Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science

Developing a Domain Decomposition-inspired Convolutional Neural Network Architecture for Image Segmentation

A thesis submitted to the
Delft Institute of Applied Mathematics
in partial fulfillment of the requirements

for the degree of

Master of Science
in
Applied Mathematics

by

Corné Verburg

Delft, Nederland
November 30, 2023

MSc thesis Applied Mathematics

Developing a Domain Decomposition-inspired Convolutional Neural Network Architecture for Image Segmentation

Corné Verburg

Delft University of Technology

Daily supervisor

dr. A. Heinlein	(Numerical Analysis)

Thesis committee

prof. dr. ir. C. Vuik	(Numerical Analysis)
dr. D. J. P. Lahaye	(Mathematical Physics)
dr. E. C. Cyr	(Sandia National Laboratories, USA)

November 30, 2023 Delft

Abstract

Contents

1 Introduction

In clinical settings, the interpretation of medical images, such as CT scans, echocardiograms, and MRI scans, has traditionally been carried out by medical experts, including radiologists and physicians. However, recent advancements in technology, particularly deep learning techniques using Convolutional Neural Networks (CNNs), have strongly influenced the field of medical image analysis. The CNN has emerged as a particularly powerful tool for image processing tasks. A systematic literature review by Fourcade and Khonsari [10] emphasized that, while CNNs are not meant to replace medical professionals, they do hold the potential to optimize routine tasks and positively impact medical practice.

Nonetheless, the use of CNNs comes with significant challenges that demand attention to enhance their effectiveness. As medical images continue to increase in resolution, CNNs must scale in size and necessitate longer training times to achieve accurate results. Researchers have responded to this challenge by proposing various advanced data compression techniques and training acceleration methods in recent years, see for an overview for example [14]. Nevertheless, further research in this domain remains necessary to fully harness the potential of CNNs and improve their practical application, as underscored in several literature reviews on CNN acceleration and compression, [6, 12].

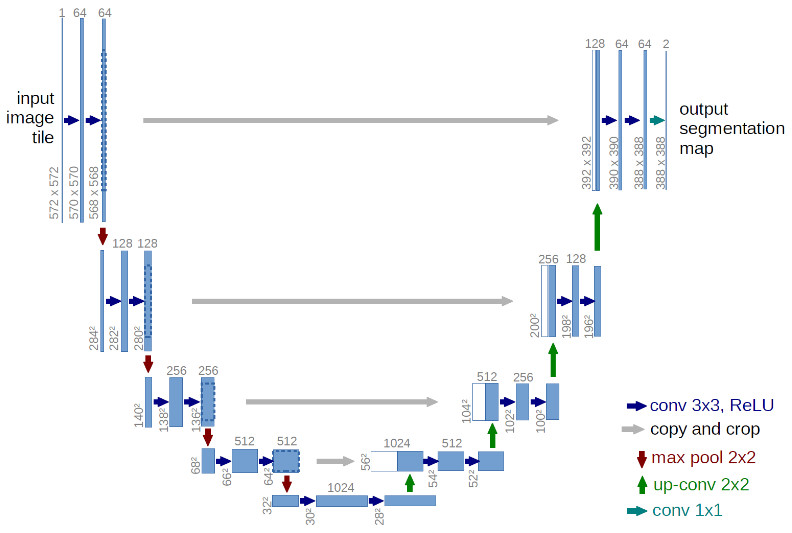


Figure 1: Schematic representation of the U-Net Architecture. The left part of the architecture is known as the *contraction path*, whereas the right part is denoted as the *expansive path*. Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. White boxes represent copied feature maps. Arrows denote the different operations. The gray arrows symbolize the skip connections in the U-Net, where a feature map from the contraction path is concatenated to feature maps in the expansive path. This image is obtained from the paper introducing the U-Net architecture [26]

In this literature study and research proposal, our primary focus is on a specific type of neural network introduced for medical image analysis, known as the U-Net. The U-Net architecture was first introduced by Ronneberger et al. in 2015 [26] and has since gained widespread popularity. It is designed specifically for automatic medical image segmentation and has proven successful across various medical image modalities due to its flexibility, optimized modular design, and overall effectiveness [1]. The architecture of the U-Net network is depicted in Figure 1. The U-Net's design resembles that of an auto-encoder network but distinguishes itself with the presence of *skip connections* connecting outputs in the encoding stage (the left part of the network, that *contracts* global features from the data) with corresponding inputs in the decoding stage (the right part, that *expanses* the found global features to the output dimensions). These connections enable the propagation of high-level contextual information through the network, allowing deeper layers to focus on more global image features.

A topic, seemingly unrelated to a convolutional neural network, is the field of *domain decomposition*. Traditionally, domain decomposition methods [29] are numerical methods employed to solve partial differential equations. These methods decompose a global problem on a large domain into smaller sub-problems on sub-domains, where the local sub-problems are addressed and solved for a large part independently. Domain decomposition methods provide ways to decompose the problem and *couple* the different sub-problems 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 major part of the work independently of the other processors.

This parallel nature inherent to many domain decomposition methods (DDMs) is relevant for other problems that demand extensive computational resources, such as training (large) neural networks, where parallelization of the problem can help to speed up computations. Consequently, combining insights, strategies, and approaches from domain decomposition with the design and training procedure of neural networks appears a promising direction. The literature study aims to provide an overview of existing research and papers exploring this combination. Additionally, we propose a novel network architecture inspired by DDM strategies for further investigation in this master thesis project.

This literature study has the following structure. Chapter 2 will provide theoretical background on the topics of convolutional neural networks and the U-Net. Chapter 3 gives a general introduction to domain decomposition methods. Existing research that describes (convolutional) neural networks incorporating domain decomposition strategies will be reviewed in Chapter 4. Finally, Chapter 5 outlines our research proposal, including a new DDM-inspired CNN architecture for image segmentation and discusses future research directions.

2 Convolutional Neural Networks

In this chapter, 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 an discuss training and optimization procedures. Furthermore, we discuss some classical CNN architectures.

2.1 Fundamentals of Neural Networks

Before we introduce CNNs, we first briefly introduce the more general concept of (feed-forward) neural networks. Neural networks are inspired by the human brain and initially started as an attempt trying to model the neurons in the human brain [21]. 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 $I = \{x_1, \ldots, x_N\}$ to corresponding output data $O = \{y_1, \ldots, y_N\}$, with $x_i \in \mathbb{R}^n$ and $y_i \in \mathbb{R}^m$, for $i = 1, \ldots, N$. In other words: we want the function F to satisfy the relation

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

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

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(F(x_i) - y_i)^2)} < \epsilon \tag{2}$$

where $\epsilon \in \mathbb{R}$ denotes the chosen upper bound for the error. Neural Networks are designed to perform this task of function approximation. They do so by taking advantage of some useful linear algebra properties. To understand this, we first consider the equation

$$P(x) = A\alpha(Bx + c). \tag{3}$$

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 ones. A model that only includes linear function 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 non-linearity 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 3 are universal function approximators. For example, for the sigmoid activation function, the following theorem can be proven [16]:

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*

$$P(x) = \sum_{i=1}^{M} a_i \alpha(Bx + c) \tag{4}$$

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

$$|P(x) - f(x)| < \epsilon, \quad \forall x \in I_n. \tag{5}$$

This theorem has been extended and proven for a more general class of activation functions in the paper with the name *Multilayer feedforward networks are universal approximators* [16]. 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. Then, the neural network is described by the set of equations

$$\begin{aligned} h_1 &= \alpha_1(W_1 x + b_1), \\ h_{i+1} &= \alpha_{i+1}(W_{i+1} h_i + b_{i+1}), \quad \text{for } i = 1, \dots, L-1, \\ y &= W_{L+1} h_L \end{aligned} \tag{6}$$

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 $i = 0, \dots, L-1$ 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 different functions for different layers.

If the weight matrices $W_i, i = 1, \dots, L+1$ 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 matrices. 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.

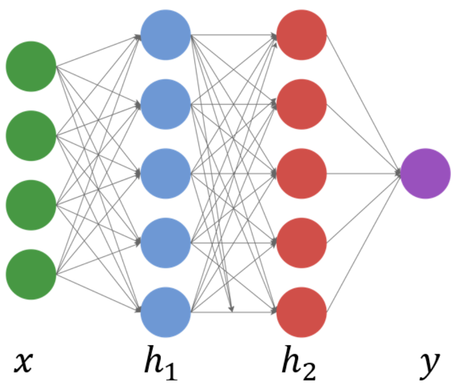


Figure 2: Visualization of a very simplistic neural network. The green circles denote the input vector x. The network consists of two hidden layers (the red and blue circles) and gives as output a 1D number, denoted by the purple circle. Note that the arrows between the different neurons represent the weights stored in the weight matrices W_i: each arrow represents another weight $w \in W_i$.

2.1.1 Activation functions

In this subsection, we explore some important activation functions, based on the survey on activation functions of Dubey et al. [8]. 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 3 shows some of the discussed activation functions.

