3 convolutions, each followed by a batch normalization layer and a ReLU activation function. After each of the two convolutions, a max-pooling operation is performed to reduce the spatial dimensions of the feature maps. The deepest layer, producing 256 feature maps, is the layer that provides the feature maps used for communication. From the 256 feature maps in this layer, the last F are selected and directed to the communication network (Section 5.1.2). This specialized network modifies the input feature maps and returns a set of F adjusted versions that replace their original feature maps.

After communication, the feature maps progress through the expansive path, yielding a final segmented sub-mask. By concatenating the different segmentation sub-masks, a full mask covering the whole image can be obtained.

The number of channels, the number of down- and sampling blocks, the size of the convolutions, and the number of feature maps communicated are hyperparameters that need to be optimized for specific datasets by doing a hyperparameter search.

5.1.2. The communication network

The communication network, depicted in Figure 5.3, is crucial in transferring contextual features captured by the local subnetworks across subdomains. The architecture of the communication network can be chosen arbitrarily as long as the spatial dimensions of the output feature maps are equal to the spatial dimensions of the input feature maps. For this thesis, the communication network comprises three layers of 3×3 (or sometimes 5×5) convolutions. We will vary the numbers of communicated feature maps in the experimental results in Chapter 7.

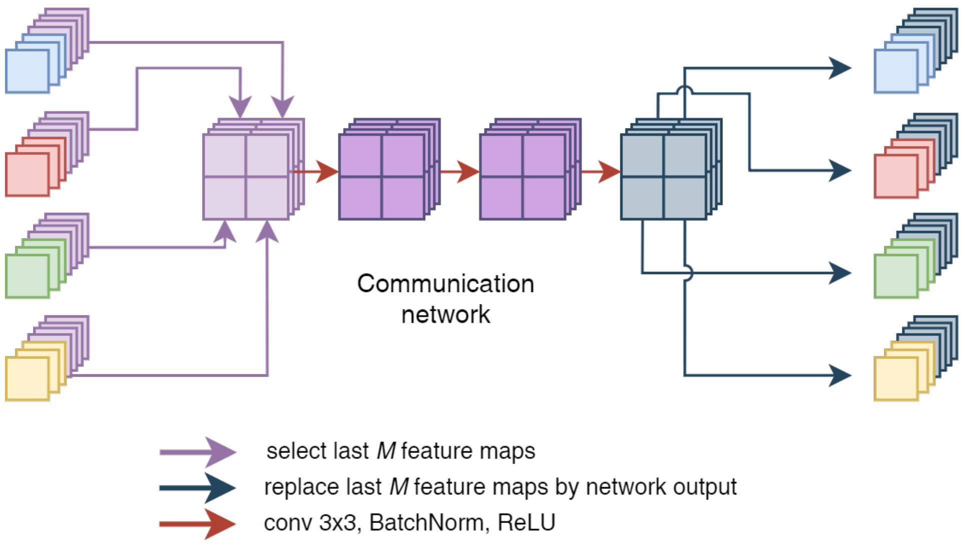

Communication network
select last M feature maps
replace last M feature maps by network output
conv 3x3, BatchNorm, ReLU

Figure 5.3: The proposed communication network for four subdomains as used in this thesis. The colors in this figure correspond to the colors of the feature maps in Figure 5.1.

The communication network takes as input F feature maps generated by the deepest layer of the contraction path from all $M \times N$ subdomains. To maintain spatial structure, these feature maps are concatenated along the height and width dimensions, aligning with the subdomains' positions in the full input image. The network processes this concatenated input and produces F output feature maps with the same spatial dimensions as the input. The output feature maps, preserving the spatial structure, are then cropped to correspond with the dimensions of the input feature maps. Subsequently, these modified feature maps are sent back to replace the feature maps that were used as input to the communication network. We emphasize here that the architecture of the communication network can be adjusted to fit the requirements of a specific task or image dimension by varying the type, complexity, and number of layers in the communication network, provided that the input and output dimensions stay the same.

5.1.3. Relation between proposed model and domain decomposition

Domain decomposition strategies inspire the proposed model. One notable similarity between the model and DDM strategies lies in the partitioning of the image domain into discrete, non-overlapping subdomains, compared to the segmentation of PDE domains into smaller computational units. Each subimage corresponds to a distinct subdomain Ω_i, where the segmentation task is performed independently for efficient processing. This strategy is equivalent to the principle behind domain decomposition in PDE solvers, where computational and memory loads are distributed among different devices, to improve parallelism and scalability.

Moreover, the use of down-sampling and up-sampling layers can be interpreted as equivalent to multi-level domain decomposition methods. The down-sampling path reduces the resolution of the input image, creating a coarser representation of the problem, which resembles the creation of higher-level, coarser subdomains in multi-level decomposition methods.

However, the differences are also important to mark here. The most significant difference lies in the tasks that our model addresses compared to DDMs. While DDMs are primarily employed to solve PDEs on a given domain, our model is designed for the machine learning task of segmentation, which is not directly relatable to PDEs. Consequently, the objectives and solvers (namely the use of CNNs and machine learning techniques) employed in the proposed model diverge from those in traditional domain decomposition methods.

5.1.4. Relation between proposed model and model/data parallelism

n Chapter 3, we explored the concepts of model and data parallelism, both of which are integral to the proposed image segmentation model. On one hand, the proposed approach uses elements of model parallelism by partitioning the image domain over multiple GPUs. Although it does not strictly fit the conventional model parallelism definition, as the encoder and decoder weights are shared among different devices, the parallelization of encoders, decoders, and the communication network effectively distributes computational tasks across multiple devices.

On the other hand, the model also integrates a form of data parallelism by distributing the input data across multiple GPUs. However, unlike traditional data parallelism approaches where the dataset itself is partitioned and distributed (the full images instead of sub-images), the proposed model operates on sub-images of each image in the dataset.

5.2. Training Procedure

The training procedure followed to train the proposed model on multiple subdomains is shown for a 2-GPU example in Figure 5.4, where GPU0 contains both a subnetwork and the communication network, and GPU1 only contains a subnetwork. The following steps are followed. We assume the model and corresponding optimizer (for this thesis, the Adam optimization algorithm [39]) is already initialized.

1. **Forward Propagation:** The model initiates a prediction. During this phase, the encoders on both GPUs can operate concurrently. For communication, GPU1 transmits (a selection of) the subnetwork's bottleneck feature maps to GPU0, where they undergo processing in the communication network. A part of the communication network's output is relayed back to GPU1. Subsequently, the decoding of these bottleneck feature maps can proceed in parallel.
2. **Loss Computation:** Following forward propagation, losses are computed for the predictions. This computation occurs locally per GPU without communication, as the predicted and true masks in different subdomains are independent. Notably, this yields two separate losses for both GPUs.
3. **Backward Propagation:** The computed losses are back-propagated through the model for both subdomains. Inter-GPU communication is once again required during this step, as subnetworks for other subdomains contribute to generating feature maps in the bottleneck layer.
4. **Gradient Accumulation:** Upon completion of backward propagation, the gradients for the encoder and decoder weights are accumulated on one device (GPU0 in this instance).
5. **Optimizer Step:** With the gradients collected on one GPU, the optimization algorithm is applied to the model weights, leveraging the computed gradients to update the model parameters.
6. **Weight Synchronization:** Following the optimization step, the updated weights are communicated to the other GPUs to ensure uniform usage of subnetwork weights in subsequent training iterations across all GPUs.

We stress here that the training procedure for 2 GPUs can be extended without much adjustments to work more subdomains. Additionally, users have the flexibility to choose different model storage configurations based on factors such as the model size, memory usage of various components (such as the optimizer and communication network), and GPU configuration. For instance, users can opt to store and train the communication on a GPU without a subnetwork, thereby allowing for a larger

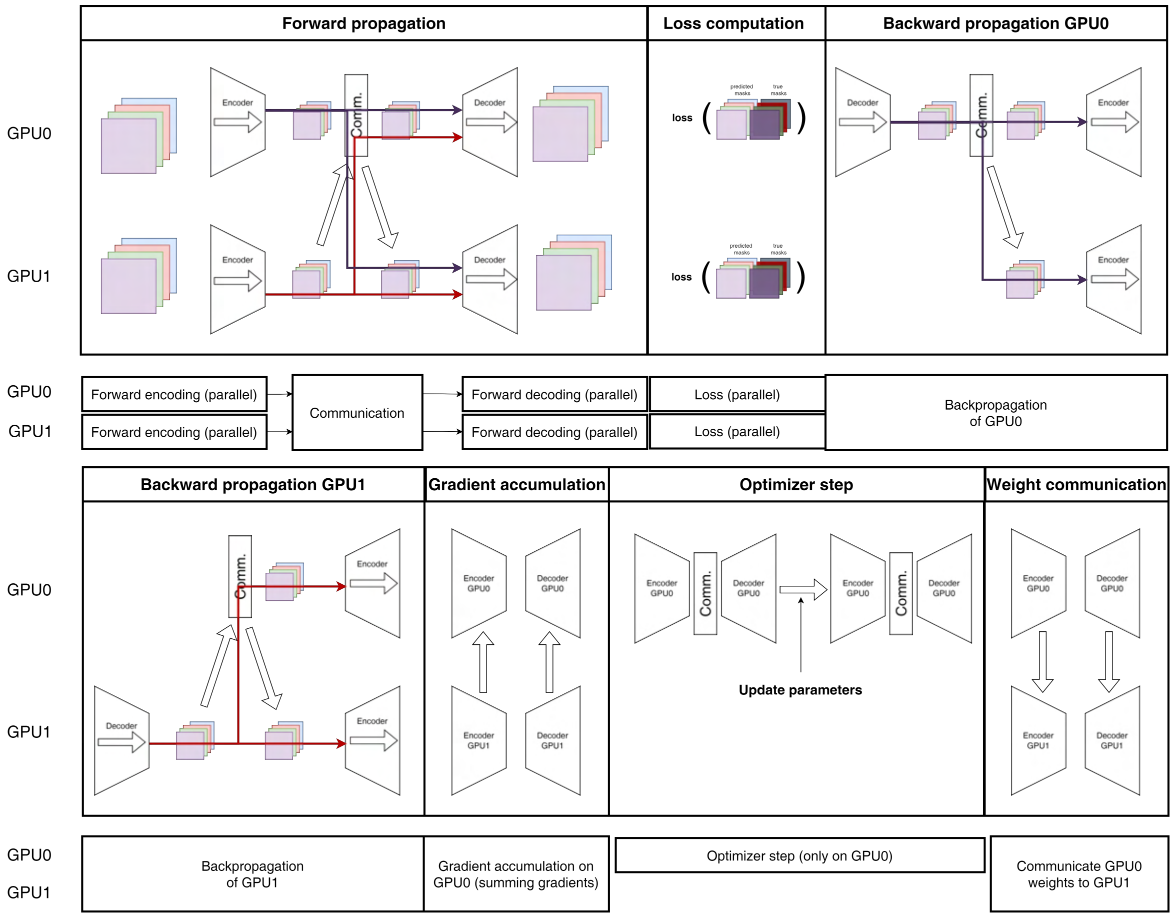


Figure 5.4: Training procedure followed for training the proposed model, where the model is split over two devices, GPU0 (containing a subnetwork and the communication network) and GPU1 (containing only a subnetwork). This picture is best viewed online.

communication network. Similarly, updating model weights on a GPU separate from the one containing the communication network can help distribute memory load more optimally.

5.3. Datasets

To assess the efficacy of the proposed network architecture, we use three different image datasets, that is, one synthetically generated clean dataset specifically designed to test the communication module and two realistic, high-resolution image semantic segmentation datasets, one for binary building segmentation, and one for multi-class land cover segmentation.

5.3.1. Synthetic dataset

In our novel approach, only deep feature maps are exchanged between subdomains, differing from previous U-Net parallelization methods [28, 65], that involve the exchange of feature maps at each U-Net layer before every convolutional operation. We crafted a synthetic dataset to assess the capability of transferring fine-grained details through the spatially coarse and deep feature maps. This dataset comprises one-channel grayscale images with dimensions $32k \times 32$ pixels (width, height), where $k \in \{2, 3, 4, 6, 8, 16\}$. This design allows the decomposition of images into k non-overlapping subdomains of 32×32 pixels. Two example images of 4 subdomains are shown in Figure 5.5.

Each image in the synthetic dataset depicts two white circles with a radius of 4 pixels, positioned entirely within separate subdomains to avoid crossing subdomain borders. The areas outside the circles are filled with black background pixels. For each image, a corresponding segmentation mask is crafted, containing three classes: (1) background, (2) a line segment that connects the centers of the two circles, and (3)

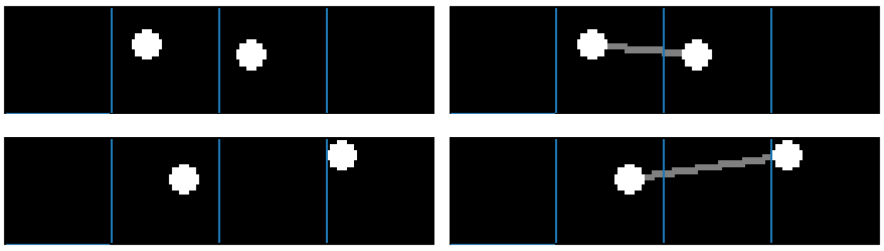


Figure 5.5: Two example images (left) and masks (right) from the synthetic dataset, both with dimensions 128 × 32 pixels. The vertical blue lines show the subdomain borders used for these images.

background pixels. To limit the maximum receptive field size necessary for successful segmentation, we limited the distance between two circles to have one subdomain in-between at maximum. This means that the circles can be maximally two subdomains apart.

Effective segmentation of the line connecting the circles entirely depends on successfully communicating the circle center positions across different subdomains. This synthetic dataset helps assess the proposed network's success in communicating global information. Furthermore, it facilitates evaluating the model's performance when trained on a limited number of subdomains but tested on a more extensive set when we leverage the input-size agnostic architecture of CNNs.

5.3.2. Generation of synthetic dataset

We shortly describe the procedure followed to generate the synthetic dataset. We develop six synthetic datasets with varying image dimensions. Each dataset consists of gray-scale images containing two white circles connected by a line segment. The images are generated based on a fixed number of subdomains. One subdomain is randomly chosen to place the first circle, and the second circle is placed within a specified range of subdomains from the first one, but never in the same subdomain. Circle positions are uniformly randomized in their subdomain, but the pixels closer to the boundary than the circle radius are excluded from this process to avoid the circle overlapping with the boundary. For the mask images, besides the circles, a line segment is drawn between the circle centers, with the circles in the foreground. See Table 5.1 for the dataset parameters used for generation of the synthetic data.

Parameter	**Value**
subdomain size	32×32 pixels
image dimensions (width, height)	$32k \times 32$ pixels, with $k \in \{2, 3, 4, 6, 8, 16\}$
number of circle pairs	2
circle radius	4 pixels
line width	2 pixels
# of subdomains between the two circles in one pair	0 or 1 (randomly chosen)

Table 5.1: Properties of the synthetic datasets used in this work.

5.3.3. Inria Aerial Image Labeling Dataset

The Inria Aerial Image Labeling Dataset [47] is a semantic segmentation dataset that covers different suburban landscapes, ranging from densely populated areas to alpine towns. It covers 5 cities, with 36 images per city. This dataset aims to assess the generalization power of different segmentation techniques when used on varying illumination, landscape, and temporal conditions. The dataset provides 180 images from five cities, each of size 5000×5000 pixels and annotated with binary masks for building/non-building areas. The dataset has a spatial resolution of 30 cm/pixel. In Figure 5.6, two example images and masks are shown.

To make the dataset suitable for our proposed model, some preprocessing steps were necessary. First of all, the image size was too large to fit on one GPU. Since at first, only one GPU was available for training, we decided to train the network on 512×512 patches. For further preprocessing, we followed

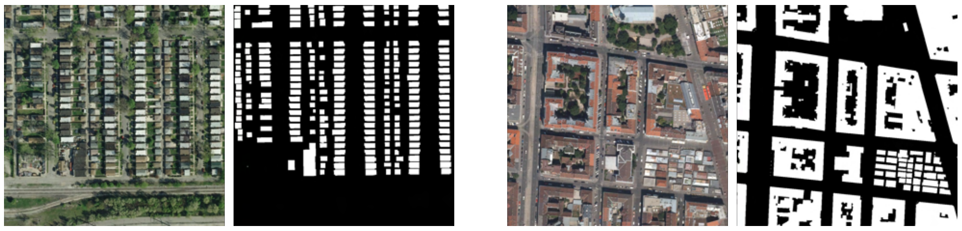


Figure 5.6: Two example image patches and their corresponding masks from the Inria Aerial Image Dataset [47].

the procedure as followed in [9]. The procedure follows here.

