University of Stavanger

Faculty of Science and Technology

MASTER'S THESIS

Study program/ Specialization: Computer Science	Spring semester, 2023. Open / ~~Restricted access~~
Writer: Erik Finnesand	Erik Finnesand (Writer's signature)
Faculty supervisor: Mina Farmanbar and Muhammad Sulaiman External supervisor(s):	
Thesis title: A Machine Learning approach for Building Segmentation using laser data	
Credits (ECTS): 30	
Key words: Deep Learning, Convolutional Neural Networks, Building Segmentation, Aerial Images, LiDAR Data	Pages: 74…… + enclosure: 22…… Stavanger, 14/06/2023…….. Date/year

University of Stavanger

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

A Machine Learning approach for Building Segmentation using laser data

Master's Thesis in Computer Science
by

Erik Finnesand

Main Supervisor

Mina Farmanbar

Co-Supervisor

Muhammad Sulaiman

June 14, 2023

"Technology should do the hard work so people can do the things that make them the happiest in life."

Larry Page

Abstract

Buildings are essential for population information, city management, and policy-making. Various computer vision technologies have proven helpful in building-related scenarios, where segmentation has proven to be the most precise method as it highlights areas of interest. The thesis is based on the NORA MapAI competition, which tasks the participant to create machine-learning models for generating accurate segmentation masks of buildings. The competition is split into two tasks, where task 1 requires the use of only aerial images, and task 2 requires the use of LiDAR data with or without aerial images. This is done by using two different encoder-decoder architectures, U-Net, and Context-Transfer-UNet (CT-UNet), and four different backbones, ResNet50V2, DenseNet201, EfficientNetB4, and EfficientNetV2S. Data augmentation, transfer learning, model ensembles, and test time augmentation (TTA) have been used to achieve greater results. The final results of the thesis show that the proposed solution achieves state-of-the-art results, as we achieve the best score for every metric in each task compared to the other participants.

Acknowledgements

I want to thank my supervisors, associate professor Mina Farmanbar and Ph.D. student Muhammad Sulaiman, for their feedback and excellent guidance throughout this work.

I would also like to thank the Norwegian Artificial Intelligence Research Consortium (NORA) for letting me present this work at the NORA Annual Conference 2023 and granting me the award for the best oral presentation of the conference.

Contents

Chapter 1

Introduction

1.1 Motivation

Buildings are important for population information, city management, and policy-making. Computer vision technologies such as object detection, classification, and segmentation have proven helpful in urban planning, population management, city modeling, and disaster management scenarios. Segmentation has proven to be the most beneficial method, as it highlights areas of interest.

Training data is derived from real-world satellite images. Because of this, they often have varying qualities, contain noise, and consist of large class imbalances. Smaller buildings are harder to detect and may be obstructed by objects like trees, powerlines, and other structures. Different types of buildings are also found in various areas ranging from rural to urban locations. Optical issues such as shadows being mistaken for buildings, reflections, and image perspective are also issues. Because of all these factors acquiring accurate segmentation masks of buildings is challenging.

1.2 MapAI: Precision in Building Segmentation

This thesis is based on the MapAI: Precision in Building Segmentation competition hosted in 2022 by the Norwegian Artificial Intelligence Research Consortium (NORA) in collaboration with the Centre for Artificial Intelligence Research at the University of Agder (CAIR), the Norwegian Mapping Authority, AI:Hub, Norkart, and The Danish Agency for Data Supply and Infrastructure [1].

NORA provides the participants with a dataset containing a training, validation, and test split. More of the dataset can be seen in chapter 3.1. The competition is split into two different tasks.

- `Task 1: Aerial Image Segmentation Task:` The first task aims to develop a machine-learning model for generating accurate segmentation masks of buildings using only aerial images.
- `Task 2: Laser Data Segmentation Task:` The second task aims to develop a machine-learning model for generating accurate segmentation masks of buildings using laser data with or without aerial images.

The competition had 11 attendees. The different solutions of the groups are evaluated on the test set using IoU and BIoU, which can be read about in chapter 2.10. The average of these two metrics is used as a total score. This is done for both tasks. The group's final score is then calculated by taking the average of both total scores. The following table shows the results from the top 3 groups.

		Task 1			Task 2			
Placement	**Team**	**IoU**	**BIoU**	**Total Score**	**IoU**	**BIoU**	**Total Score**	**Final Score**
1	FUNDATOR	0.7794	0.6115	0.6955	0.8775	0.7857	0.8316	**0.7635**
2	HVL-ML	0.7879	0.6245	0.7062	0.8711	0.7504	0.8108	**0.7585**
3	DEEPCROP	0.7902	0.6185	0.7044	0.8506	0.7461	0.7984	**0.7514**

Table 1.1: Results from the top 3 groups of the MapAI competition

1.3 Objectives

The thesis's main objective is to develop an approach that accurately segments buildings and challenges the top competitors of the MapAI competition. This will be done by implementing two different encoder-decoder neural network architectures, U-Net and CT-UNet. These architectures are implemented using different backbones. Techniques such as Test Time Augmentation (TTA) and model ensemble will be used to potentially improve the different model's predictions.