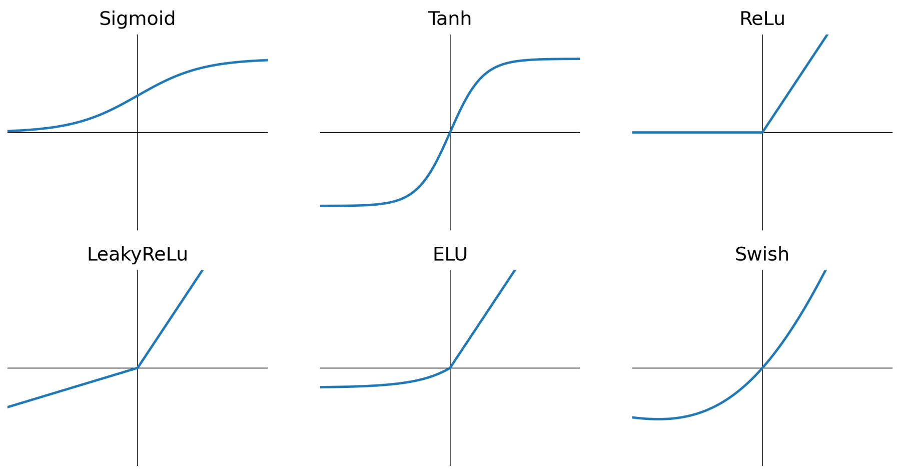


Figure 3: Some important activation functions. Note the similarities and differences between the ReLU, Leaky ReLU, and ELU activation functions: for $x > 0$ they are the same, but they are different for negative x. The first row shows the sigmoid, tanh, and yReLU respectively, whereas the second row shows the LeakyReLU, ELU, and Swish activation function respectively. The Softmax activation function is not shown since this functi

2.2 Training and Optimization

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 and optimization 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:

$$F_i(x) = \alpha(W_i x + b_i). \tag{7}$$

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

$$\mathcal{NN}^{\alpha}_{W,b} = W_{L+1} F_L \circ \ldots \circ F_1(x). \tag{8}$$

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:

$$\min_{W,b} \sum_{i=1}^{N} \mathcal{L}(\mathcal{NN}^{\alpha}_{W,b}(x_i), y_i). \tag{9}$$

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:

$$\mathcal{L}_{MSE}((\mathcal{NN}^{\alpha}_{W,b}(x_i), y_i) = \frac{1}{N} ||(\mathcal{NN}^{\alpha}_{W,b}(x_i) - y_i||_2^2 \tag{10}$$

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.

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

$$\sum_{i=1}^{N} \mathcal{L}(\mathcal{NN}^{\alpha}_{W,b}(x_i), y_i) \tag{11}$$

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

$$\nabla_{W,b} \mathcal{L}(\mathcal{NN}^{\alpha}_{W,b}(x_i), y_i), \tag{12}$$

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

$$W_{k+1} = W_k - \epsilon \nabla_W \mathcal{L}(\mathcal{NN}^{\alpha}_{W,b}(x_i), y_i), \tag{13}$$
$$b_{k+1} = b_k - \epsilon \nabla_b \mathcal{L}(\mathcal{NN}^{\alpha}_{W,b}(x_i), y_i), \tag{14}$$

where $\epsilon > 0$ 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 $\{0, \dots, N\}$ into K disjoint subsets with size k. Then, in the j-th step of the mini-batch SGD, we use the gradient

$$\sum_{i \in B_j} \nabla_{W,b} \mathcal{L}(\mathcal{N}\mathcal{N}^{\alpha}_{W,b}(x_i), y_i), \qquad j = 1, \dots, K \tag{15}$$

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.

Forward and backward propagation are the crucial steps in machine learning. 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. 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.

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. 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. Therefore, the batch-normalization approach is often used: 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.

2.3 Architecture of CNNs

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. 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*.

2.3.1 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

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v] \cdot F[i-u, j-v]. \tag{16}$$

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 respectively the input image or feature map and the filter/kernel. $2k+1$ is the size of the convolutional filter/kernel.

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 4.

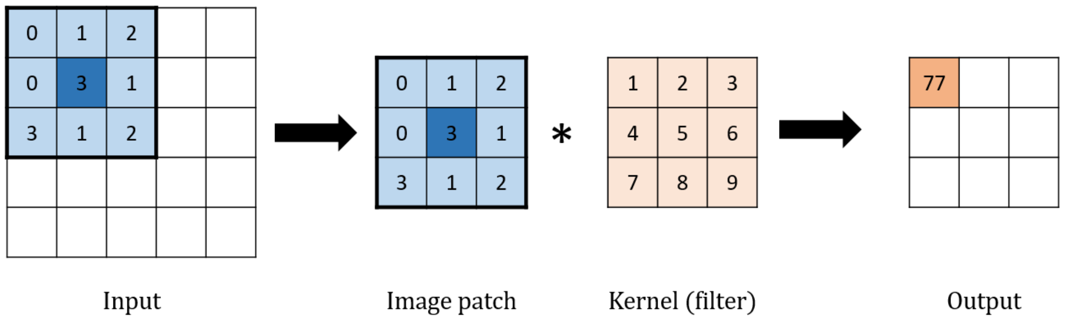


Figure 4: Visualization of the operations performed in the convolutional layer. The asterisk $*$ in this figure denotes a component-wise multiplication and summation operation. The values for the input and kernel are chosen arbitrarily - note that these are not fixed but will be updated during training. Note that the kernel weights are updated during the training phase of the neural network. Furthermore, note that there can be several kernels operating on the same input. Different resulting outputs are typically concatenated, resulting in an output with more feature maps. 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.

Note that this setup drastically reduces the number of learnable parameters in the network. For example, if we have a $64 \times 64 \times 3$ image as input and ignore the bias, one kernel of spatial size $3x3$ 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 $64 \times 64 \times 3 = 12,288$ 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 following parameters:

- The **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 dimension feature map.

- The **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 there are also studies that 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.

2.3.2 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 the cluster, and *average pooling*, which uses the average value of the cluster of neurons to generate the value of the new single neuron. This pooling operation is defined as

$$\text{MaxPooling}(x, p, q) = \max_{(i,j) \in R_{p,q}} x[i, j] \tag{17}$$

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

Pooling layers are helpful in extracting features that are invariant to translation shifts and small distortions [18]. They can also play a role in preventing over fitting and reducing the computational complexity of the network.

2.3.3 Upsampling layers

To reduce the spatial dimension of an image, pooling layers, striding and padding can be used. However, besides downsampling, for many use cases, it is necessary to increase the spatial dimensions of the feature maps. To do so, upsampling layers can be used. The simplest type of upsampling layers is the so-called *unpooling* layer, the reverse of pooling: generating a new feature map in which one neuron is replaced by a cluster of neurons. Often, this operation is followed by a convolutional layer. Another option is *transposed convolution*, in which the feature map is expanded by adding rows and columns of zeros in between the original feature map and performing a (learnable) convolution operation on this newly generated feature map.

2.4 The U-Net model

In this subsection, we discuss one of the most successful and famous [1] CNN architectures for medical image segmentation: the U-Net [26] architecture and its most important properties. The U-Net performs a pixel-wise segmentation task: for each pixel in the input image, it predicts a class label. The model uses two paths: a usual contracting path where features are extracted from the input image and stored in lower-dimension feature maps in several successive layers. Then, the low-dimensional feature maps are *expanded* again in the expansive path, consisting of up-sampling layers followed by convolutional layers and activation functions. Typical for the U-Net is furthermore that the number of feature maps in the up-sampling part is very large, allowing the network to propagate context information to higher resolution layers. In the expansive path, feature maps from the corresponding spatial resolution in the contraction path are concatenated to the expanded feature maps to include fine-grained information from the contraction path in the segmentation.

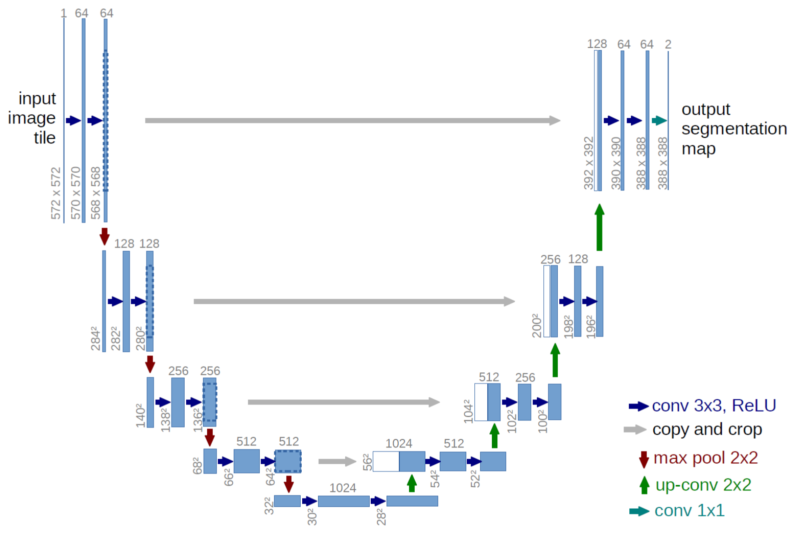


Figure 5: Schematic representation of the U-Net Architecture. Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. White boxes represent copied feature maps. Arrows denote the different operations. Image is obtained from the paper introducing the U-Net architecture [26]

The architecture of the U-net proposed by Ronneberger [26] is shown in Figure 5. Ronneberger concludes that the U-Net achieves very good performance on very 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 [28].

In [1], an overview is given of different models based on the original U-Net architecture. Challenges and opportunities are addressed for the U-Net family. As the paper states, "the primary purpose of almost all the approaches mentioned in this paper was to identify the limitation of the original U-Net model and design add-on modules to enhance feature reusability and enrich feature representation to bring more performance boost. However, including more parameters in the model usually results in large memory requirements, which makes the model unsuitable for clinical applications with limited computational devices". This shows the relevance of further research to this successful neural network, as strategically dividing the segmentation problem based on domain decomposition methods could pave the way for a more memory-efficient and practically deployable neural network architecture.

3 Domain Decomposition

This study tries to apply intuitions from the field of domain decomposition to the construction of CNNs. In order to do so, in this chapter the basic ideas of domain decomposition are introduced. The presentation of these methods is mainly based on the textbook of Toselli and Widlund [30]. This chapter is structured as follows. 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.

3.1 Basic Idea of DDMs and Model Problem

The basic idea of domain decomposition is natural and simple: it refers to the splitting of a (discretized) partial differential equation into coupled problems on smaller subdomains that form a partition of the original domain [30, p. V]. By splitting the domain into smaller subdomains, the problem size is reduced and as such the time necessary to come to a solution. Furthermore, the convenience of partitioning the domain into subdomains corresponding to various physical conditions arises from the fact that different sets of differential equations govern each subdomain.