1. For each of the 5 cities, 5 images were randomly selected for testing, 5 for validation and the remaining 26 images were used for training.
2. For each image in the dataset 10 × 10 image patches of 512 × 512 pixels were extracted, with 32 pixels overlap, following the approaches for segmentation of this dataset presented in [29].
3. Patches that did not contain a building were discarded.
4. Using the same images, 2048 × 2048 patches were also extracted for evaluation and testing on a larger scale.

Before being sent to the network, the *RGB* channel was mapped to the range $[0, 1]$ for all three channels.

5.3.4. DeepGlobe Land Cover Classification Dataset

The DeepGlobe Land Cover Classification dataset [14] is a semantic segmentation dataset for land cover types. The dataset contains 803 high-resolution (2448 × 2448 pixels) annotated satellite images with 7 classes: urban, agriculture, rangeland, forest, water, barren, and unknown. The images have 50 cm pixel resolution and span a total area size of 1716.9 km^2. One of the challenges involved in segmenting this dataset is the large class imbalance (see Table 5.2). We address this challenge by employing the Dice loss function that takes this imbalance into account [33]. In Figure 5.7, we show two example images and masks from the dataset

Class	Pixel count	Proportion
Urban	642.4M	9.35%
Agriculture	3898.0M	56.76%
Rangeland	701.1M	10.21%
Forest	944.4M	13.75%
Water	256.9M	3.74%
Barren	421.8M	6.14%
Unknown	3.0M	0.04%

Table 5.2: Class distributions in the DeepGlobe land cover classification dataset. Table entries are retrieved from [14]

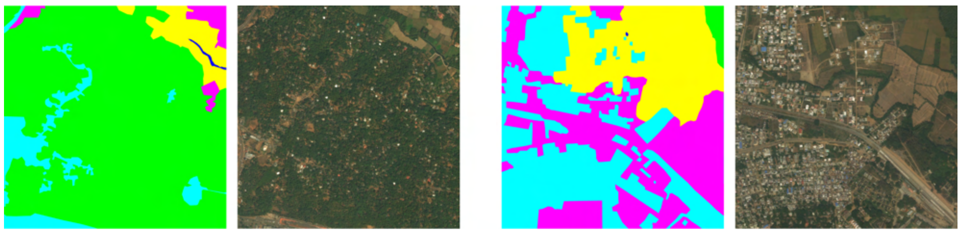


Figure 5.7: Two example images and their corresponding masks from the DeepGlobe Aerial Image Dataset [14].

The DeepGlobe dataset was preprocessed for this work using the following procedure. This preprocessing is necessary to ensure that the model we used is working with the limited memory available for training.

1. The data was randomly split into a training, validation, and testing dataset, with a ratio of 70-15-15, respectively.

2. All images and corresponding masks in the training dataset were cropped to 512×512 pixel image patches, 16 for each image. These patches are adjacent, but non-overlapping and start from the top-left corner of the image. Note that the 400 pixels left at the boundaries were discarded in this process.
3. The images and corresponding masks in the test and validation dataset were also cropped, but to 2048×2048 patches, 1 for each image in the datasets. Note that for these datasets, the 400 pixels at the right and bottom boundary were also discarded for this work.
4. Before being passed to the network, all images were normalized using the mean and standard deviation found for the ImageNet dataset [15].

6

Preliminary Results

Before we consider the results of the proposed model, we need some preliminary findings that show the effectiveness of the proposed architecture. Those results will be presented in this chapter. We focus on two different themes: (1) the transfer of positional information through CNNs, (2) the motivation for the choice of the communication network architecture, and (3) the memory requirements for our model compared to a baseline U-Net model.

6.1. Convolutions, Positional Information and Communication

This section explores vital components essential for understanding the communication module used later in this work. These concepts concentrate on the idea of positional encoding in CNNs. The problem investigated in this subsection can be stated as follows.

Problem Statement. Can a neural network architecture composed solely of convolutional and pooling layers effectively retain positional information despite the *translational equivariance* of convolutional filters and the *translation invariance* of (global) pooling operations?

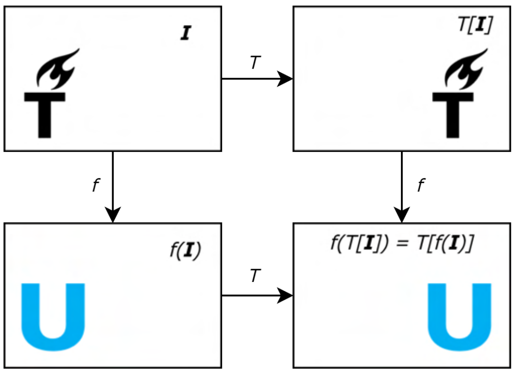


Figure 6.1: Illustration of translational equivariance. The operator, T, denotes a shift of the object in the input image to another position in the input. In contrast, the operation f denotes the output after applying the equivariant operation on the image (or the translated image).

6.1.1. Motivation

Convolutional neural networks revolutionized computer vision by leveraging convolution operations. These operations involve sliding a small filter (kernel) over an input image and performing a dot product at each location (see Chapter 2). One crucial theoretical property of convolution is that this operation is *equivariant*: shifting an object in the input shifts the output equally. This is an advantageous property when working with structured data and was therefore even further exploited in the so-called *group* *equivariant networks*, which are also equivariant under 90-degree rotations and mirror reflection [13].

However, for this thesis, we only consider conventional CNNs, with their equivariance properties, which can be formalized as follows. For $T : \Omega \to \Omega$ a translational operation for $\mathbf{x} \in \Omega$, $f : \Omega \to \Omega$ an operation on x, this means the following:

$$T[f(x)] = f(T[\mathbf{x}]), \qquad \mathbf{x} \in \Omega. \tag{6.1}$$

For an illustration of this, see Figure 6.1.

Pooling operations, like global average or maximum pooling, on the other hand, provide translational invariance (see Figure 6.2. Suppose that $T : \Omega \to \Omega$ denotes a translational operation again, and $f : \Omega \to \mathbb{R}^n$ denotes an operation (for example, image classification). Then, translational invariance is defined as:

$$f(\mathbf{x}) = f(T[\mathbf{x}]), \qquad \mathbf{x} \in \Omega \tag{6.2}$$

Local pooling on small pixel groups can be interpreted as a form of localized translation invariance: we obtain translational invariance for small groups of pixels. Despite the common assumption that CNNs are equivariant and globally invariant, several studies have challenged the assumption that CNNs are entirely translational invariant [32, 37]. These studies show that CNNs can exploit absolute object locations, utilizing image boundary effects and positional-encoding artifacts from the convolution operation around boundaries.

While translational invariance is a valuable property for some machine learning tasks, retaining positional information can sometimes also benefit tasks like segmentation, where object positions influence predictions. This necessity is evident in the synthetic dataset used in this work, where accurate segmentation relies on communicating circle positions effectively.

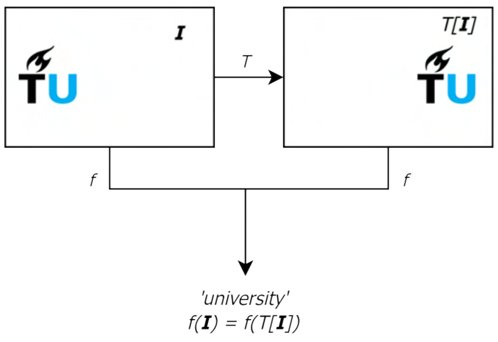


Figure 6.2: Illustration of translational invariance. The operator T denotes a shift of the object in the input image to another position in the input, whereas the operation f denotes the output after applying the translation-invariant operation (classification in this case) on the image (or the translated image).

For the proposed model, communication only takes place in the bottleneck layer, where all information is extracted to a very coarse spatial level. If the used architecture is unable to encode positional information, this would mean that the communication network can only communicate coarse-level information. There are many cases where one would like the communication network to communicate also finer-grained information, for example, to ensure consistency around the subdomain borders.

6.1.2. Theoretical background

To provide a theoretical understanding of the mechanism that allows the equivariant CNN operation to include positional information, we consider a 1-dimensional example. Let $\mathbf{x} \in \mathbb{R}^n$ be a 1-D single channel input image of size n pixels, and let $\mathbf{f} \in \mathbb{R}^{2k+1}, k \in \mathbb{N}_{>0}$ denote a 1-D single channel $(2k+1)k \times (2k+1)$ kernel for discrete convolutions. We only consider odd-sized filters of size $2k+1$. Then, the output of the discrete convolution operation is given by

$$\mathbf{y}[t] = \sum_{j=-k}^{k} \mathbf{f}[j]\mathbf{x}[t-j]. \tag{6.3}$$

Since images have finite support, we need to deal with the boundary cases $j < k+1, j > n-k$. The most common way to deal with these is the so-called *zero-padding*: it is assumed that all the values outside the image are equal to zero. Different padding types include:

1. *Valid padding:* convolving around the boundary is avoided, leading to an output of size $n-2k$ pixels. For Equation (6.3), this means we only consider the output in the region $t \in \{k+1, \ldots, n-k\}$.
2. *Same padding:* we preserve the input image dimensions by adding zeros at the boundaries and sliding over the region $t \in \{1, \ldots, n\}$. This means that $2k$ of the values during the convolution fall outside the support of the convolution f.
3. *Full padding:* this means that we apply each value in the filter f to all values in the image, i.e. $t \in \{-k, \ldots, n+k\}$, leading to an output with size $n+2k$ pixels.

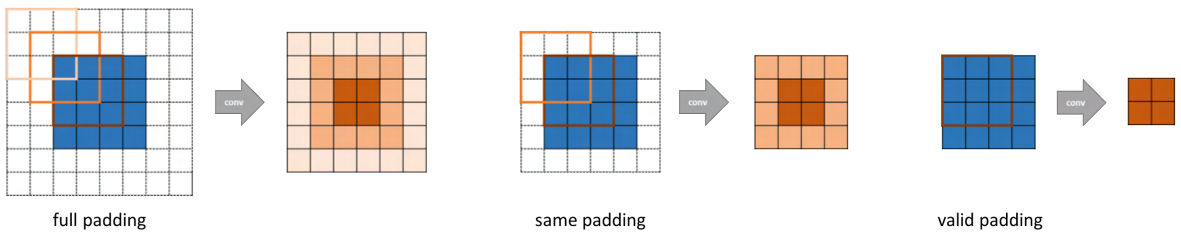


Figure 6.3: Visualization of different padding types for a 3×3 convolution with stride 1. The blue area represents the input data, with zero padding indicated in white. The gray areas depict the convolutional operation, and the orange regions represent the output feature maps. The intensity of the orange areas increases as fewer zero-padding values are used in generating the features.

The mentioned padding types are depicted in Figure 6.3. As can be noted here, the zero padding leads to boundary artifacts: only the kernel weights that overlap with the image contribute to the feature map. This leads to a difference between outer and center pixels in the further translationally equivariant convolution operation. If several convolutional operations follow each other, this can lead to the propagation of this boundary artifact through all layers. In [32], it is claimed that these artifacts make it possible to encode the positional information.

6.1.3. Experimental set-up

To verify if the CNN architectures can learn positional information, we designed another synthetic dataset and corresponding neural network architecture.

Dataset. The synthetic dataset designed for this task contains 8,000 training and 2,000 validation gray-scale images of 64×64 pixels. Each image consists of a black background, with four pixels randomly placed in the image. All 4 pixels in an image have a different color. For each of the pixels, the position of the pixel is stored, leading to a vector of 8 values corresponding to each image. For examples, see Figure 6.4.

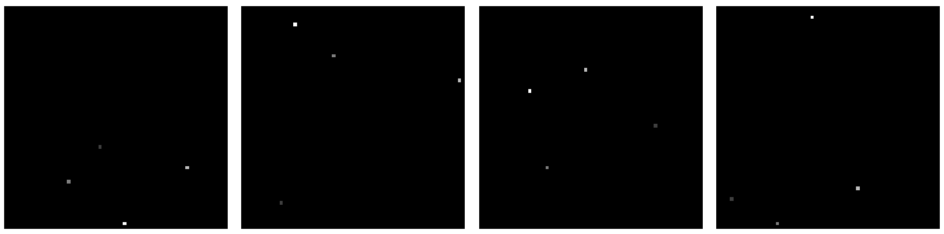


Figure 6.4: Example images from the synthetic dataset used for testing the equivariance properties of the encoder CNN.

Model. We test the ability to encode positional information for a model that shows much resemblance with the U-Net encoder. The model consists of 4 consecutive 3×3-convolution + local 2×2 max-pooling blocks. In each block, the number of channels is doubled, whereas the spatial dimension in each direction halves. In the last layer of the network, global average pooling is applied on the resulting feature maps, and at last, a 1×1 convolution is applied to reduce the number of channels from 64 to 8 (the number of coordinates to predict). The model is depicted schematically in Figure 6.5.

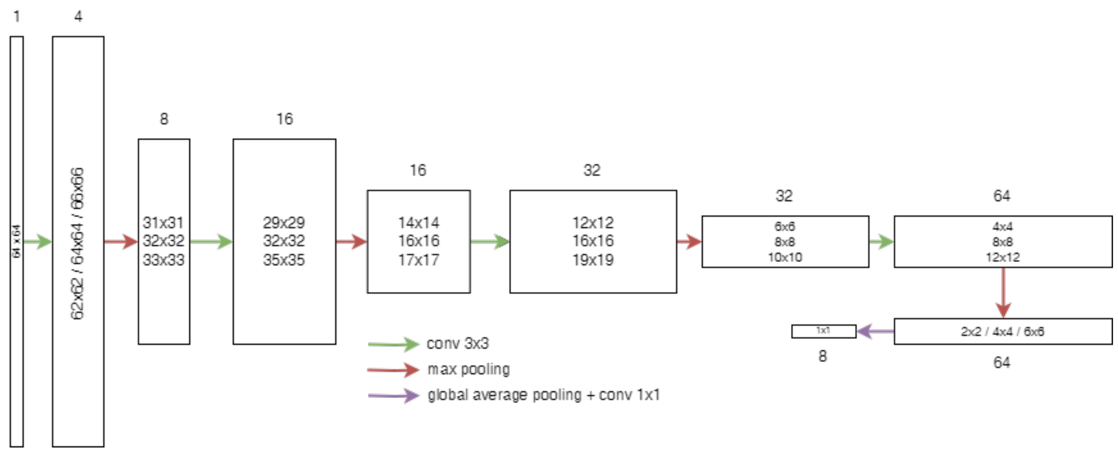


Figure 6.5: Model employed to investigate positional information encoded within CNNs. The architecture closely resembles that of the encoder depicted in Figure 5.2. Numbers atop rectangles denote the number of feature maps in a layer, while those within the bordered rectangles denote spatial dimensions for full, same, and valid padding, respectively.

Task. As described, for each synthetic image, we have a vector with eight positional values, and the model returns a vector of 8 values. We train the model to predict these locations. Suppose that the model is not able to learn any location information at all. Then, the best location it can learn to predict is the center of the square images, since this position has the lowest average distance to all points in the square. The average distance between points in a square and its center is given by:

$$\textit{average distance} = \int_{-L/2}^{L/2}\int_{-L/2}^{L/2} \frac{1}{L^2} \cdot \sqrt{x^2 + y^2}\, \mathrm{d}x\, \mathrm{d}y = \frac{L}{6}(\sqrt{2} + \log(1 + \sqrt{2}))$$

where L is the width and height of the square. For our synthetic 64×64 pixel dataset, this gives an average distance of 19.17 pixels between the image center and an arbitrary pixel. If we find a CNN model that gives (significantly) better predictions than this model, we can conclude that the model has learned to encode at least some positional information.

On the other hand, if the model is fully translational invariant, then the positional predictions should be inaccurate, as translational invariance implies that, if we feed the model two images, where image 2 is a translated version of image 1, the predicted result for both images is exactly equal.

Loss and training hyperparameters. For training, we employ the Adam optimizer with hyperparameters as shown in Table 6.1. As a loss function, we use the mean squared error. This loss function is natural to use, as this loss is similar to the distance metric between the true and predicted point (squared).

Hyperparameters	
(max.) learning rate	0.008
(max.) number of epochs	80
early stopping patience	-
learning rate decay patience	-
batch size	16

Table 6.1: Hyperparameters used for experiments for CNNs and positional encodings

6.1.4. Results

The results of the described experiment for different padding types are shown in Table 6.2. Again, we emphasize here that the number of parameters, layers, and convolutions for the results with valid, same, and full padding is exactly equal. The only difference between these three approaches is in the number of padding pixels added around the training images. The results in Table 6.2 show very clearly that a larger padding number leads to much more accurate predictions of the pixel locations.

	Valid padding	***Same padding***	***Full padding***
MSE loss	110.9	33.6	14.0

Table 6.2: MSE loss for pixel position prediction using a fully convolutional architecture.

To get a better idea of the meaning of the results shown in Table 6.2, we recall that the shown MSE loss is the mean squared error loss for both of the two pixel coordinates, in x and y direction. That means that the average distance between a true and predicted pixel location is given by:

$$\text{average pixel position} = \sqrt{2 \cdot MSE_{loss}}.$$

by the Pythagorean theorem. This means that we can rewrite the MSE loss to the average distance between a predicted and true location as in Table 6.3.

	Valid padding	***Same padding***	***Full padding***
Average distance (pixels)	14.9	8.2	5.3

Table 6.3: Average (pixel) distance between true and predicted pixel for different padding types.

These results show us that the error in predicting the location of the four pixels gets smaller as the number of padding zeros increases. Furthermore, it shows us that none of the investigated CNN architectures is *not* fully translational invariant, as we would have expected a larger loss for translational invariant CNNs then.

The relevance of these results for our proposed model is that, even in deeper layers, a CNN can encode positional information, especially when same or valid zero padding is applied. This means that in the bottleneck layer feature maps can carry information, not only on global or semi-global features, but also more local, fine-grained information, which could for example be relevant for predicting continuous, consistent segmentation masks as the boundaries.

6.2. Communication Network Architecture

In this section, we motivate the choice for the communication network as used for the results presented in Chapter 7. As we already noted in Chapter 5 when introducing the proposed model architecture, the exact composition of the communication network is not fixed: the user can adjust the network, as long as the spatial dimensions of the input and output feature maps have equal sizes. Adjustments can be made to take into account the complexity, structure, or locality of the data at hand. Here we discuss two potential communication network architectures with their properties and potential downsides.

A fully connected communication network is the most general choice, as this allows information to be exchanged between all positions in the communicated feature maps. This approach leads to a *receptive field* with a size equal to the full domain size. By adding one or more hidden layers between input and output, the *learning capacity* of the neural network can be enlarged, allowing the model to learn complex patterns and relationships between the communicated feature maps.

The largest downsides of using a fully connected NN as a communication model are that (1) the number of parameters increases very fast as the spatial dimensions increase, (2) preserving spatial structure information from the feature map structure is not explicitly enforced in this approach and (3) the fact that the input data size to the network should be fixed.

CNN communication networks on the other hand are more restricted, as they make use of the spatial structure of the communicated feature maps. Their *receptive field* is limited by the number of convolutional layers and the size of the convolutional kernels. This limited receptive field size might also limit the model to learn *global* features from the data. The *spatial structure* of the feature maps is preserved for this communication network. The number of parameters is limited, also for large spatial input dimensions (as the number of parameters in a CNN is not dependent on data input size) and the spatial size of the input data does not need to be fixed for CNNs. The downsides of this approach are (1) the limited receptive field size of CNNs and (2) the enforced spatial structure by the convolutional operations.

Complex CNN architectures such as the U-Net, which also involve down-sampling and up-sampling layers, skip connections, and pooling operations might be able to overcome those two downsides sketched for CNNs. However, in the limited, exploratory scope of this thesis, we will only consider simple CNN and NN architectures, without pooling or skip connections. We use a CNN with a relatively large kernel size (5×5 pixels) and three subsequent layers to get a receptive field of 13×13 pixels.

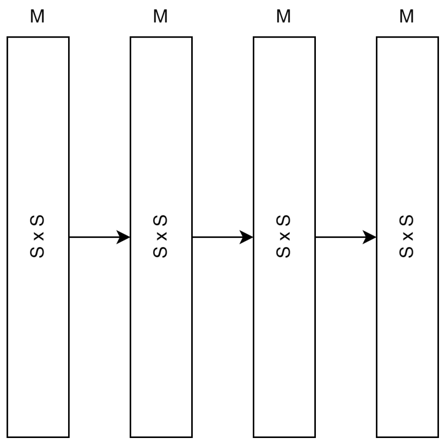


Figure 6.6: Communication network used for computing the number of parameters. M denotes the number of feature maps in a specific layer, and $S \times S$ denotes the spatial dimensions in 2D of a given layer. The black arrow denotes a fully connected layer for the fully connected communication network and a 5×5 convolutional kernel for the CNN communication network.

In Table 6.4, we show the number of parameters for a CNN and a fully connected NN variant of the communication network. For both networks, we use a 3-layer deep architecture, where the spatial dimensions of the input feature maps are preserved throughout the layers, and the number of in- and output feature maps is equal. We vary the input number of feature maps $K, K \in \mathbb{N}_{>0}$ and the spatial dimensions ($S \times S$ pixels, with $S_{>0}$) in the architecture. A schematic version of these architectures is shown in Figure 6.6.

# of feature maps (C_{in})	**# of params (CNN)**	**# of params (FC-NN)**		
spatial size ($S \times S$ pixels)	2×2 / 4×4 / 8×8	2×2	4×4	8×8
1	78	60	816	12,480
2	306	216	3168	49,536
4	1,212	816	12,480	197,376
8	4,824	3,168	49,436	787,968

Table 6.4: Number of parameters for an example CNN communication network and an example fully connected communication network. Note that the number of parameters for different spatial input sizes is equal for a CNN architecture (as the number of parameters in a CNN kernel is independent of input size).

We can derive the number of parameters for convolutional and fully connected layers as follows. We consider an input consisting of K feature maps with spatial dimensions $S \times S$ pixels. The kernel size in the convolutional layer is denoted by k, and the numbers of input channels and output channels are denoted by C_{in} and C_{out}, respectively. Then, the number of weights in a convolutional layer is given by the following two equations

$$\begin{aligned} \text{\# of weights in convolutional layer} &= C_{in} \times k^2 \times C_{out} + C_{out} \\ \text{\# of weights in FC layer} &= C_{in} \times S^4 \times C_{out} + C_{out} \times S^2 \end{aligned} \tag{6.4}$$

Using these formulas and setting the number of output channels to the communication network equal to the number of input channels ($C_{in} = C_{out}$), we can compute the numbers of parameters for a fully connected and a convolutional communication network. Table 6.4 shows that the number of parameters of a CNN is constant for varying spatial dimensions and increases with $\mathcal{O}(C_{in}^2)$, whereas the fully connected communication network shows a dependence of the input data size of $\mathcal{O}(M^2 \cdot S^4)$.

As we want to consider ultra-high-resolution image data in this work, we need to work with feature maps of sizes that are quite large. For example, if we extract 512×512 pixel image patches from the data, and use a 4-layer deep U-Net (consisting of 4 down-sampling and up-sampling layers), the feature maps in the bottleneck have spatial dimensions of 32×32 pixels. This would lead to immense numbers of parameters if we would use a fully connected network. Therefore, it was decided to use a CNN architecture as a communication network for this work.

6.3. Memory usage of the U-Net and the proposed model architecture

In this section, we investigate the memory requirements for a traditional baseline U-Net and compare these to our proposed model to motivate the design of our model in terms of memory usage. We analyze the memory usage, both from a more theoretical and a more experimental perspective in this section.

6.3.1. Theoretical memory usage during training and inference

When training a neural network, the device (GPU) handling the training process must retain specific variables for successful forward and backward propagation. Since the gradients of the weights in a layer depend on the layer's outputs, storing the intermediate feature maps of *all* convolutional layers in the network during training is necessary. Additionally, memory storage is required for the input image, model weights and biases, and the optimizer state, which encompasses the moving averages and squared gradients for each trainable model weight in the Adam optimizer [39]. Furthermore, true segmentation labels, maximum pooling indices, and batch normalization statistics must be stored for successful backward propagation. Not all intermediate convolutional outputs need to be retained during inference (evaluating the model without updating parameters). However, preserving the outputs of the last layer in each encoder block is necessary due to skip connections in the U-Net architecture that transfer those feature maps from the encoder to the decoder.

Name	**Size**	# of input channels	# of output channels	memory feature maps (# of values)	memory feature maps (MB)	memory weights (# of values)	memory weights (MB)
input	1,024	3	3	3.2M	12,0	-	-
input block	1,024	3	64	268M	1,024.0	38,848	0.148
encoder block 1	512	64	128	167M	704.0	221,696	0.846
encoder block 2	256	128	256	84M	352.0	885,760	3.379
encoder block 3	128	256	512	42M	176.0	3,540,992	13.508
encoder block 4	64	512	1,024	21M	88.0	14,159,872	54.016
communication network	64	256	1,024	30M	116.0	4,917,504	18,750
decoder block 1	64	1,024	512	50M	192.0	9,177,088	35.008
decoder block 2	128	512	256	101M	384.0	2,294,784	8.754
decoder block 3	256	256	128	201M	768.0	573,952	2.189
decoder block 4	512	128	64	402M	1,536.0	143,616	0.548
output block	1,024	64	3	3.1M	12.0	195	0.001
labels	1,024	3	3	1.0M	8.0	-	-
optimizer state (encoder)	-	-	-	-	-	37,694,336	143.792
optimizer state (decoder)	-	-	-	-	-	24,379,270	93.000
optimizer state (comm.)	-	-	-	-	-	9,835,008	18.759

Table 6.5: Memory and weight analysis of 4 deep U-Net architecture and a 3-layer communication network with 5×5 convolutions, during training for a 1024×1024 RGB image and a 3-class segmentation task. The table shows sizes, input/output channels, memory usage (number of values and megabytes), and weight count/size (number values and megabytes) for each encoder/decoder block and the communication network. Communication network memory is for 4 subdomains with 256 feature maps each. Optimizer variables are computed using the Adam optimizer.

Considering the often limited memory capacity of training devices, typically 16GB or 32GB for modern GPUs, assessing the memory demands of a model architecture is important. We evaluate the memory requirements for the baseline U-Net and our model, comparing memory requirements for a 4-block deep baseline U-Net (see Figure 3.1) with a 1024×1024 pixel input image. Since our proposed model's encoder and decoder networks share properties with those of U-Net, we focus on analyzing the additional memory needed for the communication network. The theoretical memory analysis utilizes Equation 6.4, with comparisons made to measurements on a GPU system using the `py-torch` memory profiler [52].

Table 6.5 shows the theoretical weight and memory demands for a U-Net encoder and decoder network during training, along with the proposed communication network. The table shows that storing feature maps requires significantly more memory than storing model weights. Furthermore, note that shallow layers like the input block, first decoder block, and last decoder block collectively account for nearly half of the total memory allocation for feature maps. Layers that are deeper into the encoder or decoder, require much lower storage for the output feature maps. Conversely, the number of weights increases deeper in the encoder and decoder due to the higher number of kernels and larger kernel size in deeper blocks. The communication network shows low memory requirements, highlighting the efficiency of the model proposed in this thesis regarding memory usage. On the other hand, the number of weights in the communication network is relatively large.

6.3.2. Memory allocation profile

The theoretical memory analysis presented in Table 6.5 indicates the memory required for training a neural network. However, in practice, the actual memory used is even slightly higher due to temporal allocations of memory during, for example, convolutional operations. To illustrate this, we use the `torch` functionality to record the memory allocations over time during training for 5 iterations, with 1024×1024, both for a baseline U-Net (on one GPU), see Figure 6.8, see and the proposed communication network (on 2 GPUs), see Figure 6.9.

Using the interactive `torch` profiler, which provides more information on the different allocated memory blocks, their size and the operations that form these blocks, we can make the following observations. This information is used to generate an exemplary memory profile in which we indicate some relevant regions both in time and memory direction in Figure 6.7.

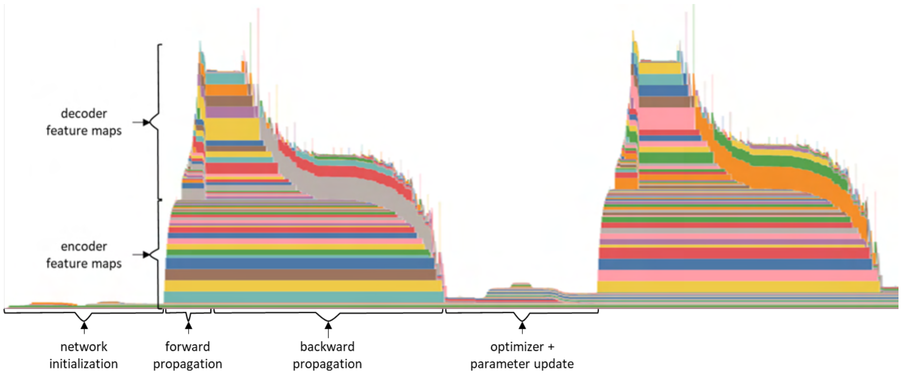
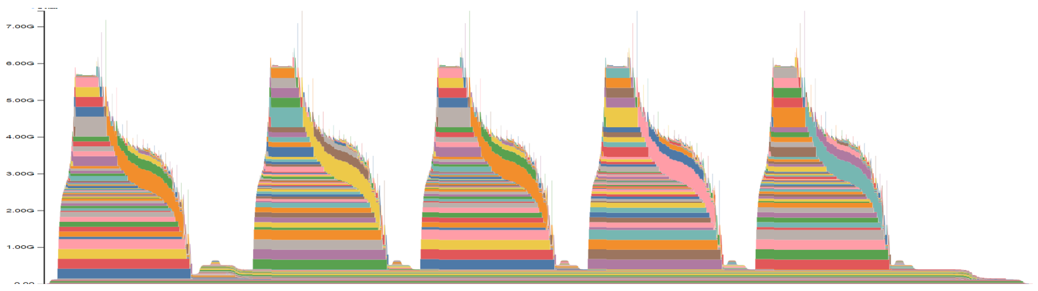


Figure 6.7: Example memory profile for a U-Net. The training phases (initialization, backward and forward propagation, and optimizer and parameter updates) are indicated in the horizontal time direction, whereas the feature map distribution for forward and backward propagation is shown in the vertical memory direction..

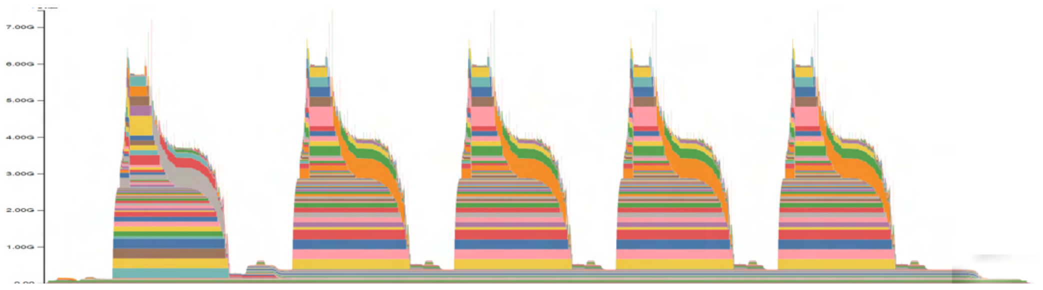
A comparison between the memory profiles for the baseline U-Net and the proposed model gives us the following observations.

1. The peak memory, both for the baseline model and the proposed model, is attained right after the forward propagation step or at the beginning of the backward propagation. The short, sharp peaks result from very short, temporary blocks of memory allocated for performing convolutions.
2. For the proposed model, we only see memory allocated for weight and parameter updates on GPU0. This is expected, as those updates are only done on GPU0, which collects all the gradients and updates the weights. After updating these, the new weights are synchronized to the other GPU.
3. There is no significant difference in memory usage between GPU0 and GPU1 for the proposed model and GPU0 of the baseline model. This shows that the communication network does not add much extra memory load to the segmentation, as expected from Table 6.5.