Also, since the emergence of (parallel) computing power, domain decomposition methods have become a matter of interest to be combined with discretization methods for PDEs (FEM, FV, FD), to solve differential equations more efficient on parallel computer platforms. Now, we introduce the model problem that will be further used in this chapter. In

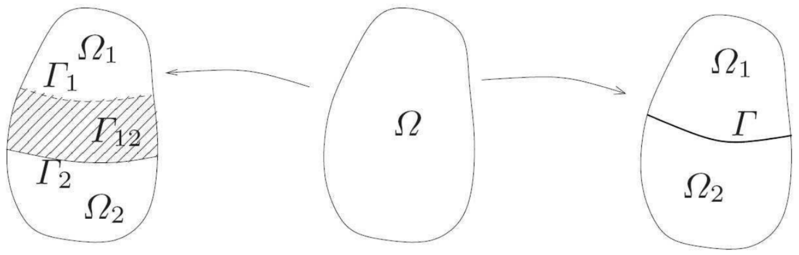


Figure 6: Overlapping subdomains (left) and non-overlapping subdomains (right) of the domain Ω (center). Figure reprinted from [25]

Figure 6, 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 domain decomposition methods (DDMs): (1) overlapping DDMs and (2) non-overlapping DDMs.

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

$$\begin{cases} Lu = f & \text{in } \Omega, \\ u = 0 & \text{on } \partial\Omega, \end{cases} \tag{18}$$

where $\partial\Omega$ denotes the boundary of the domain Ω, and L denotes a generic second-order elliptic differential operator, for example, the negative Laplacian $-\nabla^2$.

3.2 Overlapping Domain Decomposition Methods

The first domain decomposition method was introduced by Schwarz in 1870 [11]. Those methods are referred to as Schwarz (decomposition) methods. In this introduction to domain decomposition methods, we discuss some of the most fundamental methods, namely (1) the classical alternating Schwarz method, (2) the multiplicative Schwarz method (which is the multi-domain extension of the alternating Schwarz Method), and (3) the parallel Schwarz method introduced by Lions [20].

3.2.1 The Classical Alternating Schwarz Method

We will introduce the classical alternating Schwarz method by means of an example. The overlapping domain decomposition we will use in this example is shown in Figure 7. The two overlapping sub-regions are here denoted by Ω_1' and Ω_2' respectively, so that $\Omega = \Omega_1' \cup \Omega_2'$. Suppose that we want to solve the 2D Poisson problem (i.e. $L = -\nabla^2$ in our model problem, see Eq. 18) with zero Dirichlet boundary conditions on the domain given in Figure 7. Now, suppose we have an initial guess, denoted by u^0, which vanishes on $\partial\Omega$

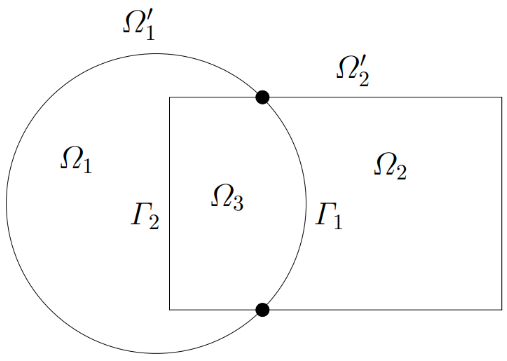


Figure 7: Domain Ω partitioned in two overlapping subdomains. Reprinted from the textbook of Toselli and Widlund [30].

(as the Dirichlet boundary conditions require). Then, we determine the iterate u^{n+1} from the previous iterate u^n in the following sequential steps presented in Equations 19 and 20:

$$(I) \quad \begin{cases} Lu^{n+1/2} = f & \text{in } \Omega_1' \\ u^{n+1/2} = u^n & \text{on } \partial\Omega_1' \\ u^{n+1/2} = u^n & \text{in } \Omega_2 = \Omega_2' \setminus \overline{\Omega_1'} \end{cases} \tag{19}$$

$$(II) \quad \begin{cases} Lu^{n+1} = f & \text{in } \Omega_2' \\ u^{n+1} = u^{n+1/2} & \text{on } \partial\Omega_2' \\ u^{n+1} = u^{n+1/2} & \text{in } \Omega_1 = \Omega_1' \setminus \overline{\Omega_2'} \end{cases} \tag{20}$$

These equations describe the *alternating Schwarz method.* In this method, first, we solve the Poisson equation restricted to the domain of the circle Ω_1'. As boundary conditions on the artificial boundary Γ_1, we use the result u^n from the previous iteration. Then, we repeat this process for the square domain Ω_2', but now we use the approximate solution computed in iteration (I) to construct a Dirichlet boundary condition for the artificial boundary Γ_2.

So, the two local Poisson equations in the overlapping subdomains Ω_1' and Ω_2' are coupled together as follows: the boundary conditions on the artificial boundaries Γ_1 and Γ_2 are constructed using the information provided by respectively Ω_2' and Ω_1'. Toselli and Widlund note that this algorithm therefore also can be viewed as a mapping of values of Γ_1 (or Γ_2) onto values of the same set [30, p. 22].

Finally, we note here that the example problem described here is not discretized yet. Later in this chapter, we will consider discretized versions of this algorithm.

3.2.2 The Multiplicative Schwarz Method

The Alternating Schwarz Method is a special case of the *Multiplicative Schwarz Method* for the case of two overlapping sub-domains. The Multiplicative Schwarz Method, on the other hand, can be applied to an arbitrary number of (overlapping) subdomains. Using the notations introduced in [4], assume we want to solve the following linear elliptic PDE:

$$\begin{aligned} Lu &= f \qquad \text{in } \Omega, \\ u &= g, \qquad \text{on } \partial\Omega \end{aligned} \tag{21}$$

with L a linear operator. Note that this is not the same as our model problem, since we have a Dirichlet boundary condition that can be non-equal to zero in this case. We decompose the domain Ω into P different subdomains. The set of subdomains is denoted by $\{\Omega_i\}_{i=1}^P$ and we construct the subdomains such that they satisfy the relation $\bigcup_{i=1}^P \Omega_i = \Omega$. Equivalent to the previous example, we denote by Γ_i 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 $\partial\Omega$. Also, we denote by $\mathcal{N}_i$ the set of indices of neighboring subdomains of the i-th subdomain Ω_i. In mathematical symbols, this means that $j \in \mathcal{N}_i \iff \Omega_i \cap \Omega_j \neq \emptyset$: if j is in the set of 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 also require that there is an overlap of at least one point 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 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 $i = 1, \ldots, P$, *exactly in this order*:

$$\begin{aligned} L_i U_i^n &= f_i \qquad \text{in } \Omega_i, \\ u_i^n &= g \qquad \text{on } \partial\Omega_i \setminus \Gamma_i, \\ u_i^n &= \overline{g}^* \qquad \text{on } \Gamma_i. \end{aligned} \tag{22}$$

In these equations, L_i is the restriction of the linear operator L to subdomain Ω_i, 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. Also, different parts on Γ_i can use information from different neighboring subdomains.

As well as for the alternating Schwarz Mmthod, we note that this method is sequential: since it requires the previous iterate in the construction of the artificial boundary condition, solving the Poisson equation on different subdomains cannot be done simultaneously.

3.2.3 The Parallel Schwarz Method

Lions was the first researcher to realize the potential of the Schwarz method on parallel computers [20]. To make the Schwarz multiplative Method more suitable for parallel computing, he proposed one minor change to the classical approach. The difference between the parallel Schwarz Method of Lions and the multiplicative Schwarz method lies in the way of updating the boundary conditions Γ_i. Recall that for the multiplicative Schwarz Method, we used

$$u_i^n = \overline{g}^* \qquad \text{on } \Gamma_i \tag{23}$$

where the boundary condition $\overline{g}^*$ was obtained using the most recent solution in the neighboring cells. For the parallel Schwarz method, the following artificial boundary condition is used:

$$u_i^n = \overline{g}^{n-1} \qquad \text{on } \Gamma_i, \tag{24}$$

instead of Equation 23. 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. Only after each iteration, communication of the boundary values needs to be done in order to update the boundary conditions for the next iteration. However, its convergence is inferior to that of the multiplicative Schwarz Method [4].

3.2.4 Extending continuous algorithms to discrete domain algorithms

The Additive, Multiplicative and Parallel Schwarz Methods defined previously hold for continuous domains. In numerical analysis, we often have to deal with discretized problems that only use a finite set of points, intervals or elements. As we will show in this subsection, the discretization of the continuous methods to a discrete one is rather straightforward.

Recall that we considered the continuous equation $Lu = f$ on the domain Ω in the continuous case. Now, suppose we have discretized this domain Ω^n into a grid of n points. Furthermore, suppose we have found some way to discretize the system $Lu = f$ on this grid, for example by finite difference methods, finite volume methods, or finite element methods. Then, we have a discretized system that we can denote by:

$$Au = f \tag{25}$$

where $A \in \mathbb{R}^{n\times n}$, $u, f \in \mathbb{R}^n$. If we want to discretize the Domain Decomposition Methods we saw before, we need to determine how to subdivide this discretized problem into sub-problems on the subdomains. To show how we can do so, we give an example of the basic Alternating Schwarz Method, using the example given in [5]. We choose a discretization of n points of the given domain Ω that is partitioned into two overlapping subdomains Ω_1 and Ω_2, and we let $\{\hat{I}_1 \hat{I}_2\}$ denote the sets of indices of the nodes in the interiors of domain $\hat{\Omega}_1$ and $\hat{\Omega}_2$, respectively. Note that these sets of indices are overlapping. We denote the number of indices in set $\hat{I}_i$ by $\hat{n}_i$ for $i = 1, 2$. Since some indices overlap (because of our overlapping subdomains), we have that $\hat{n}_1 + \hat{n}_2 > n$.

For each region $\hat{\Omega}_i$, we define a rectangular extension matrix R_i^T which extends a vector of nodal values in $\hat{\Omega}_i$ by zeros at the positions that are not in the indices set $\hat{I}_i$. I.e.: given a sub-vector x_i of length $\hat{n}_i$ with nodal values at the interior nodes on $\hat{\Omega}_i$, we define:

$$(R_i^T x_i)_k = \begin{cases} (x_i)_k & \text{for } k \in \hat{I}_i \\ 0 & \text{else.} \end{cases} \tag{26}$$

The transpose R_i of the extension map R_i^T is a restriction matrix that takes a full vector u of length n on the whole (discretized) domain Ω and restricts it to a vector of size $\hat{n}_i$ by choosing the entries of u that have indices $\hat{I}_i$. Using these matrices, we can find the local subdomain matrices:

$$A_1 = R_1 A R_1^T, \quad A_2 = R_2 A R_2^T \tag{27}$$

Now, we can derive the discrete version of the Schwarz alternating method. This method starts with a suitable initial guess u^0 and generates the iterates as follows:

$$\begin{aligned} u^{k+1/2} &= u^k + R_1^T A_1^{-1} R_1 (f - A u^k) \\ u^{k+1} &= u^{k+1/2} + R_2^T A_2^{-1} R_2 (f - A u^{k+1/2}) \end{aligned} \tag{28}$$

Note that the structure of these equations is very similar to the structure of the continuous case equations.