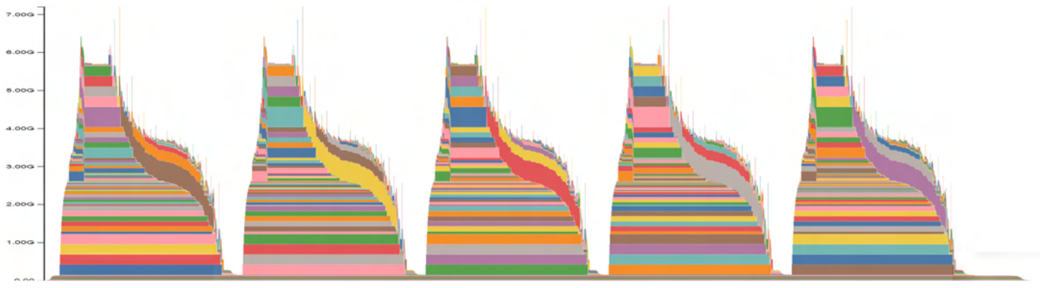


(a) GPU0

Figure 6.8: Memory allocations for the baseline U-Net plotted against time during 5 training iterations (forward and backward propagation) for a 1024 × 1024 input image. Blocks of different colors represent various variables (feature maps or parameter groups). The height of each block indicates the memory size, while the width corresponds to the duration of allocation.



(a) GPU0



(b) GPU1

Figure 6.9: Memory allocations for the proposed model plotted against time during 5 training iterations (forward and backward propagation) for a 1024 × 1024 input image. Note that this model was trained on 2 different GPUs. GPU0 contains the communication network and one encoder-decoder network. GPU1 only contains an encoder-decoder network. Optimizer state and parameter updates are only performed on GPU1.

Theoretical and experimental memory usage for different image resolutions

Lastly, we investigate how the memory usage for both a baseline U-Net and the proposed model scales when we increase the image size, both for the theoretical estimations such as in Table 6.5 as experimental values. To do so, we plot the theoretical memory usage prediction for a baseline U-Net architecture and the peak memory usage for a GPU containing both this U-Net encoder-decoder architecture and the proposed communication network across various image resolutions in Figure 6.10.

This figure clearly shows a discrepancy of a constant factor between the theoretical and measured memory. This discrepancy can be attributed to the temporal memory allocated for convolutional operations and parameter management within the implemented `torch` framework.

For smaller image resolutions, memory is primarily used to store model weights and the optimizer state. However, as image size surpasses $2^7 \times 2^7$ pixels, memory predominantly goes to feature maps. Furthermore, note that the peak memory scales quadratically with resolution; doubling the resolution results in fourfold memory consumption. The communication network uses little memory compared to the memory used by the encoder and decoder networks, especially for large image sizes.

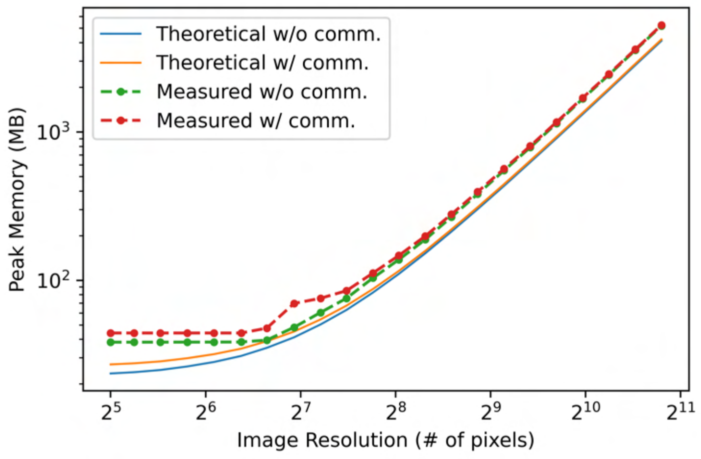


Figure 6.10: Theoretical and experimental peak memory of the U-Net encoder and decoder for different square image resolutions. The communication network processes 64 feature maps in this example.

This shows the effectiveness of our approach: using this allows us to process images with equal sizes as a baseline U-Net per GPU device, but additionally, the communication network adds communication between the devices, ensuring global communication between all image subdomains.

7 Experimental Results

As outlined in the introduction, this study aims to introduce a novel approach for memory-efficient image segmentation by leveraging a domain decomposition-inspired strategy. To achieve this, we developed a new model architecture, building upon the successful U-Net framework, and trained it for segmentation tasks. In this chapter, we evaluate the performance of our proposed model architecture on three different datasets: a synthetic dataset, the Inria Aerial Dataset, and the DeepGlobe dataset. Our evaluation compares the proposed model against a baseline U-Net model in terms of both memory efficiency and accuracy.

7.1. Implementation Details

In this section, we will define the general set-up of the models and training procedure. Dataset- and experiment-specific hyperparameters and training settings will be provided later. All code for the model and generating synthetic datasets is available on GitHub [link to be added].

7.1.1. Models

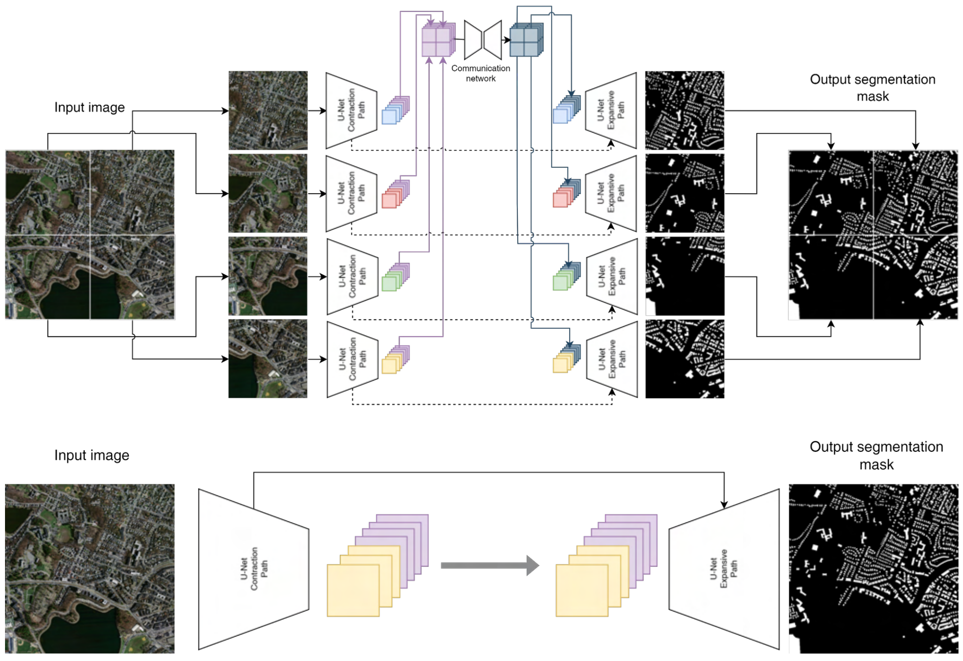


Figure 7.1: The models trained for evaluation: a baseline U-Net (bottom row), and the proposed model with a communication module (bottom row)

For each dataset, we assess the memory efficiency and accuracy metrics of three different models:

1. **A traditional U-Net**: This baseline model, based on the architecture presented in Ronneberger et al. (2015) [55], takes either the full image or large image patches as inputs.
2. **Proposed network architecture with communication**: Our proposed architecture, depicted in Figures 5.1, 5.2, and 5.3, comprises multiple subnetworks and a communication network. All subnetworks share their weights.
3. **Proposed network architecture without communication**: Similar to the previous architecture, but without a communication network. In this setup, no information exchange occurs between subnetworks. This model is included to see the effect of the communication module.

Architectural hyperparameters. We customize the architecture of the communication module and the subnetworks according to the specific dataset, adjusting the following architectural hyperparameters.

1. **Input data size of subnetworks**: The spatial dimensions of the input data for the network. This input data size also determines the size and number of the subdomain.s
2. **Depth of subnetwork**: The number of up-sampling and downsampling blocks in the encoder and decoder network varies based on the dataset.
3. **Number of channels in subnetworks**: We adapt the number of channels in the encoder and decoder paths to suit the dataset.
4. **Number of feature maps K sent to communication network**: To limit the communication network's parameters, we only send a fixed number of (encoded) feature maps from the bottleneck to the communication network.
5. **Encoder Selection**: For the DeepGlobe dataset, we utilize a pre-trained encoder network, specifically the ResNet-18 architecture [25], instead of the conventional U-Net encoder path.
6. **Dropout rate and batch normalization momentum**. The hyperparameters dropout rate and the batch normalization momentum parameters β_1, β_2 need to be defined by the user.

7.1.2. Training settings

Platform. Training for the synthetic dataset was performed on the DelftBlue supercomputer [1], which contains, among others, NVIDIA V100 Tensor Core GPUs with 32 GB video RAM each. Experiments are conducted using the PyTorch framework [52], with version `torch 1.12.1`. Training for the realistic datasets was done on the data science and machine learning competition platform Kaggle, which provides free NVIDIA Tesla P100 GPUs to its users, with PyTorch version `2.1.2`. To evaluate the memory usage of the different models, we use the `torchsummary` library [69], which provides us with information on the model parameters, forward and backward pass. As the number of available GPUs was often limited to one, parallel training was performed on *one* GPU. The proposed network was simulated by choosing a relatively small patch size such that the full proposed network fits on one GPU. Unfortunately, this means that we cannot give exact measurements for comparisons in runtime between the proposed approach and other approaches.

Loss Function. For training, we utilized the Dice Loss (DL) function, as this function deals with the class imbalance for semantic segmentation of images [33, 45]. The Dice Loss is defined as

$$DL = 1 - 2\frac{\sum_{p=1}^{P}\sum_{k=1}^{K} y_{k,p}\hat{y}_{k,p} + \epsilon}{\sum_{p=1}^{P}\sum_{k=1}^{K} y_k + \sum_{p=1}^{P}\sum_{k=1}^{K} \hat{y}_k + \epsilon}. \tag{7.1}$$

Here, K represents the number of classes, P is the total number of pixels; $\hat{y}_{k,p}$ is the predicted probability of pixel p for class k (obtained by applying a Log-Softmax function to the model's output so that all the output logits are in the range $[0, 1]$); $y_{k,p}$ denotes the true probability of pixel p for class k, with values restricted to $0, 1$ as the true mask is known, and ϵ serves as a small smoothness constant to avoid dividing by zero, set to $\epsilon = 10^{-7}$ in this paper.

Optimization and Learning Rate Scheduling. We employed the Adam optimizer [39], see Chapter 2, with momentum parameters $\beta_1 = 0.9, \beta_2 = 0.999$. Furthermore, we use a plateau learning rate decay strategy, where the learning rate decays with a factor of 2 when validation loss does not decrease, maintaining patience for a chosen number of epochs.

Metrics. To evaluate the models' performance, we use the losses defined in Chapter 2, namely the pixel accuracy (Equation (2.1)), (mean) IoU score (Equations (2.2) and (2.3)), recall (Equation (2.4)) and precision (Equation (2.5)).

7.2. Experiments on the Synthetic Dataset

We begin by applying our proposed model to the synthetic dataset, which was introduced in Section 5.3.1. This dataset is tailored for evaluating the model's performance in predicting the location of line segments. This dataset comprises 12,000 randomly generated grayscale images for each image dimension within the range of $32k \times 32$ pixels, where $k \in \{2, 3, 4, 8, 16\}$.

For each image dimension, we partition the dataset into training, validation, and test sets, consisting of 8,000, 2,000, and 2,000 images, respectively. Our focus lies on assessing the models' ability to accurately predict the locations of line segments. This evaluation is crucial as it directly correlates with the effectiveness of the communication module in transferring global information between subdomains.

We answer the following four questions using the synthetic dataset.

1. To what extent does the proposed model facilitate the transfer of global information through the communication network?
2. What is the relationship between the precision and accuracy of the predicted segmentation maps and the complexity and size of the subnetworks?
3. How are the precision and accuracy of the predicted segmentation maps influenced by the complexity and size of the communication network?
4. Is it feasible to train the proposed model using images of one size and then evaluate its performance on images of a different size?

In the following sections, we will describe the synthetic dataset in more detail and answer these questions.

7.2.1. Network parameters

Due to the simplicity of the synthetic dataset, a simple model architecture can already make accurate predictions for this segmentation task. We test a large variety of networks in terms of the depth and complexity of the communication model. In Table 7.1, the model configuration parameters depth, channel distribution, bottleneck size, and number of parameters are shown for the subnetworks and the communication network utilized in these experiments. In these configurations, the initial entry in the channel distribution represents the input gray-scale channel, whereas the final channel corresponds to the logits channel for the three classes (background, line segment, and circle). We note here that the size of the networks is independent of the spatial dimensions of the input size, as the used networks are fully convolutional, and convolution kernel sizes are independent of input size.

	Depth	**Channel distribution**	**Bottleneck size**	**# of parameters**
Subnetworks	2	1-4-8-16-16-8-4-3	8×8	7,487
	3	1-4-8-16-32-32-16-8-4-3	4×4	30,470
	4	1-4-8-16-32-64-64-32-16-8-4-3	2×2	122,031
	# of feature maps communicated	**Channel distribution**	**Kernel size**	**# of parameters**
Communication network	1	1-1-1-1	5×5	84
	2	2-2-2-2	5×5	318
	4	4-4-4-4	5×5	1,236
	8	8-8-8-8	5×5	4,872
	16	16-16-16-16	5×5	19,344
	32	32-32-32-32	5×5	77,088
	64	64-64-64-64	5×5	307,776

Table 7.1: Properties of the used subnetworks and communication networks for the synthetic dataset.

The baseline U-Nets for this experiment are configured in the same way as the subnetworks of the proposed model, using the same depth and channel configurations. The same holds for the proposed model *without communication*.

7.2.2. Receptive field size

Before we start looking at the data, we compare the proposed model and the baseline model in terms of their receptive field size (see Chapter 2). The theoretical size of this field can be computed using the theory outlined in [44]. The receptive field is shown as a function of the depth (the number of up-sampling and downsampling blocks) of the U-Net. Note that, for the baseline model and the model without communication, the receptive field size is the same, as they make use of the same U-Net architecture.

However, for cases where the subdomain size is smaller than the receptive field size, this will limit the receptive field for the model without communication, whereas for the baseline model, this is no limit (since this model operates on the full-resolution image without subdomains). Also, close to the boundary, the receptive field will also be smaller.

Model / Depth	2	3	4
Baseline U-Net	44×44	92×92	188×188
Without communication	44×44	92×92	188×188
Proposed model	92×92	188×188	380×380

Table 7.2: Receptive field size for different model depths for baseline model, proposed model with and without communication.

The communication network increases the receptive field size significantly, as can be observed in Table 7.2. This was expected, since the communication networks add convolutional operations on the coarsest level of feature maps, where each value represents a large region of the input image, causing a fast increase of the receptive field in only a few iterations.

7.2.3. The transfer of global information

Before starting with detailed results of the network architecture, we show for a small, minimal example that the proposed network architecture is indeed able to transfer information between different local subdomains. We extensively discuss the results for this dataset, to help the reader to understand the following sections more easily. To do this, we show in this subsection the results for a simple 2-subdomain synthetic problem. Two example images and corresponding masks are shown in Figure 7.2.

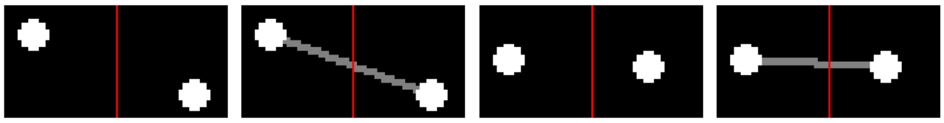


Figure 7.2: Two 64×32 pixel example image-mask pairs from the synthetic dataset. The subdomains have size of 32×32 pixels, their border is denoted by the red line in the horizontal center of the images and masks.

We train the proposed model, a baseline U-Net, and the proposed model without a communication module on this dataset, using the hyperparameters shown in Table 7.3. All models utilized in this section have a depth of 3, resulting in a bottleneck size of 4×4 pixels and a receptive field of size 188×188 pixels (see Tables 7.1 and 7.2). In this subsection, we only use a *communication module set-up* with 16 feature maps communicated. As mentioned before, a correct segmentation of the line segments is only possible when the network on one side of the model knows the circle's position on the other side. Therefore, we expect to see clear differences in results for the models with and without communication modules.

The training and validation loss values throughout the training process are illustrated in Figure 7.3. All models eventually converge to a stable loss value. Notably, the model lacking a communication module converges to a notably higher loss compared to the baseline and proposed models. Additionally,

Hyperparameters	
(max.) learning rate	0.008
(max.) number of epochs	40
early stopping patience	8 epochs
learning rate decay patience	3 epochs
minimum stopping epoch	20
dropout rate	0.1
batch size	16

Table 7.3: Hyperparameters used for synthetic data experiments

we notice a peculiar difference between the train and validation losses. Typically, the validation loss is expected to be equal to or higher than the training loss in neural network training scenarios. However, in our case, due to the inclusion of dropout layers (applied only during training) and batch normalization (utilizing batch mean and variance during training but learned mean and variance during inference), the validation loss remains lower than the training loss.

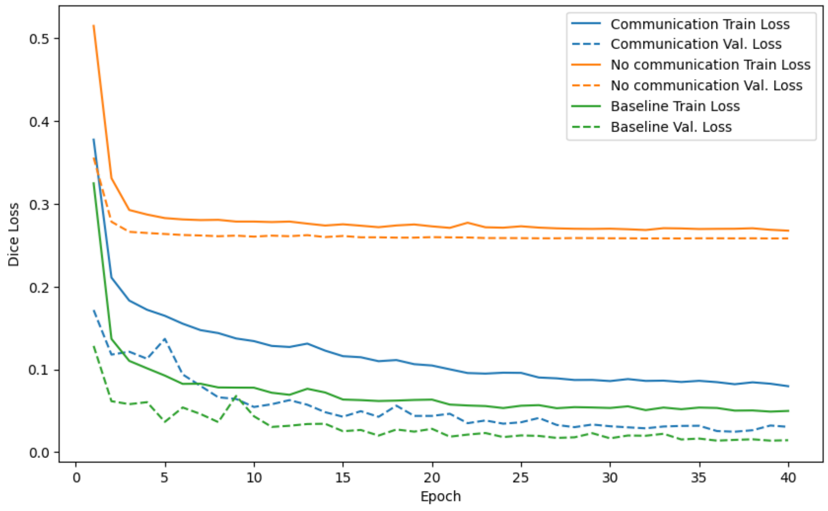


Figure 7.3: Training and validation loss for the baseline U-Net, proposed architecture, and proposed architecture without communication module.

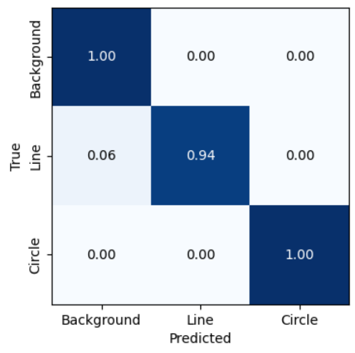
Now, we compare the metrics for the predictions generated by these three models. For extra visualization, we also depict the confusion matrices of the three models in Figure 7.4. In Table 7.4, the evaluation metrics for the three different models are shown.

It is obvious from these results that the proposed model with communication module performs much better, almost comparable to the baseline U-Net, compared to the baseline model without communication. This supports our approach and shows that the used communication module can transfer information through the coarse-grained feature maps in the bottleneck layer of the U-Net subnetworks.

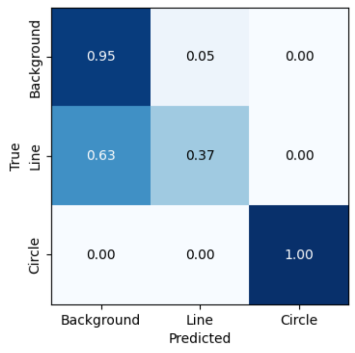
model	mean IoU	precision	recall	accuracy
ours w/ communication	0.952	0.908	0.942	0.996
ours w/o communication	0.692	0.190	0.371	0.939
baseline U-Net	0.973	0.958	0.961	0.998

Table 7.4: Metrics for the line segment class for the three different models on a 2-subdomain synthetic dataset problem.

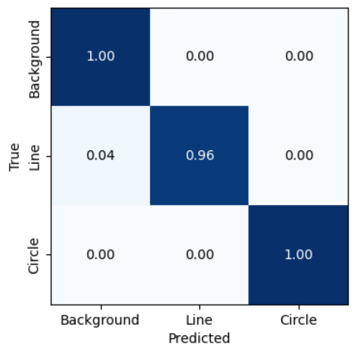
To get a better understanding of the meaning of the shown results, we depict the true and predicted masks for two images in the synthetic dataset in Figure 7.5. We see that the difference between true mask and masks predicted by the baseline model and proposed model is very small, as was already



(a) Confusion matrix for proposed model w/ communication



(b) Confusion matrix for proposed model w/o communication



(c) Confusion matrix for the baseline U-Net

Figure 7.4: Normalized confusion matrices for the three trained models on the 2-subdomain synthetic dataset.

expected from the metrics. However, for the model without communication, the direction of the line segments in one subdomain is not towards the circle in the other subdomain. Furthermore, both sides of the circles contain the beginning of a line segment, indicating that the model is not aware of its relative position (left or right from the other sub-image). The large regions of pixels (incorrectly) predicted as line segment lead to a low precision value, whereas the wrong orientation and missing pixels lead to a low recall metric.

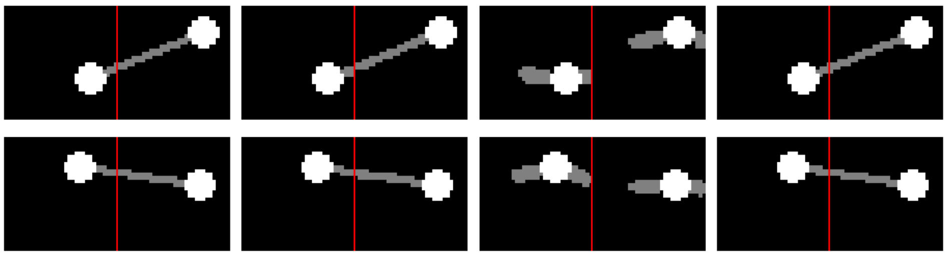


Figure 7.5: From left to right: true mask, mask predicted by proposed model w/ communication, w/o communication, and the baseline U-Net. The red vertical line in the horizontal center of the image indicates the subdomain border.

In conclusion, we can say that the results for this simple, 2-subdomain large synthetic problem show that the proposed architecture can communicate global information across subdomains. In this section, the results for the model with the communication module were slightly better than for the baseline U-Net model. In the next section, we investigate the relation between the number of communicated feature maps and the accuracy of the predictions.

7.2.4. Relationship between model performance and model complexity

In the proposed architecture, two model hyperparameters intuitively could lead to a boost in the performance of the communication model result: (1) increasing the complexity of the subnetwork and (2) increasing the amount of communication. In this subsection, we investigate the effect of both changes on the model performance. First, we discuss the experimental setup for both approaches, then we present the results for both simultaneously.

Increasing the model complexity. We modify the depth (the number of successive down-sampling or up-sampling blocks) of the subnetwork or baseline U-Net architecture to measure the effect of an increasing number of parameters. Also, note that the receptive field size increases fast if we make the U-Net deeper. On the other hand, adding down-sampling layers leads to smaller spatial dimensions of the bottleneck layer, that is, a coarser feature map. For deep subnetworks, the question is if this very coarse spatial resolution can effectively capture the positional information needed by the communication network. We train networks with depth 2, 3 and 4. As we can see in Table 7.1, the spatial dimensions of the bottleneck are for these depths 8×8, 4×4 and 2×2 pixels, respectively.

Increasing the number of communicated feature maps. To enlarge the number of communication variables between the different subdomains, we increase the number of feature maps that is communicated. Note that the maximal amount of feature maps for communication is limited by the number of feature maps in the bottleneck layer, and this number is determined by the depth of the subnetwork. We choose the number of feature maps M communicated to and by the communication from the set $\{1, 2, 4, 8, \ldots, M_{\max}\}$, with $M_{\max}$ for a given depth determined by $M_{\max}(\text{depth}) = 4 \cdot 2^{\text{depth}}$.

To compare the results for different models, we train all of the different complexity configurations utilizing the training hyperparameters as shown in Table 7.3. All models are trained only on the synthetic dataset with **3** subdomains (the reason for this choice will become clear in the next subsection). For each model, we compute the IoU score per class. Furthermore, we train a baseline U-Net with depths 2, 3, and 4 to get a result for comparison with the baseline model.

The results of these experiments are shown in Figure 7.6.

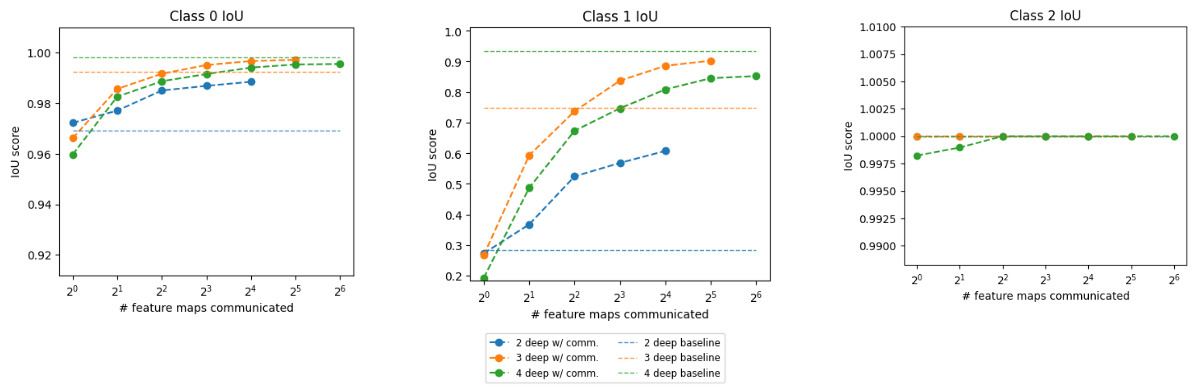


Figure 7.6: IoU score for different depths and numbers of communicated feature maps. From left to right: IoU scores for class 0 (background pixels), class 1 (line segment pixels), and class 2 (circle pixels). Note that the last dot of the dashed lines corresponds to the highest possible number of communicated feature maps for an architecture with a given complexity.

In Figure 7.6, several noteworthy observations can be made. Firstly, for the baseline model, the highest IoU score is achieved with the most complex architecture, namely a 4-deep baseline U-Net. Conversely, for the proposed model incorporating communication, the optimal IoU score is achieved with a 3-deep architecture. Even with a lower number of communicated feature maps, the 3-deep architecture with communication surpasses the performance of the 4-deep architecture with communication. Notably, both the 2-deep and 3-deep models with communication outperform their corresponding baseline models, provided that a sufficient level of communication is maintained.

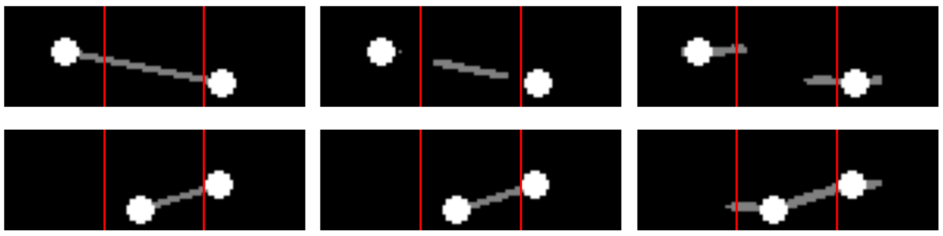


Figure 7.7: From left to right: true mask, the mask predicted by the proposed model (depth 2) with communication, and the mask predicted by the baseline U-Net (depth 2). The subdomain borders are indicated by the red vertical lines in the image.

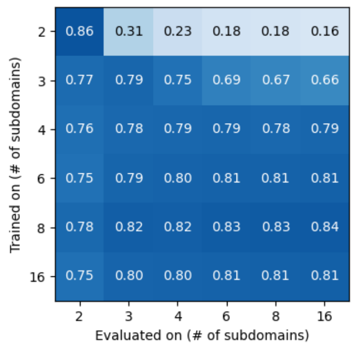
These findings raise an important question regarding how the proposed architecture may outperform the baseline U-Net in certain scenarios. To address this question, we need to revisit the theory of the receptive field, defined as the region of the input space influencing the output prediction at a specific point. The size of the receptive field expands with an increase in the number of convolutional layers, particularly when incorporating down-sampling layers followed by convolutional operations.

As shown in Table 7.2, the receptive field of a model with a communication module is more than twice as wide and high as the receptive field of the baseline model. Notably, the receptive field size of a 2-deep baseline model is only 44×44 pixels. However, since the dataset has dimensions of 96×32 pixels, pixels more than 44 pixels apart from either or both of the two circles do not receive sufficient information on the position of the circle(s), leading to inevitable mistakes in segmentation. Some examples of bad segmentation results are shown in Figure 7.7. We observe in these figures the results of the circles being too far apart from each other for the receptive field size, especially for the baseline model: the segmentation result is only correct for the line segment part that is close to both circles. For the line segment further apart from one of the circles, the line segment is classified as a background pixel, or background pixels are classified as a line segment.

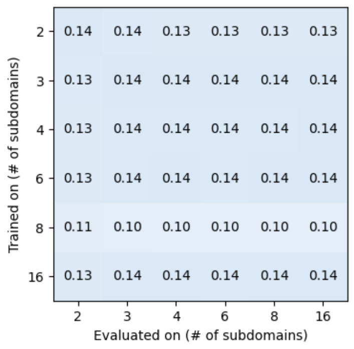
7.2.5. Training and evaluating on different numbers of subdomains

Even when we have several GPUs available, it may be the case that the input size of the data is still too large to divide it over the available devices. Then, a relevant question is if the proposed model can be trained on (large) image patches, that are subsequently divided over the different GPUs. We first remark here that our model does allow for such an approach due to its fully convolutional architecture - convolutional operations are not bound to a fixed input size.

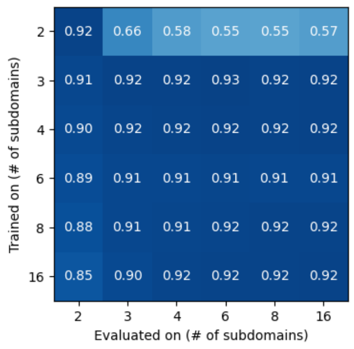
To test the performance of the model when trained on a different number of subdomains, we train a model using the dataset with spatial dimensions of $32k \times 32$ pixels, with $k \in \{2, 3, 4, 6, 8, 16\}$, and subsequently evaluate the models on the datasets with other image dimensions. We do this for the proposed model with communication, the proposed model without communication, and the baseline U-Net, all with an encoder and decoder depth of 4, and with 16 feature maps communicated to and from the communication module. Again, we use the same training hyperparameters as shown in Table 7.3. Important to remark here is that we can evaluate the model on other numbers of subdomains *without further training or fine-tuning*. If it is possible to train on a smaller number of subdomains and evaluate on a larger number, this can significantly help to work with limited resources and speed up training.



(a) IoU scores for model w/ communication



(b) IoU scores for model w/o communication



(c) IoU scores for baseline model

Figure 7.8: IoU score for the line segment class for the model with and without communication module, and the baseline U-Net. The y-axis shows the number of subdomains on which the model was trained, and the x-axis the number of subdomains the model was evaluated on.

The results of this experiment (see Figure 7.8) show that the baseline method as well as the proposed model with communication module is also usable for larger numbers of subdomains. Only for the case where the initial model is trained on 2 subdomains and evaluated on larger numbers of subdomains, we see a lower IoU score. Probably, this is because the length of the line segments seen during training is limited by the size of the two subdomains, whereas for larger numbers of subdomains, this limit is higher, leading to longer line segments.

Despite these results, the question can be asked if these results generalize to arbitrary datasets. For the synthetic dataset, the contextual information for making the correct prediction (the position of the two circles was imposed to be maximally apart with one subdomain in-between (so maximally 96 pixels), see Section 5.3.2. If a segmentation task requires truly global information, including information from the whole image might be necessary. Then, training on (large) image patches will lead to less accurate results.

7.2.6. Memory Requirements

This last section compares the memory requirements for the proposed model and the baseline U-Net model. The input and pass sizes were computed using the library `torch-summary` [69]. We computed the memory requirements for forward and backward pass, input, and parameters for an example *RGB* image of spatial dimensions 512 × 512 pixels, both for a standard, non-parallel, baseline U-Net model and our approach split over N GPUs. The results are shown in Table 7.5.