3.3 Non-Overlapping Domain Decomposition Methods

In the previous subsection, we discussed overlapping domain decomposition methods, the first class of DDMs. The other class of DDMs is the class of non-overlapping or *sub-structuring* domain decomposition methods. These methods use partitions of the global domain that are completely disjoint.

To introduce this class of DDMs, we use a concrete example: the Poisson equation on a region Ω, with zero Dirichlet boundary conditions on the boundary of Ω that is denoted by $\partial\Omega$. Now, suppose that we can partition the region Ω into two non-overlapping subdomains, which we denote by Ω_i, with $i = 1, 2$. These subdomains satisfy the following properties:

$$\overline{\Omega} = \overline{\Omega_1 \cup \Omega_2}, \;\; \Omega_1 \cap \Omega_2 = \emptyset, \;\; \Gamma = \partial\Omega_1 \cup \partial\Omega_2.$$

We also assume that the boundaries of the subdomains Ω_1 and Ω_2 overlap part of the whole domain boundary, i.e.:

$$\text{measure}(\partial\Omega_1 \cap \partial\Omega) > 0, \;\; \text{measure}(\partial\Omega_2 \cap \partial\Omega) > 0$$

with $\partial\Omega_1 \cup \partial\Omega_2 = \partial\Omega$and that the boundaries of the subdomains of Ω are Lipschitz continuous (meaning that the boundaries are *sufficiently regular*). A visualization of the partitioned domain Ω can be found in Figure 8.

Then, we consider the model problem Equation 18 on the region Ω. Under suitable regularity assumptions on f and on the boundaries, which typically are that f is square-summable and that the boundaries are Lipschitz continuous, problem **??** can be rewritten to the following coupled problem:

$$\begin{aligned} -\nabla^2 u_1 &= f && \text{in } \Omega_1, \\ u_1 &= 0 && \text{on } \Omega_1 \setminus \Gamma, \\ u_1 &= u_2 && \text{on } \Gamma, \\ \frac{\partial u_1}{\partial \mathbf{n}_1} &= -\frac{\partial u_2}{\partial \mathbf{n}_2} && \text{on } \Gamma, \\ -\nabla^2 u_2 &= f && \text{in } \Omega_2, \\ u_2 &= 0 && \text{on } \Omega_2 \setminus \Gamma. \end{aligned} \tag{29}$$

In this system, u_1 and u_2 are the restrictions of u to Ω_1 and Ω_2 respectively. The $\mathbf{n}_1$ and $\mathbf{n}_2$ denote the outward normal vectors to the boundaries $\partial\Omega_1$ and $\partial\Omega_2$.

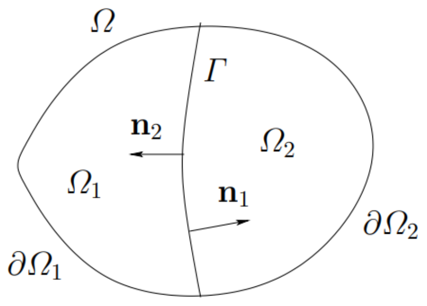


Figure 8: Domain Ω partitioned in two non-overlapping subdomains. Reprinted from the textbook of Toselli and Widlund [30].

This coupled problem shows that the problem on the global domain Ω is equivalent to two problems on both sub-domains, with Dirichlet and Neumann boundaries on the interconnecting boundary. Note that the boundary condition depends on both restrictions of the solution, u_1 and u_2. Therefore, the problem cannot be solved independently for both subdomains. The question that remains is: how can we use this equivalence result to define an iterative algorithm that approximates the solution u? In the following subsections, we will describe two of the most traditional non-overlapping algorithms that provide an answer to this question, namely (1) the Dirichlet-Neumann and (2) the Neumann-Neumann Algorithm

3.3.1 The Dirichlet-Neumann Algorithm

The basic Dirichlet-Neumann algorithm consists of two steps that both correspond to subdomain Ω_i, with $i = 1, 2$. Suppose we have an initial guess for the solution on the boundary Γ, denoted by u_Γ^0. In the first subdomain, Ω_1 we solve a Dirichlet problem with the Dirichlet data u_Γ^0 on the boundary Γ. Then, on the second subdomain Ω_2, we solve a Neumann-problem with a Neumann condition determined by the approximation on Ω_1 obtained in the previous step. Then, the new iterate at the boundary is chosen as a linear combination of the trace of the solution on Ω_2 and u_Γ^0, with weights of respectively *theta* and $(1-\theta)$. In equations, this can be written as follows:

$$(D) \quad \begin{cases} -\nabla^2 u_1^{n+1/2} & = f \qquad \text{in } \Omega_1, \\ u_1^{n+1/2} & = 0 \qquad \text{on } \partial\Omega_1 \setminus \Gamma, \\ u_1^{n+1/2} & = u_\Gamma^n \qquad \text{on } \Gamma, \end{cases} \tag{30}$$

$$(N) \quad \begin{cases} -\nabla^2 u_2^{n+1} & = f \qquad \text{in } \Omega_2, \\ u_2^{n+1} & = 0 \qquad \text{on } \partial\Omega_1 \setminus \Gamma, \\ \frac{\partial u_2^{n+1}}{\partial n_2} & = -\frac{\partial u_1^{n+1/2}}{\partial n_1} \qquad \text{on } \Gamma, \end{cases} \tag{31}$$

$$u_\Gamma^{n+1} = \theta u_2^{n+1} + (1-\theta) u_\Gamma^n \quad \text{on } \Gamma. \tag{32}$$

Here, $\theta \in \mathbb{R}$ is the parameter that determines the relaxation, with $\theta \in (0, \theta_{\max})$, with $\theta_{\max} \in \mathbb{R}$ dependent on the problem. Note that this method uses as well the Dirichlet condition as the Neumann condition described in the system (29). Although this algorithm is consistent (i.e. it approaches an exact solution to the equations when the step size goes to zero), its convergence is not always guaranteed [25].

3.3.2 The Neumann-Neumann Algorithm

The Neumann-Neumann Algorithm also starts from an initial guess of the boundary solution u_Γ^0. First, Dirichlet problems on *each* of the subdomains Ω_i, with boundary data u_Γ^0 are defined. Then, on each subdomain, a Neumann problem is solved, with Neumann data on Γ that is chosen to be the difference of the normal derivatives of the solutions of the two Dirichlet problems. Using the computed value, the initial guess is corrected to find the new iterate u_Γ^1. For $i = 1, 2$, we can write this in the following equations

$$(D_i) \quad \begin{cases} -\nabla^2 u_i^{n+1/2} & = f \qquad \text{in } \Omega_i, \\ u_i^{n+1/2} & = 0 \qquad \text{on } \partial\Omega_i \setminus \Gamma, \\ u_i^{n+1/2} & = u_\Gamma^n \qquad \text{on } \Gamma, \end{cases} \tag{33}$$

$$(N_i) \quad \begin{cases} -\nabla^2 \phi_i^{n+1} & = 0 \qquad \text{in } \Omega_i, \\ \phi_i^{n+1} & = 0 \qquad \text{on } \partial\Omega_i \setminus \Gamma, \\ \frac{\partial \phi_i^{n+1}}{\partial n_i} & = \frac{\partial u_1^{n+1/2}}{\partial n_1} + \frac{\partial u_2^{n+1/2}}{\partial n_2} \qquad \text{on } \Gamma, \end{cases} \tag{34}$$

$$u_\Gamma^{n+1} = u_\Gamma^n - \theta(\phi_1^{n+1} + \phi_2^{n+1}) \quad \text{on } \Gamma. \tag{35}$$

In this equation, $\theta > 0$ is called the *acceleration parameter.*

3.4 Multi-level methods in general

Unfortunately, numerical results show that most domain decomposition methods that are based only on the use of local subdomains do not scale with the number of subdomains. This is since 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 [25]. Therefore, global information travels slowly through the system, which can lead to slow convergence. This can be fixed by using a two- or multi-level method. This means that we introduce a ”coarse” global problem over the whole domain to facilitate a mechanism of global communication, not only between neighboring subdomains but among all subdomains.

4 Domain Decomposition and Neural Networks

This chapter further explores the relationship between domain decomposition and neural networks from a literature perspective. In previous years, several papers have been published on how to combine neural network architecture and training with ideas from domain decomposition. We give a concise overview of the ideas presented in these papers and try to sketch how these are relevant to this project.

In the past years, much work has been done to include *domain knowledge* into machine learning algorithms, helping them to accelerate training and prediction, improving accuracy and explainability [2]. Domain knowledge can be interpreted as including physical information, domain constraints, expert knowledge, or other relevant information in the machine learning algorithm. Furthermore, ML models have been used to replace parts of numerical methods where the domain knowledge is not sufficient to choose a sufficiently working numerical model.

The most recent published literature study on combining Domain Decomposition and Machine Learning can be found in [15]. In their study, the authors provide a brief overview of the combination of machine learning and DDMs. They divide the approaches combining ML and DDM into three classes. Note that these classes are not necessarily disjoint.

1. The first class consists of approaches where ML techniques are used within a classical DDM to improve the convergence properties or the computational efficiency.
2. The second class consists of approaches where deep neural networks (DNNs) are used as discretization methods and solvers for differential equations, replacing classical finite difference or finite element approaches. These DNNs are used as subdomain solvers then in classical DDMs.
3. The third class consists of approaches where ideas from DDMs are used to improve ML algorithms. Especially, the parallelization properties and convergence speed of the training speed of a neural network are mentioned here.

In this study, we focus on neural networks that perform segmentation tasks and not on a classical DDM problem, involving a domain and a boundary value problem that needs to be solved on this domain. Therefore, we believe that the third category is the most relevant for this study. Whereas in [15], the literature study is focused on the first two classes, in this work we will focus on the third class. We refer to [15] for more background information on the other two classes.