	Classical U-Net	**Our approach (split on N GPUs)**
Number of params.	1,927,042	1,954,978
Input size (MB)	3.00	3.00 / N
Forward/backward pass size (MB)	2404.00	2404 / N + 4.25
Params. size (MB)	7.35	7.5
Estimated total size (MB)	2414	2407 / N + 11.75

Table 7.5: Memory requirements per GPU for a conventional U-Net and our approach. Parameters are computed for an input image of dimensions 512 × 512 × 3 pixels

This table shows that the forward and backward pass memory requirements for our approach scale are almost inversely linear with the number of GPUs available. The only parameter that does not scale inversely linear is the number of parameters. This is because all subnetwork weights are required at every GPU, also when we also partition the full problem into sub-problems. However, as visible in Table 7.5, the largest amount of memory is not required for the parameters, but for the forward and backward pass size.

7.3. Experiments on the Inria Aerial Image Dataset

After testing and training the proposed model on a synthetic dataset that is specifically designed for this network, it remains an open question whether the proposed model can also operate successfully on out-of-the-wild segmentation tasks. The question is if the communication network - which operates on the coarse, bottleneck, level - can communicate all relevant information to its neighboring subdomains. To test this, we use the Inria Aerial Image Dataset [47]. This dataset has been specially designed to be able to generalize to different kinds of urban settlements by choosing satellite images from various regions of the world. The dataset has been used as a benchmark dataset in several works (see e.g. [26, 36, 70]).

	Module	**Channel distribution**	**Bottleneck size**	**# of parameters**
Subnetwork	down-sampling path	3-32-64-128	16 × 16	1,163,008
	up-sampling path	128-64-32-2	16 × 16	753,760
	# of feature maps communicated	**Channel distribution**	**Kernel size**	**# of parameters**
Communication network	64	64-64-64-64	3 × 3	111,168

Table 7.6: Properties of the used subnetworks and communication networks for the Inria Aerial Image Dataset.

7.3.1. Network parameters

For training, we employ the proposed architecture, together with a baseline model and an architecture without a communication module. However, instead of the 32 × 32 pixel subdomains we used for the synthetic dataset, we now consider 128 × 128 pixel subdomains, corresponding to approximately 40 × 40 m^2 areas. Other network hyperparameters are shown in Table 7.6. Note that the size of the encoder-decoder (sub)network is the same for the baseline model, the proposed model with and without communication. The only difference between the model with communication and the other models is the additional parameters for the communication network.

7.3.2. Training hyperparameters and training procedure

We train the models described in the previous section using the Adam optimizer. As a loss function, we again use the multi-class Dice Loss function Equation (7.1). The hyperparameters for training are shown in Table 7.7. Note that we did not use dropout layers for this dataset, as we did not have problems with over-fitting for this task, probably due to the large number of training samples (approximately 11k) combined with data augmentation.

Hyperparameters	
(max.) learning rate	0.001
(max.) number of epochs	40
early stopping patience	-
learning rate decay patience	-
minimum stopping epoch	-
dropout rate	0.0
batch size	16

Table 7.7: Hyperparameters used for training the model on the Inria Aerial Image Dataset

Before presenting the results, it is crucial to outline the training methodology employed for this dataset, given its large spatial resolution (5000×5000 pixels). Due to memory constraints, particularly with batch sizes larger than 1 on GPU devices, pre-processing the dataset was necessary. We provide a summary of the pre-processing and traing procedure followed here.

- **Pre-processing**
  - Data is initially partitioned into non-overlapping patches of 512×512 and 2048×2048, as discussed in Chapter 5.
  - During training, we use data augmentation techniques to augment the training dataset size. We employ random horizontal flipping and random vertical flipping (both with probability $p = 0.5$) and random rotation of θ radians, with $\theta \in \{0, \pi/2, \pi, 3\pi/2\}$ and the probabilities for each θ equal to $p = 0.25$.
- **Training**
  - During training of the proposed model, both with and without a communication module, 512×512 patches are utilized for training. These patches are further subdivided into 4×4 non-overlapping subdomains, each with spatial dimensions of 128×128 pixels.
  - The baseline U-Net model is trained using full 512×512 image patches. Notably, the U-Net architecture is fully convolutional and therefore agnostic to input size.
- **Validation and testing**
  - For validation and testing, we use the 2048×2048 image patches, as these are the largest that fit in the device (GPU) (when we do not store gradients).
  - Validation and testing are performed on 2048×2048 patches to minimize (global) boundary effects. For the proposed model (with and without communication), this entails splitting the validation/test image into 16×16 subdomains.
  - As the baseline model is also fully convolutional, it can be directly evaluated on the full 2048×2048 validation/test image, having been trained on 512×512 patches.

7.3.3. Results

After training the model, we evaluate the model on training, validation, and test datasets. First, we show some predictions on 512×512 patches (corresponding to 4×4 subdomains) from the test dataset generated by the three models in Figure 7.9.

In Figure 7.9, we observe several interesting phenomena. Firstly, the efficacy of the communication network in ensuring consistency in building prediction around boundaries is clearly visible. Notably, if

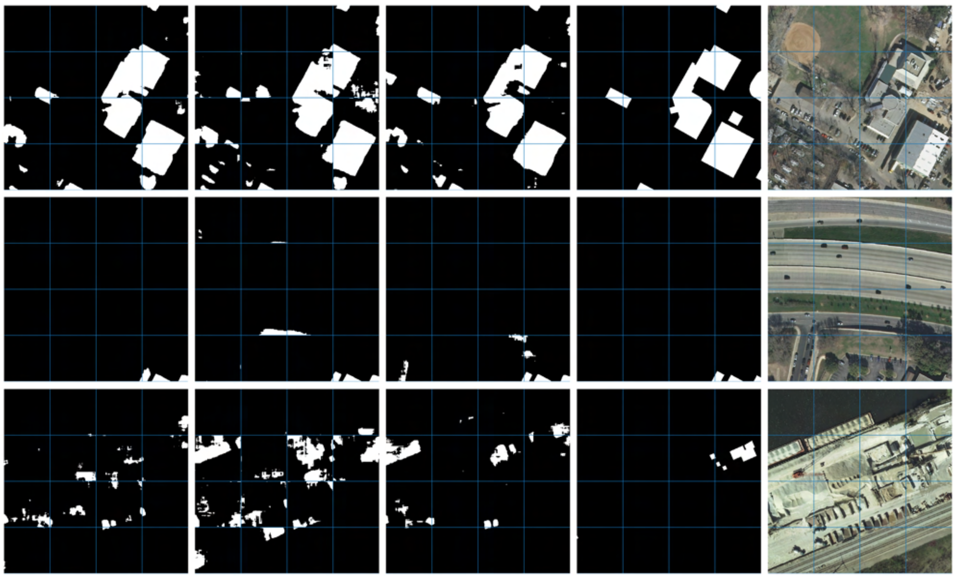


Figure 7.9: From left to right: mask predicted by the model with communication, mask predicted by the model without communication, mask predicted by the baseline U-Net, true mask, and image. The blue lines in the image indicate the 128 × 128 pixel subdomain borders.

a building is identified on one side of the boundary, the communication network anticipates the presence of a building on the other boundary side as well.

Secondly, the communication module appears to add spatial context to the subdomains. For instance, in the second row, we note that without communication, the network misinterprets the side of the road as a building. The communication module rectifies this misinterpretation. This corrective function is also visualized in the third-row image. Here, the erroneous segmentation of ships and containers as buildings is reduced, although not fully removed, by the communication module. An interesting question, which was left open for further research, is if this result could be improved by using a deeper (sub)network, with a larger receptive field, as this can take into account more global context.

Besides a qualitative look at what the networks predict and what the role of the communication network is, we also provide a more quantitative analysis of the results. This is accomplished by computing the IoU score (see Equation (2.2)) for the background and building class in the dataset. The results for this are presented in Table 7.8.

metric	**IoU (background)**			**IoU (building)**		
dataset	*train*	*val*	*test*	*train*	*val*	*test*
baseline U-Net	0.951	0.951	0.946	0.774	0.735	0.734
proposed w/ communication	**0.952**	**0.952**	**0.949**	**0.786**	**0.748**	**0.751**
proposed w/o communication	0.946	0.948	0.944	0.761	0.723	0.731

Table 7.8: IoU scores for the Inria Aerial Image Dataset

We conclude from these results that the proposed model with a communication module performs significantly better than the model without communication, and surprisingly also even better than the baseline model. This latter result might be caused by the fact that the proposed model with a communication module has a larger receptive field than the baseline model (see also Table 7.2). Furthermore, the number of parameters of the proposed model is larger than the number of parameters in the baseline U-Net (see Table 7.6, which could help to increase the predictive capacities of the proposed model compared to the U-net.

7.3.4. Feature Maps

To get more insights into what the model is learning exactly, several techniques are available. One of the most intuitive methods is to look at the feature maps generated by the different layers of the model.

In this subsection, we investigate some feature maps generated by the proposed model.

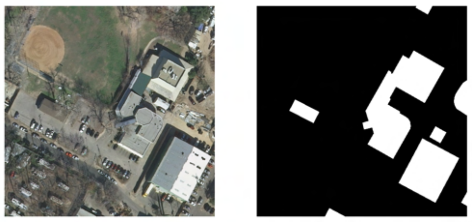


Figure 7.10: Image used to generate the feature maps for the Inria Aerial Image Dataset (512 × 512 pixels, corresponding to 4 × 4 subdomains). The corresponding mask is also shown on the right.

To compare the three models, we investigate 8 feature maps generated by two different intermediate layers of the proposed model with and without communication and the baseline model. The feature maps are generated using the image illustrated in Figure 7.10. Firstly, we examine the feature maps after the third downsampling block, see Figure 7.11. Secondly, we inspect the feature maps that are the output of the upsampling block in the subnetworks, see Figure 7.12. It is important to note here that for the proposed model, communication has not taken place yet.

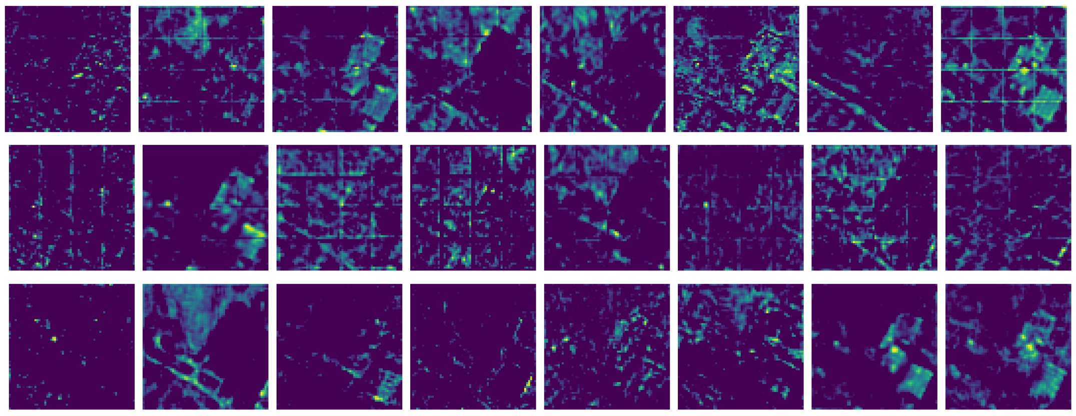


Figure 7.11: Feature maps after the third (and last) downsampling block for the proposed model with communication module (top row), without communication module (center row), and the baseline U-Net (bottom row).

Since the U-Net subnetwork architecture for all three models - with and without communication, as well as the baseline model - is similar, we can directly compare the features as shown in the feature maps. Several notable observations can be drawn from these feature maps.

1. **Effect of boundaries on the segmentation mask:** In both Figures 7.11 and 7.12, for both models with and without communication module, we observe distinct horizontal and vertical lines in the feature maps. These lines correspond to the positions of the subdomain borders. This phenomenon is likely an artifact resulting from the convolution operation at the boundaries, where zero padding (same) is applied to maintain the input size. As discussed previously in Chapter 6, this phenomenon leads to boundary artifacts. For the baseline model, this effect is absent.
2. **Effect of the communication module:** Examining Figure 7.12, which depicts the feature maps after two upsampling blocks, the influence of the communication module becomes apparent. Comparing the feature maps of the models with and without the communication module, we observe that many of the boundary artifacts disappear for the model with communication. Furthermore, the feature maps of the model with communication show greater consistency between subdomains.

To delve deeper into the impact of the communication network, we show 8 input and 8 output feature maps for the communication network in Figure 7.13. The subdomain boundaries are visible in the input feature maps. Conversely, in the output feature maps, those boundaries appear smoothed by the convolutional operations within the communication network.

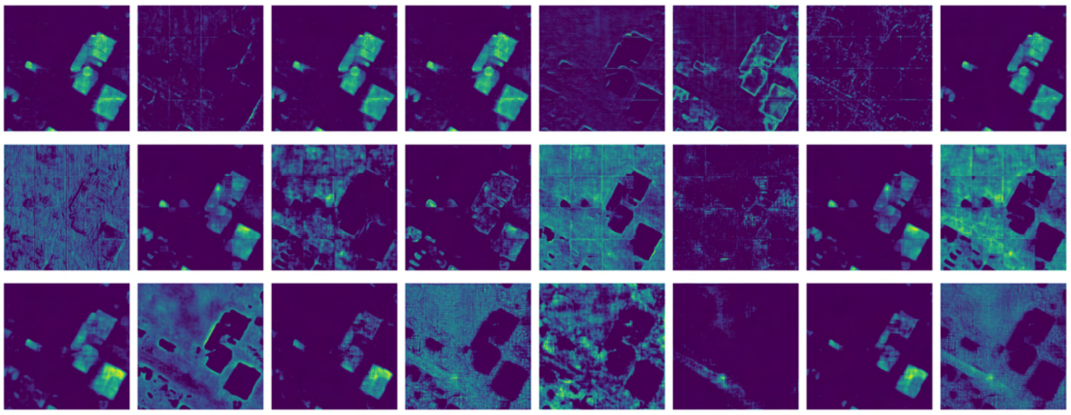


Figure 7.12: Feature maps after the second upsampling block for the proposed model with communication module (top row), without communication module (center row), and the baseline U-Net (bottom row).

Moreover, after comparing the feature maps with the input image and corresponding mask (Figure 7.10), we observe that several feature maps in the output give a (coarse-grained) initial estimation of the building and background locations in the image. This observation suggests that the output generated by the communication network contains not only high-level, semantic information but also spatially consistent predictions that are a first estimation for the final prediction.

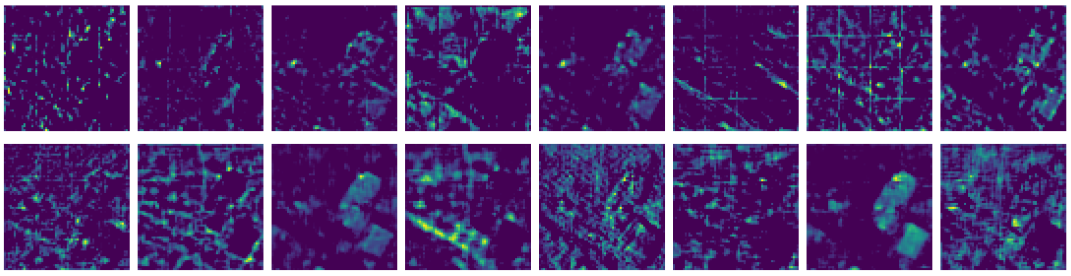


Figure 7.13: 8 input feature maps sent to the communication network (top row) and 8 feature maps generated by the communication network (bottom row).

7.4. Experiments on the DeepGlobe Satellite Segmentation

In contrast to the relatively local task of classifying each pixel as a building or background in the Inria Aerial Image Dataset, the segmentation of land types in the DeepGlobe Satellite Segmentation Dataset [14] demands a broader understanding of the global context for accurate predictions, as the segmentation task now is to segment land types instead of buildings. This section first discusses the adjustments made to the model to address the challenges posed by this high-resolution dataset, outlines the training procedure, and presents the obtained results.

7.4.1. Network parameters

The size of the DeepGlobe dataset is small compared to the complexity of the segmentation task, which leads to a considerable risk of overfitting the data. To mitigate this risk, several adjustments can be made to the network, including batch normalization, and random dropout layers. Also, data augmentation can play a role in avoiding overfitting. However, it turned out that, even with the application of these methods, the models were prone to heavy overfitting.

As an alternative approach, a pre-trained image encoder model was utilized as an encoder within the subnetwork. This strategy was implemented to leverage the pre-existing knowledge embedded in the pre-trained model and enhance the network's ability to generalize patterns from the limited dataset.

We use the pretrained ResNet-18 Network architecture [25] to initialize the encoder of our model. This model is trained for image classification in over 1000 classes on ImageNet [15], a very large and diverse

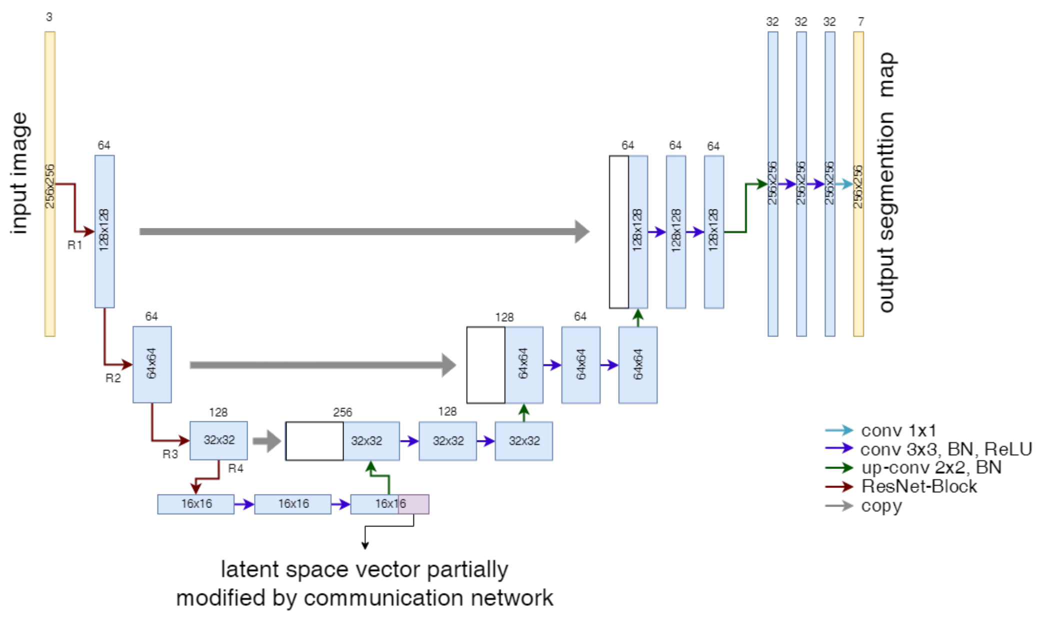


Figure 7.14: Architecture of the subnetwork ResNet-UNet for image segmentation. The pretrained blocks of the ResNet-18 are shown as $R1, R2, R3, R4$. Note that the architecture of these blocks is not shown completely, but is largely simplified in these images.

image dataset (3.2 million images in total), and has learned a very rich feature representation for a wide range of images. The ResNet-18 is an 18-layer deep convolutional network that consists of several residual *blocks*, introducing skip connections, which allow the gradients to flow more effectively without vanishing gradients. We use the first four building blocks of the model to initialize our encoder. The resulting architecture is shown in Figure 7.14.

It is important to note here that the ResNet-18 network performs a convolution with a stride of 2 directly on its input data. Consequently, this operation leads to feature maps with dimensions halved in width and height compared to the input image. As a result, utilizing a skip connection between feature maps of spatial size 256×256 pixels becomes infeasible due to the mismatch in dimensions. Therefore, on this fine level, no skip connection is used, but only a transposed convolution.

During the training process, we adopt a strategy of fixing the weights of the pre-trained ResNet-18 model. Consequently, only the weights of the decoder and the communication network (if included) undergo parameter updates. The decoder module is tasked with leveraging the features extracted by the pre-trained encoder to generate a segmentation mask.

As previously mentioned, ResNet models are not trained for segmentation, but image classification, which necessitates less spatial context in the deeper features. To address this difference in the application, we introduce two additional 3×3 convolutional layers in the bottleneck layer of the U-Net architecture. This addition aims to enhance the network's capacity to capture spatial dependencies crucial for segmentation tasks. The efficacy of these supplementary convolutional layers in the bottleneck will be investigated in the results section Section 7.4.3.

For a comprehensive overview of the parameter distribution within each component of the network, see Table 7.9.

7.4.2. Training hyperparameters

To assess the effectiveness of the extra convolutional operations in the bottleneck layer, we train models with and without these extra convolutions. We use the Adam optimizer [39], the Dice Loss (Equation (7.1)), and we set the dropout rate of the models to 0.1. Furthermore, we use the training hyperparameters shown in Table 7.10.

We encounter the same problem as for the Inria Image Dataset when loading the data 2448×2448 pixel

	Module	**Channel distribution**	**output feature map size**	**# of parameters**
Subnetwork	ResNet18-block1	3-64	128×128	9,536
	ResNet18-block2	64-64-64-64-64	64×64	147,968
	ResNet18-block3	64-128-128-128-128	32×32	525,568
	ResNet18-block4	128-256	16×16	2,099,712
	inter-conv (3×3)	256-256-256	16×16	1,180,672
	up-sampling path	256-128-64-32-7	256×256	845,415
	# of feature maps communicated	**Channel distribution**	**Kernel size**	**# of parameters**
Communication network	64	64-64-64-64	5×5	307,776

Table 7.9: Properties of the used subnetwork and communication networks for the DeepGlobe Satellite Segmentation Dataset.

Hyperparameters	
(max.) learning rate	0.001
(max.) number of epochs	50
early stopping patience	-
learning rate decay patience	5
minimum stopping epoch	-
dropout rate	0.1
batch size	8

Table 7.10: Hyperparameters used for training the model on the DeepGlobe Satellite Dataset

images on the one GPU device (with 16 GB video RAM) that we use for training: these dimensions are too large to train on. To work around this problem, we use the same procedure as described in Section 7.3.2, with two differences:

1. Instead of 128×128 pixel subdomains, we use 256×256 pixel subdomains now, as we don't need to store the intermediate outputs for the encoder network during training. This means that the 512×512 image patches are partitioned into 2×2 subdomains instead of the 4×4 we used for the Inria Aerial Image Dataset.
2. The ResNet-18 architecture is pre-trained using normalized images. Therefore, we normalize the input patches using the mean and standard deviation of the ImageNet (with RGB mean $(0.485, 0.456, 0.406)$ and RGB standard deviation of $(0.229, 0.224, 0.225)$, both applied after first rescaling to the domain $[0, 1]$).

7.4.3. Results

In this subsection, we present the results obtained from training models on the DeepGlobe satellite dataset. The presentation of these results is divided into two parts: first, we start by showcasing and discussing some generated segmentation masks, and next, we provide quantitative results.

Qualitative comparison

We compare the results of the three models discussed in Section 7.1.1. Additionally, due to the use of a pre-trained encoder network, we compare these models with and without extra convolutional layers in the bottleneck layer of the subdomains, see Table 7.9. So, in total, six different models are compared. We start by showing three of the predictions generated by the models with and without extra convolutions in the bottleneck layer, as depicted in Figure 7.15.

First of all, the amount of incorrectly predicted small, noisy regions is higher than the building dataset. This increased noise level can be explained by the inherent complexity of the segmentation task associated with this dataset. Unlike the building segmentation task for the Inria Aerial Labeling, this dataset requires a more global, contextual understanding. For instance, the presence of a tree in a region does not correspond to one specific land type but can correspond to multiple types, such as urban land, rangeland, or forest land. Consequently, the network must assimilate more extensive contextual information to predict the appropriate class accurately.

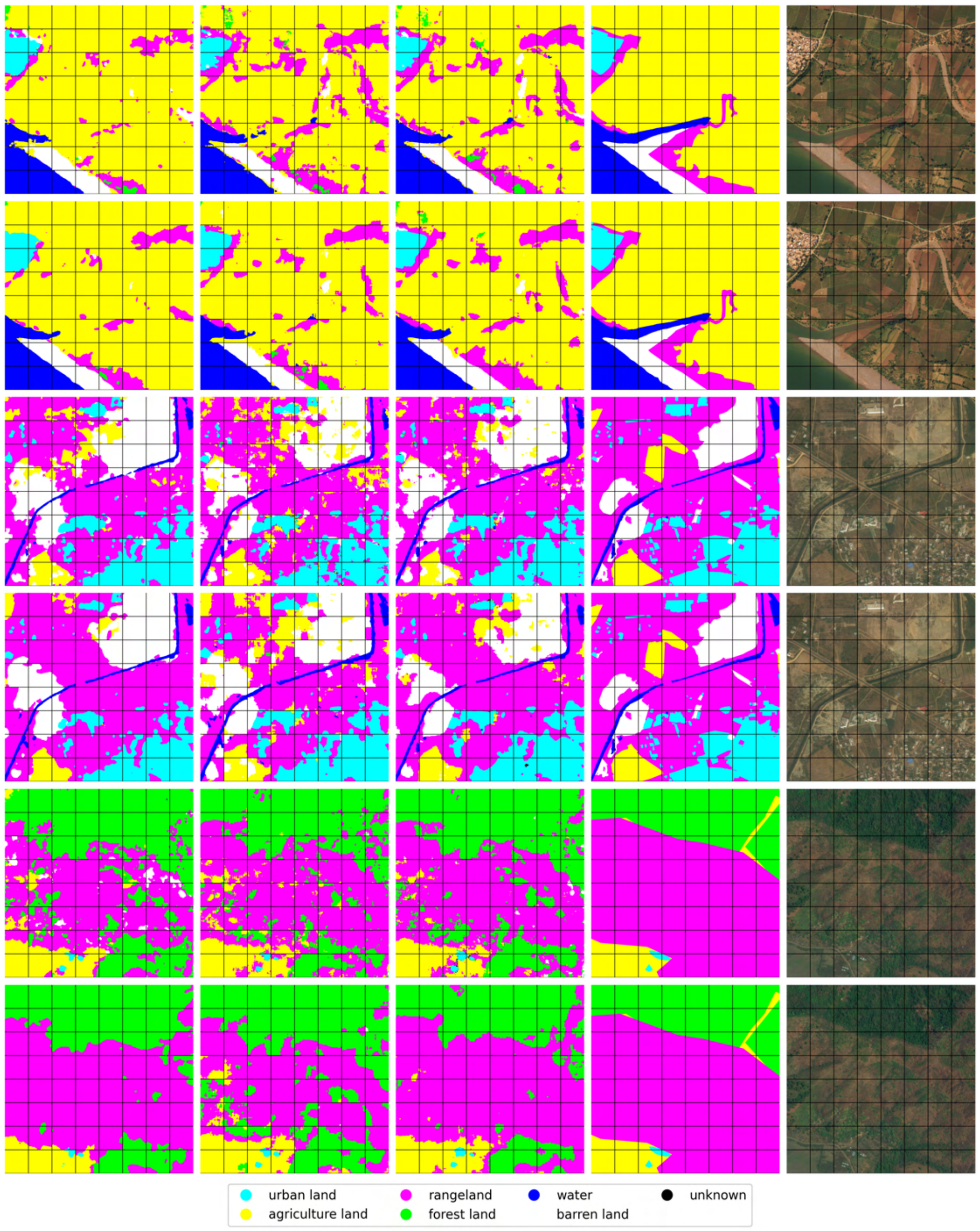


Figure 7.15: Some segmentation results for the DeepGlobe Aerial Image Dataset. The results are shown in blocks of two. The top row shows the predictions generated **without extra convolutions** in the bottleneck of the subnetwork, and the bottom row shows the predictions **with extra convolutions** in the bottleneck. From left to right: proposed model w/ communication, proposed model w/o communication, baseline U-Net, true mask, and image.

In all the example predictions showcased, it is evident that models incorporating extra convolutions in the bottleneck layer outperform the models lacking these extra convolutions. Notably, the additional convolutions contribute to the reduction of small, noisy segmented regions within the images. This improvement can be attributed to two key factors: Firstly, extra convolutions render the models more flexible and give them a larger prediction capacity, thereby enhancing their predictive capabilities. As depicted in Table 7.9, the convolutional bottleneck layers introduce a substantial number of trainable parameters, enabling the models to better adapt to the specific properties of the segmentation task. Secondly, these additional convolutions operate on coarse, deep feature maps, significantly expanding the network's effective receptive field. Consequently, the models are empowered to utilize features from a broader region to make local predictions, enhancing their overall performance.

Furthermore, an interesting observation from the segmentation results is that the model with a communication network exhibits more consistent predictions around subdomain borders, even at a fine-grained level. This phenomenon underscores the efficacy of the communication network in facilitating the transfer of contextual information across borders. Notably, this accomplishment is particularly remarkable, considering that the communication occurs at a coarse level.

It is clear from the predictions that all models, including the baseline and proposed architecture with and without communication, encounter challenges in accurately predicting certain land types. This difficulty is particularly evident in the examples of the river in the center row of Figure 7.15. Although there is an improvement in predicting these features with the addition of bottleneck convolutions, parts of the river or landline remain undetected. Upon closer inspection of the images, it becomes apparent that these regions pose challenges for distinction within the image, as their colors closely resemble the surrounding areas. However, when considering a broader context, such as the contours of the river, these features become more discernible.

Quantitative comparison

Significant variations exist between different classes in terms of the difficulty in predicting them. This variation is evident in the Intersection over Union (IoU) scores presented in Table 7.12 (for the models without extra bottleneck convolutions) and Table 7.11 (for the models with extra bottleneck convolutions). Classes such as "rangeland" and "barren land" consistently exhibit lower IoU scores across all models, indicating the relative difficulty in accurately predicting these classes.

	class	**IoU**		
		proposed w/ comm.	*proposed w/o comm.*	*baseline U-Net*
🟦	urban land	0.7453	0.7269	**0.7543**
🟨	agriculture land	**0.8337**	0.8149	0.8255
🟪	rangeland	0.3245	0.3369	**0.3434**
🟩	forest land	**0.6703**	0.6538	0.6649
🟦	water	0.7274	0.7036	**0.7298**
	barren land	0.4835	0.4915	**0.4798**

Table 7.11: IoU score for each class (except for the class "unknown") for the test dataset and the models **without extra convolution** in the bottleneck layer.

	class	**IoU**		
		proposed w/ comm.	*proposed w/o comm.*	*baseline U-Net*
🟦	urban land	0.7594	0.7638	**0.7732**
🟨	agriculture land	**0.8505**	0.8406	0.8484
🟪	rangeland	0.3774	0.3371	**0.3825**
🟩	forest land	**0.7156**	0.6727	0.6982
🟦	water	0.7686	0.7412	**0.7729**
	barren land	0.5365	0.5291	**0.5587**

Table 7.12: IoU score for each class (except for the class "unknown") for the test dataset and the models **with extra convolution** in the bottleneck layer.

By averaging the IoU scores across classes, we can compute the mean IoU scores (Equation (2.3)) for the training, test, and validation datasets, as presented in Table 7.13. These results show us that the baseline model exhibits the highest performance on the test dataset, both with and without the presence of additional convolutions in the bottleneck. However, for the training and validation datasets, the proposed model demonstrates superior performance, possibly (partially) caused by the higher number of weights in this model.

Moreover, a notable mean IoU improvement of over 0.02 is observed in the IoU scores between the proposed model and both the model lacking communication and the proposed model with additional bottleneck convolutions. In contrast, the difference in performance between the baseline and proposed models is comparatively marginal. This discrepancy underscores the capability of the proposed model to achieve prediction accuracy akin to that of the baseline model while enabling more efficient memory utilization.

model	**mean IoU**					
bottleneck	w/ extra convolution			w/o extra convolution		
dataset	*train*	*test*	*val*	*train*	*test*	*val*
baseline U-Net	0.7239	**0.6723**	0.6858	0.6742	**0.6339**	0.6457
proposed w/ communication	**0.7229**	0.6680	**0.6907**	**0.6829**	0.6308	**0.6562**
proposed w/o communication	0.7025	0.6474	0.6631	0.6614	0.6213	0.6359

Table 7.13: Results for the DeepGlobe Satellite Segmentation Dataset

8 Conclusion

In conclusion, this thesis addresses the challenge of memory issues for segmenting ultra-high-resolution images. Classical approaches to segment ultra-high resolution images include resizing and extracting image patches. These approaches lead, however, often to the loss of essential, spatially contextual information and reducing computation overhead. Other newer approaches focus on parallelizing the segmentation models over several devices, leading to either significant communication overhead or many redundant computations. We propose a novel method that is a trade-off between including spatial context and reducing memory requirements by combining the successful U-Net architecture with strategies from domain decomposition.