In the remainder of this chapter, we will start by giving an overview of approaches that can be distinguished in this third class of approaches where DDMs are used as a source of inspiration source for ML algorithms and we discuss some relevant papers for this thesis project and identify challenges and opportunities.

4.1 Classifying DDM-inspired ML approaches

As discussed in Chapter 3, the power of domain decomposition methods is their ability to partition a global problem into smaller subproblems that can be processed *in parallel*, which helps us to gain both *computational* and *memory* efficiency. This parallel property is desirable for many machine-learning tasks too. Therefore, in this chapter, we give a short overview of the three most important classes of parallelism in machine learning. Then, we identify to which class of parallelism most DDM-inspired ML algorithms belong.

4.1.1 Parallelism in Machine Learning

Parallelism (or distributed learning) in neural networks refers to the approach that splits the computational workload into smaller parts and distributes those among several worker nodes. Three of the most prevalent partitioning strategies are **data parallelism**, **model parallelism** and **pipelining** [3]. In this section, we focus on parallelism for neural networks.

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 increments 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. This approach 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 interdependent communication and independent communications determines how much time can be gained by data parallelism.

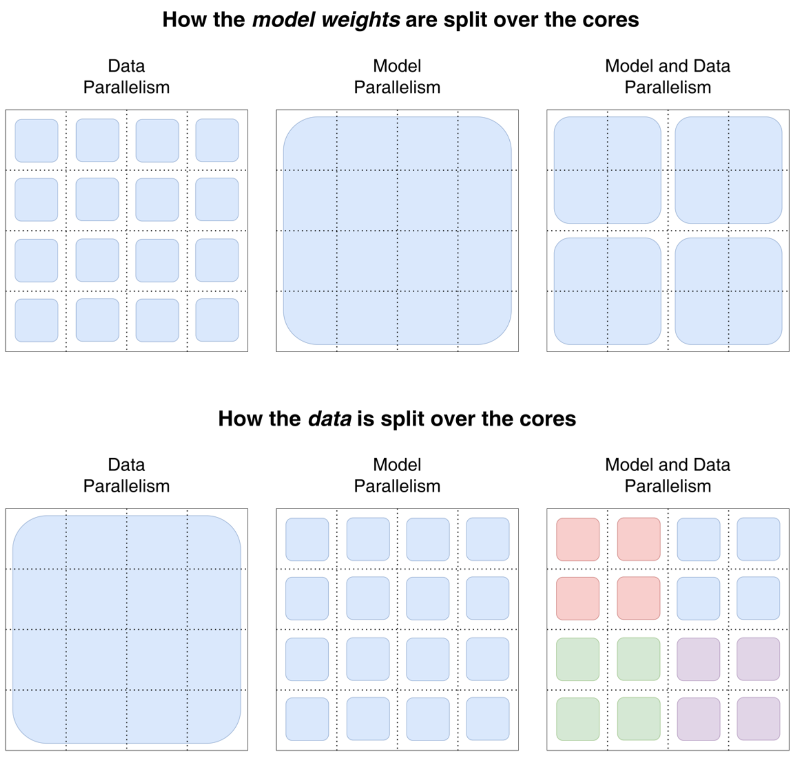

How the *model weights* are split over the cores
Data Parallelism
Model Parallelism
Model and Data Parallelism
How the *data* is split over the cores
Data Parallelism
Model Parallelism
Model and Data Parallelism

Figure 9: 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.

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 [3]. 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 9, the different forms of parallelism are visualized. We note that this visualization shows the similarity between DDMs 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.

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 [14, 31].

4.2 Research on DDM-inspired classification- and segmentation CNNs

Machine learning has been employed for solving PDE problems, also combined with approaches from Domain Decomposition. Physics-Informed Neural Networks (PINNs) have been used to replace a discretization and solver for the classical Schwarz method, for example in [17, 27]. However, in this literature study, we will focus on neural networks that are 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 the best of our knowledge, the number of papers published on this particular topic is very limited.

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 [22]. In this research, the purpose is 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 the fact that the retinal layers in the left half of the training samples behave differently than the retinal layers on the right side of the training images. Therefore, they decompose the training data images into two sub-images and they train two separate U-Nets on both sub-images. This approach allows both U-Nets to specialize in their part of the data and also reduces the overall complexity of the model. Furthermore, it allows for completely 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 [22, p. 198].

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

In [23], the authors propose a physically-motivated topology of a deep neural network 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

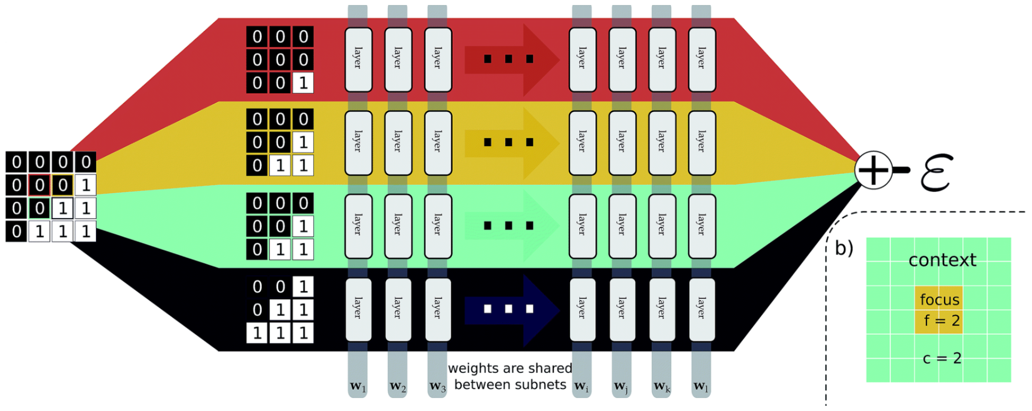


Figure 10: Schematic visualization of the EDNN. An input example is decomposed into four tiles, with each tile consisting of a focus and a context region (see bottom right. The focus region is denoted as a yellow part here, the context as green). For this case both the focus and the context are unit width, resulting in 3×3 tiles. The tiles are simultaneously passed through the same neural network (i.e. the same weights). The individual outputs are summed, producing an estimate of ϵ, an extensive quantity. When training, the cost function is assessed after this summation, forcing the weight updates to consider all input tiles simultaneously. Different colors denote different subdomains. The figure is reprinted from [23].

proposed in [23] is shown in Figure 10. 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. [19] present a CNN-DNN architecture that naturally supports a model parallel training strategy. Their method is ”loosely inspired by two-level domain decomposition methods” [19, p. 1]. The network they propose is shown in Figure 11. In their approach, first, the image is split into multiple sub-images. For each of the (either

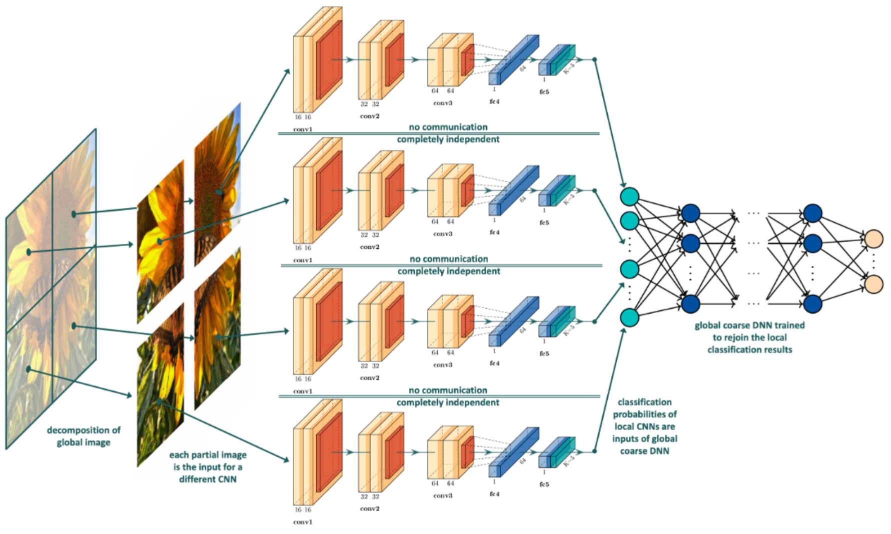

decomposition of global image
each partial image is the input for a different CNN
no communication completely independent
no communication completely independent
no communication completely independent
conv1
conv2
conv3
fc4
fc5
classification probabilities of local CNNs are inputs of global coarse DNN
global coarse DNN trained to rejoin the local classification results

Figure 11: Two-level decomposition of a CNN used for an exemplary image recognition problem. Left: The original image is decomposed into $p \times p, p = 2$ non-over-lapping sub-images. Middle: The $p \times p$ sub-images are used as input data for p×p independent, local CNNs. Each of these local CNNs has a proportionally smaller size than a global CNN for the same problem and is exclusively trained on local parts of the original image. Right: The probability values of the local CNNs are used as input data for a DNN which we refer to as a global coarse net. The global coarse net is trained to make a final classification for the decomposed image by weighting the local probability distributions. Image reprinted from [19].

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, which is 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 and 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. [13] propose another method to parallelize the training of CNNs, visualized in Figure 12. 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 is based on the assumption 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

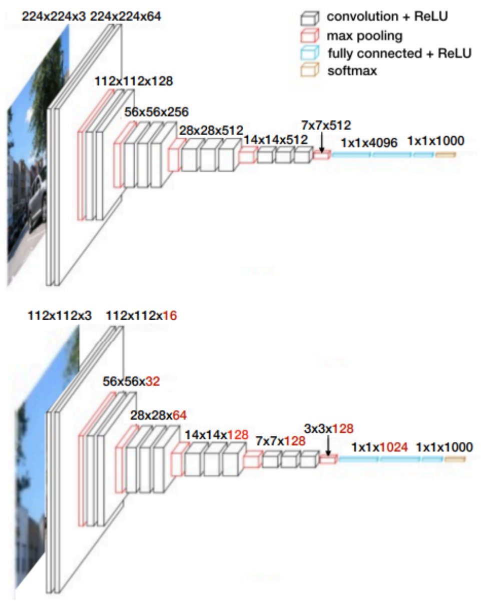


Figure 12: 1. Illustration of the decomposition of DCNN by the method proposed in [13] The global network (VGG16) is uniformly decomposed into 4 sub-networks by partitioning the “width” of the global network while keeping the depth constant. The input is also decomposed into 4 partitions. (top) The architecture of VGG16. (bottom) The architecture of one of the sub-networks. Image is reprinted from [13].

fundamentally different from the method in [19]: the method of Gu 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 Other papers

In [24], 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 segmentation accuracy and accurately segments small objects in real-time. Duan et al. [7] use a non-overlapping image partitioning to generate a Chan-Vese model for variational image segmentation, which can cause inaccurate segmentation results around boundaries of decomposed sub-domains. Firsov et al. [9] use an embarrassingly parallel domain decomposition method to denoise digital images. The authors do not incorporate communication between subdomains in this network since the noise is a local feature because the noise is local. Their model shows linear time complexity in the number of pixels.

4.3 Challenges and opportunities

In this subsection, we identify opportunities and open challenges for DDM-inspired CNNs in the field of image segmentation. As was discussed, many of the DDM-inspired approaches for machine learning show large potential for speeding up training and or evaluating large data sets with images of high quality. Especially, 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/or evaluation. This can be advantageous when working with large or complex data to avoid memory issues and to speed up computations.
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. **Reduced Computational Complexity**. For certain types of problems, such as the extensive parameter prediction problem in [23], the size-unaware property of CNNs (convolutional layers can operate on inputs of arbitrary size) is fully exploited, leading to a much smaller model with lower computational complexity.
4. **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 only gives all the necessary information for problems with a high locality. If a problem has more global features, this approach may not be sufficient, leading to lower accuracy.
2. **Model complexity**. The introduction of sub-networks can easily increase the overall complexity of the CNN architecture since more weights need to be stored.
3. **Communication overhead**. For many non-local applications of CNNs, 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 lead to improved accuracy or training speed, it is a challenge 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 ML approaches for image segmentation and classification.

5 Research Proposal

In this section, we propose new network architectures inspired by domain decomposition methods. First, we describe some choices that were made in this research to limit the amount of work. Then, we propose a new DDM-inspired network architecture and we discuss the components of this network in more detail. Furthermore, we present the results of some preliminary experiments performed with this network to show that it his able to facilitate global communication. Finally, we conclude by formulating a research purpose and additional questions that will be investigated in the remaining time for this project.

5.1 Research choices for this project

In order to limit the amount of work for this project, we made some explicit choices for which type of network we want to investigate. In the following list, we clarify and explain these choices.

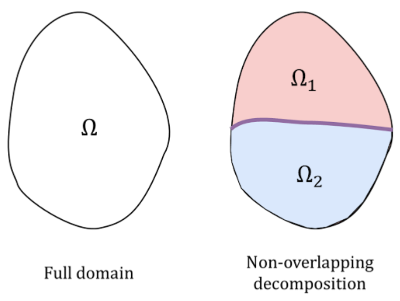
- *We only consider neural pixel-wise segmentation tasks.* This means that, for each pixel in the input image, a probability prediction should be made for each pixel telling us what the probability is that it belongs to class $1, \dots, n$ with n the number of classes.
- *The classical U-Net architecture is used as baseline mdoel.* Throughout the years, many variations of U-Net have been proposed, trying to reduce computational efforts or improve segmentation accuracy, for example by modifying the skip connections. For the sake of clarity, we chose not to consider those models, but use the classical U-Net architecture as a baseline model and as an important inspiration for the proposed network.
- *The focus of this study is on the algorithmic/methodic side of the implementation.* The design of a new, efficient architecture requires good programming, especially when implementing the architecture on a multiple GPU platform. In this thesis, we will focus on understanding the architecture and not too much on a high-standing implementation - however, opportunities and challenges for efficient implementation will qualitatively be discussed.

5.2 Proposed network architecture

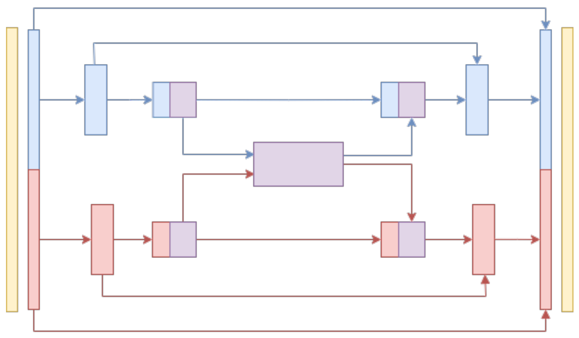
We propose a DDM-inspired network architecture that first partitions the data into non-overlapping sub-images, equivalent to partitioning the domain in non-overlapping DDMs. All sub-images are processed separately and in the end they are concatenated back together to obtain a segmentation mask with the same dimensions as the input image.

Before we discuss the proposed network further, note that Figure 13 shows a schematic representation of the proposed network (in Figure 13a) with a schematic representation of a non-overlapping domain decomposition (in Figure (13b each other.

Now, we discuss how the proposed model relates to domain decomposition methods. In the first place, by the decomposition of the image into two or more sub-images. In



(a) Non-overlapping domain decomposition



(b) Simplified visualization of the proposed network architecture.

Figure 13: Relation between domain decomposition methods and the proposed network. Note that, just as in a non-overlapping domain decomposition method, the input data to the network is decomposed into two non-overlapping parts that are processed independently. Whereas the communication between both non-overlapping domains takes place through the boundary, this is represented in the proposed architecture by a small *communication network* (purple) that combines latent space vectors from both subnetworks.

the second place, it was noted in Chapter 3 that communication between subdomains in a non-overlapping DDM only happens through the boundary. By enforcing certain boundary conditions, consistency between the solution on both subdomains is obtained, only using boundary data (which is only a small part of the whole domain). This idea was used to construct a network that only uses a small part of the information of the sub-images for data exchange between both sub-domains: the last m feature maps in the latent space are selected and sent to a *communication network* that combines the information from both sub-networks and returns *modified feature maps*. Those are sent back to the sub-networks and either replace part of the latent space vector or are concatenated to the latent space there (this is a design choice). Important is that this information is **not** necessarily boundary information.

Note that the visualization in Figure 13b is heavily simplified. The number of layers does not correspond to the number of layers that will be used. In the following two subsections, the subnetwork and communication network will be presented in more detail.

5.2.1 Sub-network architecture

In Figure 14, one of the subnetworks in the proposed architecture is shown in more detail for input sub-images of dimension $32 \times 32 \times 1$ (gray-scale images). This sub-network is similar to a classical U-Net architecture; the only difference is the modified latent space vector. Instead of directly passing the feature maps of the deepest layer to the expansive path, the communication network modifies the last k feature maps. Note that all subnetworks have the same architecture, so it is sufficient to show only one of them.

Concerning the architecture of the sub-network, different choices need to be made, dependent on the size and complexity of the data set. For example, the input dimension, the number of channels in each layer, and the number of layers are variables that can be adjusted. Furthermore, if the input sub-images are similar in what they depict, it can be worth considering using only *one* sub-network for both sub-images. This is something which can be different for different datasets.

Comparing the proposed network to domain decomposition methods, parallels can be

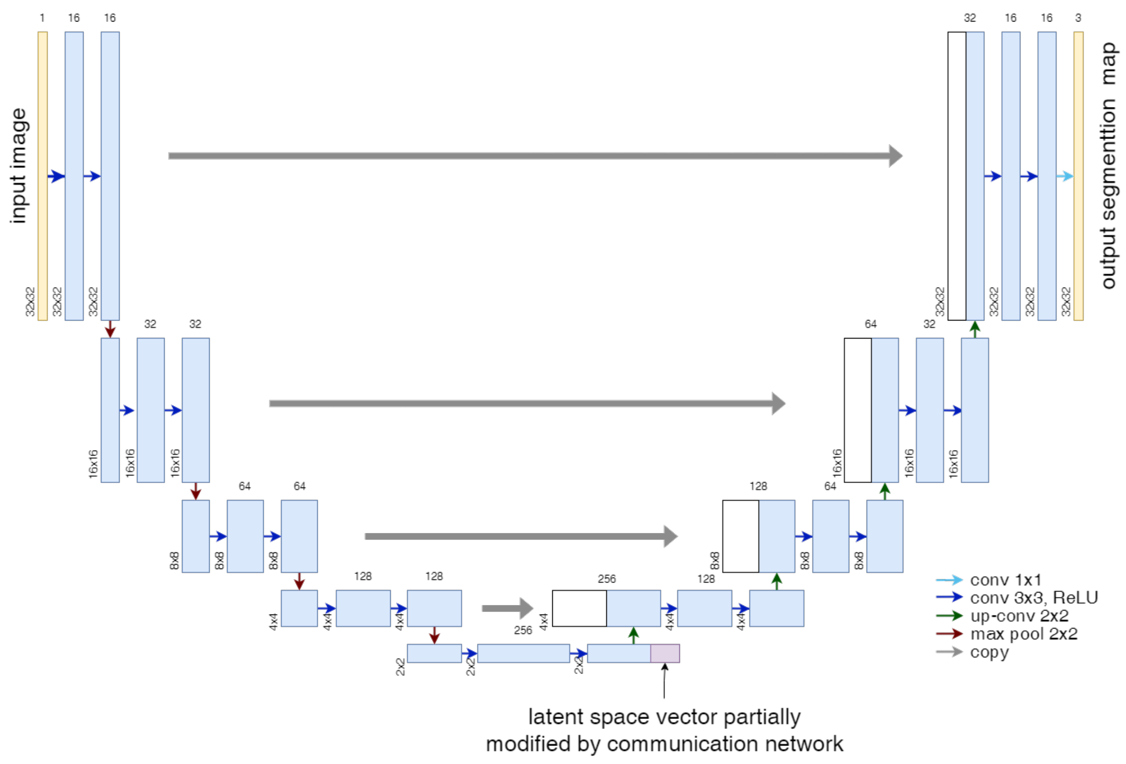


Figure 14: The proposed sub-network architecture. Note that the architecture of the sub-network is almost the same as the architecture of U-Net [26]. The only difference is located in the latent space vector, where a number of the feature maps are modified by the communication network. This is indicated in purple in this visualization. The input of this network consists of $32 \times 32 \times 1$ gray-scale images and it returns segmentation maps for a 3-class pixel-wise classification problem. Note that the input and output number of channels can be easily modified without changing the structure of the model. This subnetwork represents one of the two subnets shown in 13

seen between the *coarse problem* used in multi-grid methods and the bottleneck in the U-Net architecture. Recall that the coarse problem in DDMs is introduced to enhance the scalability of domain decomposition methods. Typically, the coarse problem has much fewer variables than the problem on the 'fine' grid. This smaller problem size allows us to apply a direct solver.