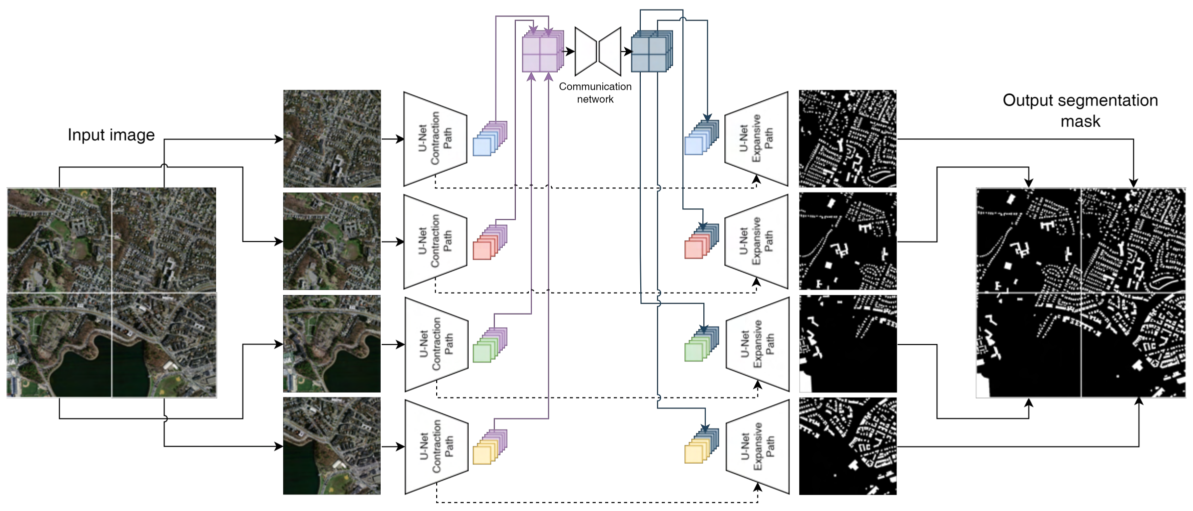


Figure 8.1: The proposed network architecture. The input image is partitioned into non-overlapping patches. Each computational device processes one or more patches. The contraction paths for each patch are processed independently. Before the expansive path operations occur, a selection of the feature maps of the bottleneck layer is communicated to one device and processed by a communication network. The modified feature maps replace the original feature maps. The expansive paths are again completely independent of each other. Dashed arrows denote the skip connections between the expansive and contraction paths in the U-Net architecture.

In Chapter 6, it is shown experimentally that the encoder part of our proposed approach, depicted in Figure 8.1, can transfer fine-grained spatial information. Furthermore, this chapter shows that the proposed model does introduce only a slight memory overhead for the communication module compared to a baseline U-Net model. The segmentation efficacy of our approach is evaluated on a synthetic dataset and realistic image datasets, including the Inria Aerial Image Dataset and the DeepGlobe Satellite Segmentation Dataset, and compared to a baseline U-Net model. Additionally, an ablation study is conducted to assess the impact of the communication module of our proposed model for inter-subdomain communication.

Our findings show, first of all, the effectiveness of the communication network in enlarging the (effective) receptive field with minimal additional parameters. Moreover, on synthetic data, our model can transfer positional information between subdomains that are close together and further apart, with segmentation performance comparable to a baseline U-Net model. Additionally, the scalability of the proposed approach is demonstrated by showing that training on a limited number of subdomains is not a limitation for the performance across higher numbers of subdomains.

Evaluation of realistic datasets shows competitive or even superior performance compared to the baseline U-Net model. Notably, our model facilitates consistent transitions of the class predictions around the boundaries, even with communication taking place on the coarse bottleneck level of the U-Net. The communication network plays an essential role in transferring this information, which is also supported by the visualization of feature maps. The results on the DeepGlobe dataset show that the proposed approach can also function when using a pre-trained encoder, particularly when additional convolutions are added to the subnetwork bottleneck layers.

In conclusion, our proposed model has proven memory-efficient and accurate for segmenting ultra-high-resolution images while effectively incorporating spatial context. Future research could explore other forms of the proposed model, such as overlapping subdomains to enhance boundary consistency, communication on different levels of the U-Net for more effective communication of fine-grained information, or communication networks with even larger receptive fields to stimulate the use of global information. Moreover, further research is necessary regarding the scalability of our model to gain more insight into how performance, in terms of memory usage, runtime, and segmentation accuracy, evolves with an increase in the number of subdomains.

References

[1] Delft High Performance Computing Centre (DHPC). *DelftBlue Supercomputer (Phase 1)*. `https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1`. 2022.

[2] André Araujo, Wade Norris, and Jack Sim. “Computing receptive fields of convolutional neural networks”. In: *Distill* 4.11 (2019), e21.

[3] Saeid Asgari Taghanaki et al. “Deep semantic segmentation of natural and medical images: a review”. In: *Artificial Intelligence Review* 54 (2021), pp. 137–178.

[4] Reza Azad et al. “Medical image segmentation review: The success of u-net”. In: *arXiv preprint arXiv:2211.14830* (2022).

[5] Arian Bakhtiarnia, Qi Zhang, and Alexandros Iosifidis. “Efficient high-resolution deep learning: A survey”. In: *ACM Computing Surveys* (2022).

[6] Tal Ben-Nun and Torsten Hoefler. “Demystifying parallel and distributed deep learning: An in-depth concurrency analysis”. In: *ACM Computing Surveys (CSUR)* 52.4 (2019), pp. 1–43.

[7] Max Bramer and Max Bramer. “An introduction to neural networks”. In: *Principles of Data Mining* (2020), pp. 427–466.

[8] X Cai. “Overlapping domain decomposition methods”. In: *Advanced Topics in Computational Partial Differential Equations: Numerical Methods and Diffpack Programming*. Springer, 2003, pp. 57–95.

[9] Wuyang Chen et al. “Collaborative global-local networks for memory-efficient segmentation of ultra-high resolution images”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 8924–8933.

[10] Yilong Chen et al. “Channel-Unet: a spatial channel-wise convolutional neural network for liver and tumors segmentation”. In: *Frontiers in genetics* 10 (2019), p. 1110.

[11] Özgün Çiçek et al. “3D U-Net: learning dense volumetric segmentation from sparse annotation”. In: *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2016: 19th International Conference, Athens, Greece, October 17-21, 2016, Proceedings, Part II 19*. Springer. 2016, pp. 424–432.

[12] Mirza Cilimkovic. “Neural networks and back propagation algorithm”. In: *Institute of Technology Blanchardstown, Blanchardstown Road North Dublin* 15.1 (2015).

[13] Taco Cohen and Max Welling. “Group equivariant convolutional networks”. In: *International conference on machine learning*. PMLR. 2016, pp. 2990–2999.

[14] Ilke Demir et al. “DeepGlobe 2018: A Challenge to Parse the Earth Through Satellite Images”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. June 2018.

[15] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[16] Foivos I Diakogiannis et al. “ResUNet-a: A deep learning framework for semantic segmentation of remotely sensed data”. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 162 (2020), pp. 94–114.

[17] Victorita Dolean, Pierre Jolivet, and Frédéric Nataf. *An introduction to domain decomposition methods: algorithms, theory, and parallel implementation*. SIAM, 2015.

[18] Yuping Duan, Huibin Chang, and Xue-Cheng Tai. “Convergent non-overlapping domain decomposition methods for variational image segmentation”. In: *Journal of Scientific Computing* 69 (2016), pp. 532–555.

[19] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. “Activation functions in deep learning: A comprehensive survey and benchmark”. In: *Neurocomputing* (2022).

[20] William Fedus, Barret Zoph, and Noam Shazeer. “Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity”. In: *Journal of Machine Learning Research* 23.120 (2022), pp. 1–39.

[21] Linyan Gu et al. “Decomposition and composition of deep convolutional neural networks and training acceleration via sub-network transfer learning”. In: (2022).

[22] Gousia Habib and Shaima Qureshi. “Optimization and acceleration of convolutional neural networks: A survey”. In: *Journal of King Saud University-Computer and Information Sciences* 34.7 (2022), pp. 4244–4268.

[23] Shijie Hao, Yuan Zhou, and Yanrong Guo. “A brief survey on semantic segmentation with deep learning”. In: *Neurocomputing* 406 (2020), pp. 302–321.

[24] Wang Hao et al. “The role of activation function in CNN”. In: *2020 2nd International Conference on Information Technology and Computer Application (ITCA)*. IEEE. 2020, pp. 429–432.

[25] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[26] Wei He et al. “Building extraction from remote sensing images via an uncertainty-aware network”. In: *arXiv preprint arXiv:2307.12309* (2023).

[27] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural networks* 2.5 (1989), pp. 359–366.

[28] Le Hou et al. “High resolution medical image analysis with spatial partitioning”. In: *arXiv preprint arXiv:1909.03108* (2019).

[29] Bohao Huang et al. “Large-Scale Semantic Classification: Outcome of the First Year of Inria Aerial Image Labeling Benchmark”. In: *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*. 2018, pp. 6947–6950. DOI: 10.1109/IGARSS.2018.8518525.

[30] Ziyan Huang et al. “AdwU-Net: adaptive depth and width U-Net for medical image segmentation by differentiable neural architecture search”. In: *International Conference on Medical Imaging with Deep Learning*. PMLR. 2022, pp. 576–589.

[31] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *International conference on machine learning*. pmlr. 2015, pp. 448–456.

[32] Md Amirul Islam, Sen Jia, and Neil DB Bruce. “How much position information do convolutional neural networks encode?” In: *arXiv preprint arXiv:2001.08248* (2020).

[33] Shruti Jadon. “A survey of loss functions for semantic segmentation”. In: *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*. 2020, pp. 1–7. DOI: 10.1109/CIBCB48159.2020.9277638.

[34] Ameya D Jagtap, Ehsan Kharazmi, and George Em Karniadakis. “Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems”. In: *Computer Methods in Applied Mechanics and Engineering* 365 (2020), p. 113028.

[35] Biswajit Jena et al. “Analysis of depth variation of U-Net architecture for brain tumor segmentation”. In: *Multimedia Tools and Applications* 82.7 (2023), pp. 10723–10743.

[36] Deyi Ji et al. “Ultra-High Resolution Segmentation with Ultra-Rich Context: A Novel Benchmark”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 23621–23630.

[37] Osman Semih Kayhan and Jan C van Gemert. “On translation invariance in cnns: Convolutional layers can exploit absolute spatial location”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 14274–14285.

[38] Asifullah Khan et al. “A survey of the recent architectures of deep convolutional neural networks”. In: *Artificial intelligence review* 53 (2020), pp. 5455–5516.

[39] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[40] Axel Klawonn, Martin Lanser, and Janine Weber. “A Domain Decomposition-Based CNN-DNN Architecture for Model Parallel Training Applied to Image Recognition Problems”. In: *arXiv preprint arXiv:2302.06564* (2023).

[41] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *nature* 521.7553 (2015), pp. 436–444.

[42] Baojun Li et al. “Real-time object detection and semantic segmentation for autonomous driving”. In: *MIPPR 2017: Automatic Target Recognition and Navigation.* Vol. 10608. SPIE. 2018, pp. 167–174.

[43] Pierre-Louis Lions et al. “On the Schwarz alternating method. I”. In: *First international symposium on domain decomposition methods for partial differential equations.* Vol. 1. Paris, France. 1988, p. 42.

[44] Wenjie Luo et al. “Understanding the effective receptive field in deep convolutional neural networks”. In: *Advances in neural information processing systems* 29 (2016).

[45] Jun Ma et al. “Loss odyssey in medical image segmentation”. In: *Medical Image Analysis* 71 (2021), p. 102035.

[46] Bohdan Macukow. “Neural Networks - State of Art, Brief history, Basic Models and Architecture”. In: *Computer Information Systems and Industrial Management: 15th IFIP TC8 International Conference, CISIM 2016, Vilnius, Lithuania, September 14-16, 2016, Proceedings 15.* Springer. 2016, pp. 3–14.

[47] Emmanuel Maggiori et al. “Can semantic labeling methods generalize to any city? the inria aerial image labeling benchmark”. In: *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS).* IEEE. 2017, pp. 3226–3229.

[48] N Man et al. “Multi-layer segmentation of retina OCT images via advanced U-net architecture”. In: *Neurocomputing* 515 (2023), pp. 185–200.

[49] Kyle Mills et al. “Extensive deep neural networks for transferring small scale learning to large scale systems”. In: *Chemical science* 10.15 (2019), pp. 4129–4140.

[50] Ozan Oktay et al. “Attention U-Net: Learning where to look for the pancreas”. In: *arXiv preprint arXiv:1804.03999* (2018).

[51] Jinhee Park et al. “Small object segmentation with fully convolutional network based on overlapping domain decomposition”. In: *Machine Vision and Applications* 30 (2019), pp. 707–716.

[52] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32.* Curran Associates, Inc., 2019, pp. 8024–8035. URL: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[53] S Vishnu Priyal et al. “Modified UNet Architecture with Less Number of Learnable Parameters for Nuclei Segmentation”. In: *Soft Computing and Signal Processing: Proceedings of 3rd ICSCSP 2020, Volume 2* 1340 (2021), p. 101.

[54] Alfio Quarteroni. “Introduction to Domain Decomposition Methods”. In: *Lecture-notes, 6th Summer School in Analysis and Applied Mathematics Rome* (2011), pp. 20–24.

[55] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional networks for biomedical image segmentation”. In: *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18.* Springer. 2015, pp. 234–241.

[56] Aheli Saha, Yu-Dong Zhang, and Suresh Chandra Satapathy. “Brain tumour segmentation with a muti-pathway ResNet based UNet”. In: *Journal of Grid Computing* 19 (2021), pp. 1–10.

[57] Sudip K Seal et al. “Toward large-scale image segmentation on summit”. In: *Proceedings of the 49th International Conference on Parallel Processing.* 2020, pp. 1–11.

[58] Qusay Sellat, Sukant Kishoro Bisoy, and Rojanlina Priyadarshini. “Semantic segmentation for self-driving cars using deep learning: a survey”. In: *Cognitive Big Data Intelligence with a Metaheuristic Approach*. Elsevier, 2022, pp. 211–238.

[59] Khemraj Shukla, Ameya D Jagtap, and George Em Karniadakis. “Parallel physics-informed neural networks via domain decomposition”. In: *Journal of Computational Physics* 447 (2021), p. 110683.

[60] Nahian Siddique et al. “U-Net and Its Variants for Medical Image Segmentation: A Review of Theory and Applications”. In: *IEEE Access* 9 (2021), pp. 82031–82057. DOI: `10.1109/ACCESS.2021.3086020`.

[61] Azzeddine Soulaimani et al. “An Object-Oriented Approach for Building PC Clusters”. In: *International journal on information* 6 (Jan. 2003), pp. 251–260.

[62] Kevin Swingler. *Applying neural networks: a practical guide*. Morgan Kaufmann, 1996.

[63] Xian Tao et al. “Automatic metallic surface defect detection and recognition with convolutional neural networks”. In: *Applied Sciences* 8.9 (2018), p. 1575.

[64] Andrea Toselli and Olof Widlund. *Domain Decomposition Methods-Algorithms and Theory*. Vol. 34. Springer Science & Business Media, 2004.

[65] Aristeidis Tsaris et al. “Distributed training for high resolution images: A domain and spatial decomposition approach”. In: *2021 IEEE/ACM Redefining Scalability for Diversely Heterogeneous Architectures Workshop (RSDHA)*. IEEE. 2021, pp. 27–33.

[66] Aristeidis Tsaris et al. “Scaling Resolution of Gigapixel Whole Slide Images Using Spatial Decomposition on Convolutional Neural Networks”. In: *Proceedings of the Platform for Advanced Scientific Computing Conference*. 2023, pp. 1–11.

[67] Ragav Venkatesan and Baoxin Li. *Convolutional neural networks in visual computing: a concise guide*. CRC Press, 2017.

[68] Joost Verbraeken et al. “A survey on distributed machine learning”. In: *Acm computing surveys (csur)* 53.2 (2020), pp. 1–33.

[69] Tyler Yep. *torchinfo*. Mar. 2020. URL: `https://github.com/TylerYep/torchinfo`.

[70] Renhe Zhang, Qian Zhang, and Guixu Zhang. “SDSC-UNet: Dual Skip Connection ViT-based U-shaped Model for Building Extraction”. In: *IEEE Geoscience and Remote Sensing Letters* (2023).

[71] Zongwei Zhou et al. “UNet++: A nested U-Net architecture for medical image segmentation”. In: *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support: 4th International Workshop, DLMIA 2018, and 8th International Workshop, ML-CDS 2018, Held in Conjunction with MICCAI 2018, Granada, Spain, September 20, 2018, Proceedings 4*. Springer. 2018, pp. 3–11.