Similarly, the U-Net architecture returns a much smaller representation the further we go down the contraction path. For example, note that in the subnetwork in Figure 14, the first stack of feature maps contains $32 \times 32 \times 16 = 16,384$ whereas the deepest stack of feature maps contains $2 \times 2 \times 256 = 1024$ variables. Thus, if we can use the feature maps in this deepest layer as a representation of the input image, this guarantees that we can deal with a smaller problem that is computationally easier to solve. How we process these feature maps is explained in the following subsection, introducing the proposed communication network.

5.2.2 Communication network architecture

Figure 15 shows the proposed communication network. We discuss the network as shown in this Figure, and some of the network parameters that can be adjusted are identified.

As inputs, the communication network receives the feature maps from two subnetworks.

In this example, the last 16 feature maps of both subnetworks are selected and flattened into 1D vectors. The feature maps in this example have spatial dimensions 2×2, leading to a flattened vector of size 128, which is a fraction of $128/32^2 = 12,5\%$. The flattened vector is put through a fully connected network of three layers, all with 128 neurons. The output of the network is reshaped into two feature maps of dimensions 2×2, the same spatial dimension as the input feature maps. Then, the output is used to replace the input feature maps. The modified feature maps are sent back to the subnetworks and are used in the expansion path.

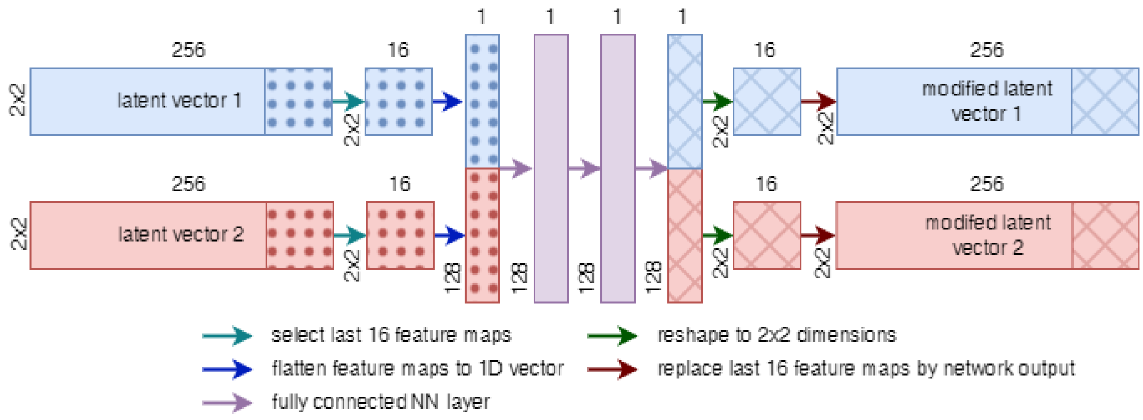


Figure 15: The proposed communication network for two subdomains. The shown network modifies two pairs of 16 feature maps with dimension 2×2 in three fully connected layers of 128 neurons. As input, the network needs 16 feature maps from both subnetworks, and it returns 32 feature maps, 16 for both subnetworks. The returned feature maps replace part of the latent space representation before this goes back through the expansion path in the subnetwork.

The proposed communication network uses only a small amount of data from both subdomains to communicate. What we investigate in this master's project is if we can make this amount of data so small that it does not lead to large communication latencies in the training and evaluation of the network. Furthermore, we consider how much this communication network affects the training procedure, as the different sub-networks need to 'wait' until the communication network has modified the latent space feature maps before they can proceed with the expansion path.

5.3 Preliminary Results

To find out if the proposed network can facilitate global communication, we performed some preliminary experiments. We constructed an artificial data set of $64 \times 32 \times 1$ gray-scale images showing two circles, both randomly located on a different half of the image. For each image, we constructed a corresponding segmentation mask. Not only the circle pixels were segmented in these masks, but also a straight line, connecting the centers of the two circles. An example image and corresponding mask are shown in Figure 16.

Then, the images were split in half horizontally and three networks were trained to perform the segmentation task. Note that for a correct segmentation, the network needs to be aware of the location of *both* circles, otherwise it is not able to segment the line between both circles. The following three networks were trained to perform this task:

1. A traditional U-Net, taking as inputs the complete 64×32 images. We will use this

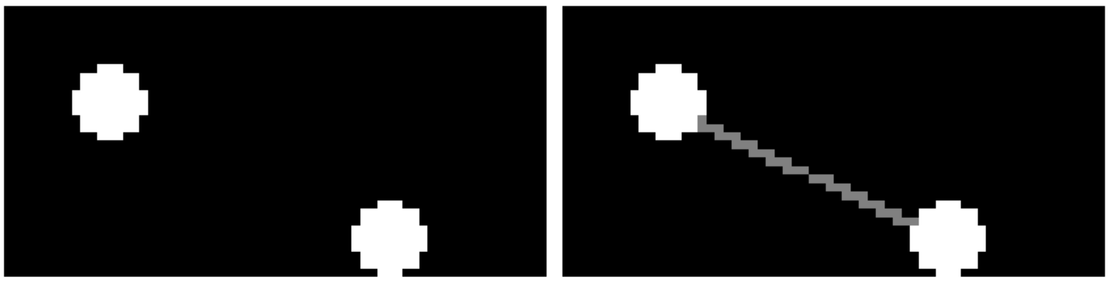


Figure 16: Example mask and image from the generated synthetic data set. The left image shows the input to the neural network, whereas the right image shows the corresponding mask, that the network should predict. Both networks receive one-half of the left image. Note that communication is necessary to draw the gray line correctly.

model as the baseline model.

2. The proposed network architecture as shown in the Figures 13 ,14, 15, consisting of two subnetworks and a communication network. Both subnetworks share their weights (i.e.: we only train one sub-network).
3. The proposed network architecture, but without a communication network. Note that in this architecture, no communication happens between both subnetworks. For this network, we expect that the line segments will be drawn randomly since the different subnets have no information on the position of the circle in the other subdomain(s).

All models were training using the Adam gradient descent algorithm with parameters $\beta_1 = 0.9, \beta_2 = 0.999$, learning rate 0.0005, mini-batch sizes of 16 data samples on the same data set of 2,000 images and corresponding masks for 250 epochs. The loss function used is the cross-entropy Loss. The networks were implemented using `PyTorch 2.0.1`. The resulting loss and accuracy ($\frac{TP}{TP+FN}$) curves are shown in Figure 17.

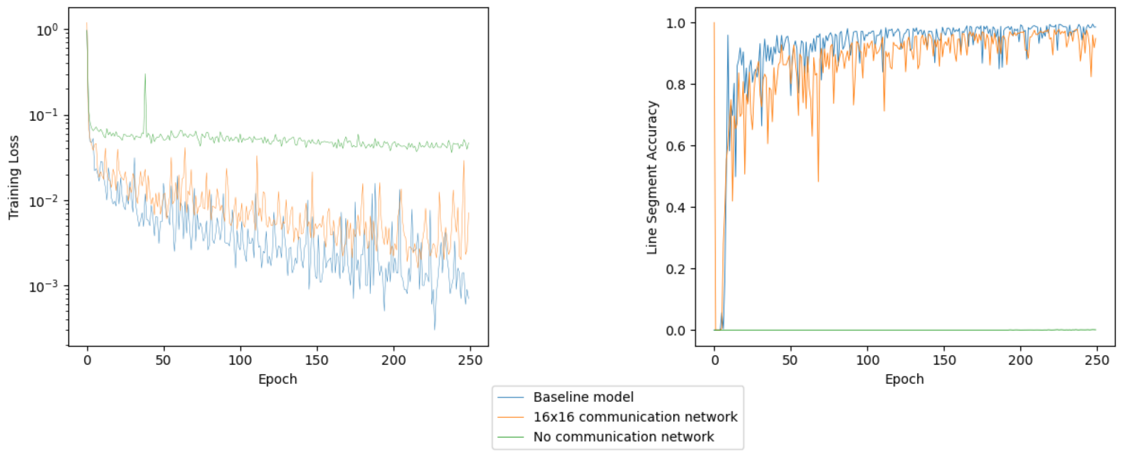


Figure 17: Left: (Cross-Entropy) training loss for the baseline model, the proposed model including communication, and the proposed model without communication between sub-networks. Right: accuracy of the line segment predictions for the three models.

This figure shows that, although the baseline model converges slightly faster, the proposed network with communication between the two sub-networks can transfer data between the two sub-networks effectively, whereas the case without no communication shows (as expected) no further convergence after some epochs. It can correctly predict the circle and background pixels, but the two sub-networks cannot communicate the location of the circles, leading to inaccurate line predictions. The maximum accuracies over the last 200 epochs from the baseline U-Net, the proposed network including communication, and the proposed network without communication are 0.995, 0.978, and 0.0014 respectively.

As we noted before, the number of communicated feature maps can be varied to see how much information is necessary for accurate predictions. To investigate this, we variy the number of feature maps passed to the communication network by each sub-network and the number of output feature maps returned by this network to each of the sub-networks. We measured the accuracy obtained by the trained network for the line segmentation. We repeated this experiment for each combination of input and output feature maps six times, to be able to measure the effect of the random weight initialization of the network. The mean accuracy for these six networks and the measured variance in these accuracies is shown in Figure 18. We note that the average accuracy for all of these models is 0.91 or

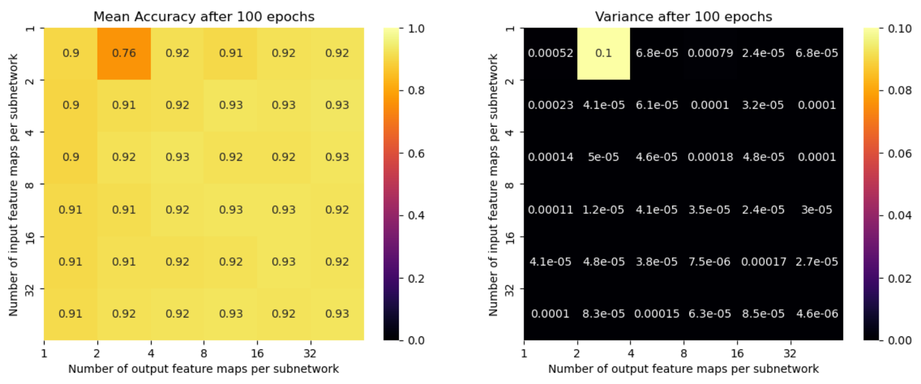


Figure 18: Left: Average accuracy of the line segment prediction after 100 epochs of training for six separately initialized and trained models. Right: variance of the accuracies. Note that the accuracy is small for most models. The outlier for 1 input feature map and 2 output feature maps is caused by one network that did not converge (but the other five converged). All models were trained using the Adam gradient descent algorithm with parameters $\beta_1 = 0.9, \beta_2 = 0.999$, learning rate 0.0005, mini-batch sizes of 16 data samples on the same data set of 2000 images.

higher, except for the case with 1 input feature map and 2 output feature maps for both sub-networks. However, it turned out that this lower accuracy was only caused by one of the models, the other five converged to values of 0.89 or higher. Furthermore, it turned out that the models with a higher number of input and output feature maps converged in a smaller number of epochs than the models with lower numbers of input and output feature maps.

These results show us that, although convergence might take more epochs of training, communicating only a few variables for each sub-network can lead to quite accurate predictions. The question for the remainder of this project will be how we can use this knowledge for the design of an optimal if possible parallel, training procedure using this network.

5.4 Research purpose and sub-questions

Although the first results show that the proposed architecture can successfully facilitate global communication between sub-networks that operate on a local part of the data, it needs to be further investigated if this architecture can be used for parallel training and/or evaluation. For example, one approach could be to train the subnetworks first only locally, and then at some point add the communication network to the training procedure for global fine-tuning.

Furthermore, it is interesting to see how the communication network functions for several types of segmentation tasks and data. The examples shown before were strongly dependent on the global information for correct line segmentation. The question is what will happen for problems where only communication around the segmentation task is more local, and where only information around the boundaries is necessary for highly accurate segmentation.

In this subsection, we identify some open questions that need further investigation to be able to conclude the effectiveness and performance of this network. Below, we give the research purpose and corresponding sub-questions for this master project. For each sub-question, we also briefly discuss how we plan to approach this.

Research purpose: *To develop and investigate a domain decomposition-inspired multi-CNN-sub-network architecture interconnected by a feature map communication network.*

Research sub-questions:

1. *How many feature maps does the communication network need for accurate global communication?* To answer this question, we will follow a systematic approach: the accuracy will be measured for different numbers of input and output feature maps to compare the performance of communication networks of different sizes. This question is very correlated with the following one.
2. *How does the amount of necessary communication change for different types of data sets and segmentation tasks?* Initially, we will use an artificially constructed dataset. The circle images as used in the first experiments can easily be made more complex by adding more circles, overlapping circles, several colors and types of lines, and/or modifying the distance between the circles. If we have a good understanding of the relation between input data and the complexity of the communication, real-life datasets can be investigated.
3. *How can we extend the proposed network architecture to more subdomains and more dimensions? How does this scale with problem size and complexity?* By adding more subdomains, the communication network will need to be adjusted. The two options we see now are (1) making the communication network larger to include feature maps from all sub-domains (leading to a higher communication network complexity) or (2) defining a subnetwork that operates on two sets of feature maps but with extra knowledge on the position of the sub-image feature maps it is processing. Both options can be tested on the artificial data set.
4. *How do the accuracy, computational performance, and memory requirements of this*

model compare to baseline methods such as the U-Net model? For all experiments, the results will need to be compared to the results obtained by using a traditional U-Net model, to see how well the proposed network performs compared to this baseline model.

5. *How can the proposed architecture be used to speed up parallel training and evaluation and/or improve accuracy for segmentation tasks?* To answer this question, all previous sub-questions need to be answered. Furthermore, it can be interesting to see how well the model performs without a communication network - and to add the communication network after local training to ensure global consistency. However, for this approach, it is necessary to investigate how we can ensure that the different sub-networks are properly scaled compared to each other and how the newly added global information can be processed by the sub-networks.

It is important to note that the sub-questions provided here are not exhaustive, and as the project progresses and more results become available, other questions could come up. Therefore, the approach followed in this thesis project needs to be adapted to the new insights and findings.

References

[1] Reza Azad, Ehsan Khodapanah Aghdam, Amelie Rauland, Yiwei Jia, Atlas Haddadi Avval, Afshin Bozorgpour, Sanaz Karimijafarbigloo, Joseph Paul Cohen, Ehsan Adeli, and Dorit Merhof. Medical image segmentation review: The success of u-net. *arXiv preprint arXiv:2211.14830*, 2022.

[2] Nathan Baker, Frank Alexander, Timo Bremer, Aric Hagberg, Yannis Kevrekidis, Habib Najm, Manish Parashar, Abani Patra, James Sethian, Stefan Wild, et al. Brochure on basic research needs for scientific machine learning: Core technologies for artificial intelligence. Technical report, USDOE Office of Science (SC)(United States), 2018.

[3] Tal Ben-Nun and Torsten Hoefler. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *ACM Computing Surveys (CSUR)*, 52(4):1–43, 2019.

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

[5] Tony F Chan and Tarek P Mathew. Domain decomposition algorithms. *Acta numerica*, 3:61–143, 1994.

[6] Tejalal Choudhary, Vipul Mishra, Anurag Goswami, and Jagannathan Sarangapani. A comprehensive survey on model compression and acceleration. *Artificial Intelligence Review*, 53:5113–5155, 2020.

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

[8] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing*, 2022.

[9] D Firsov and SH Lui. Domain decomposition methods in image denoising using gaussian curvature. *Journal of Computational and Applied Mathematics*, 193(2):460–473, 2006.

[10] Arthur Fourcade and Roman Hossein Khonsari. Deep learning in medical image analysis: A third eye for doctors. *Journal of stomatology, oral and maxillofacial surgery*, 120(4):279–288, 2019.

[11] Martin J Gander et al. Schwarz methods over the course of time. *Electron. Trans. Numer. Anal*, 31(5):228–255, 2008.

[12] Deepak Ghimire, Dayoung Kil, and Seong-heum Kim. A survey on efficient convolutional neural networks and hardware acceleration. *Electronics*, 11(6):945, 2022.

[13] LINYAN GU, WEI ZHANG, JIA LIU, and XIAO-CHUAN CAI. Decomposition and composition of deep convolutional neural networks and training acceleration via subnetwork transfer learning. *Electronic Transactions on Numerical Analysis*, 56:157–186, 2022.

[14] Gousia Habib and Shaima Qureshi. Optimization and acceleration of convolutional neural networks: A survey. *Journal of King Saud University-Computer and Information Sciences*, 34(7):4244–4268, 2022.

[15] Alexander Heinlein, Axel Klawonn, Martin Lanser, and Janine Weber. Combining machine learning and domain decomposition methods for the solution of partial differential equations—a review. *GAMM-Mitteilungen*, 44(1):e202100001, 2021.

[16] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[17] 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. *Computer Methods in Applied Mechanics and Engineering*, 365:113028, 2020.

[18] Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Artificial intelligence review*, 53:5455–5516, 2020.

[19] Axel Klawonn, Martin Lanser, and Janine Weber. A domain decomposition-based cnn-dnn architecture for model parallel training applied to image recognition problems. *arXiv preprint arXiv:2302.06564*, 2023.

[20] Pierre-Louis Lions et al. On the schwarz alternating method. i. In *First international symposium on domain decomposition methods for partial differential equations*, volume 1, page 42. Paris, France, 1988.

[21] 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*, pages 3–14. Springer, 2016.

[22] N Man, S Guo, KFC Yiu, and CKS Leung. Multi-layer segmentation of retina oct images via advanced u-net architecture. *Neurocomputing*, 515:185–200, 2023.

[23] Kyle Mills, Kevin Ryczko, Iryna Luchak, Adam Domurad, Chris Beeler, and Isaac Tamblyn. Extensive deep neural networks for transferring small scale learning to large scale systems. *Chemical science*, 10(15):4129–4140, 2019.

[24] Jinhee Park, Dokyeong Kwon, Bo Won Choi, Ga Young Kim, Kwang Yong Kim, and Junseok Kwon. Small object segmentation with fully convolutional network based on overlapping domain decomposition. *Machine Vision and Applications*, 30:707–716, 2019.

[25] Alfio Quarteroni. Introduction to domain decomposition methods. *Lecture-notes, 6th Summer School in Analysis and Applied Mathematics Rome*, pages 20–24, 2011.

[26] 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*, pages 234–241. Springer, 2015.

[27] Khemraj Shukla, Ameya D Jagtap, and George Em Karniadakis. Parallel physics-informed neural networks via domain decomposition. *Journal of Computational Physics*, 447:110683, 2021.

[28] Nahian Siddique, Sidike Paheding, Colin P. Elkin, and Vijay Devabhaktuni. U-net and its variants for medical image segmentation: A review of theory and applications. *IEEE Access*, 9:82031–82057, 2021.

[29] Azzeddine Soulaimani, Tony Wong, Y. Azami, and Amine Ben Haj Ali. An object-oriented approach for building pc clusters. *International journal on information*, 6:251–260, 01 2003.

[30] Andrea Toselli and Olof Widlund. *Domain decomposition methods-algorithms and theory*, volume 34. Springer Science & Business Media, 2004.

[31] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S Rellermeyer. A survey on distributed machine learning. *Acm computing surveys (csur)*, 53(2):1–33, 2020.