The following research questions (RQ) will be explored through the thesis and answered in chapter 6.

- `RQ1:` How does the addition of laser data affect the performance of the models?
- `RQ2:` How does the performance of U-Net and CT-UNet compare for building segmentation?

- `RQ3:` How does the performance of ensembled models compare to individual models for building segmentation?
- `RQ4:` Does TTA improve the predicted results in building segmentation?

1.4 Approach and Contributions

The approach and contributions of this thesis are split into three parts. The first part is pre-processing, where the dataset is pre-processed to fit the models used for the thesis. The second part is model creation and training, where the different model architectures used for the thesis are made and trained using data augmentation and various callback methods. The third part is testing and evaluation, where the model predictions are compared to the ground truth masks and evaluated using different metrics. This part evaluates single model predictions, single model predictions using TTA, model ensemble predictions, and model ensemble predictions using TTA.

1.5 Environment

The approach for the thesis was implemented using Python [2]. Python is a high-level, general-purpose programming language. A high-level programming language has strong abstractions from the details of the computer. This means that hardware operations, such as memory allocation, are automated. Being a general-purpose programming language means that Python is made for developing software in various application domains. Python contains a lot of useful external libraries. The following libraries are the main libraries used for the proposed approach.

1.5.1 TensorFlow

TensorFlow is a machine learning and artificial intelligence library. It supports various ways to prepare data easily, as well as creating, compiling, training, and testing models [3]. With the help of TensorFlow's interface, which is called Keras [4], it is possible to create custom artificial neural networks(ANN).

This approach uses TensorFlow to import datasets from the file directory, augment training images, create neural networks, and compile, train, and test these neural networks.

1.5.2 Datasets

Datasets is a library used to efficiently download and preprocess public datasets, provided on the Huggingface Dataset Hub [5]. Huggingface Dataset Hub is a database containing over 26.000 datasets for computer vision, audio, and natural language processing problems [6].

As the dataset used for this thesis is available on Huggingface, the library has been used to download and save the dataset.

1.5.3 NumPy

NumPy is a mathematical library that adds support for large, multi-dimensional arrays and matrices, as well as high-level mathematical functions to operate on these arrays [7].

NumPy is used to operate on the LiDAR data in the dataset, augment images during test time augmentation and operate on the predictions given by the models.

1.5.4 OpenCV

OpenCV is a library that supports real-time computer vision and has different functions related to image processing [8].

OpenCV is used for reading, writing, and resizing images.

1.5.5 Matplotlib

Matplotlib is a library for creating static, animated, and interactive visualizations in Python [9].

Matplotlib is used for plotting the images that OpenCV has read and for plotting graphs.

1.6 Outline

The thesis starts with a brief explanation of the motivation and objectives. It also mentions the competition the thesis is based on and the environment used for the approach. The remaining part of this thesis is structured into five different chapters.

- The second chapter presents relevant background theory related to the methods used in the thesis.
- The third chapter presents the dataset used for the thesis and the pre-processing done on the dataset.
- The fourth chapter presents the approach for the thesis. This chapter explains the model architectures and methods used while training the different models.
- The fifth chapter presents and discusses the experimental setup and results for the thesis.
- The sixth chapter presents the conclusion of the work done for the thesis.

Chapter 2

Background

This chapter contains relevant background theories related to this thesis, such as the general concept of deep learning, different techniques used to train and test a model, and metrics used for evaluation.

2.1 Light Detection and Ranging

Light detection and ranging (LiDAR) is an active remote sensing technology that transmits laser pulses towards an area of interest and receives the light scattered by the objects in the area [10]. By doing this, it is possible to create accurate 3-dimensional representations of an environment. This representation contains x- and y- and z-coordinates representing the location and height of the detected objects.

Airborne LiDAR is commonly used to acquire LiDAR data over a broader area, such as cities. These systems are typically mounted on planes, helicopters, or drones and contain four main components, a laser scanner, a global positioning system (GPS), an Inertial Measurement Unit (IMU), and a computer. The airborne vehicle flies over an area and sends pulses of near-infrared light from side to side. The reflected light is then caught by a sensor that records the time from the laser pulse to the return of reflected light. The GPS tracks the altitude and location of the aerial vehicle, while the IMU tracks the orientation and speed of the aerial vehicle. The information from the GPS, IMU, and the time recorded by the sensor is used to track the actual position of where the pulse is reflected from. A computer is then used to keep track of all this information.

2.2 Artificial Neural Networks

An Artificial Neural Network (ANN) is a computational network whose structure is inspired by the biological human brain. Like the human brain, which has multiple interconnected neurons, an ANN consists of multiple interconnected artificial neurons, also known as nodes. The connection between the nodes, known as edges, contains an associated weight. The weight is adjusted during training to help determine the importance of the signal sent through the edge with respect to the final output of the network. The signal transmitted through the edge is multiplied by the weight associated with the edge to determine its value. The ANN architecture consists of multiple layers containing nodes, where the input of a node is the output of all the nodes in the previous layer, as seen in figure 2.1.

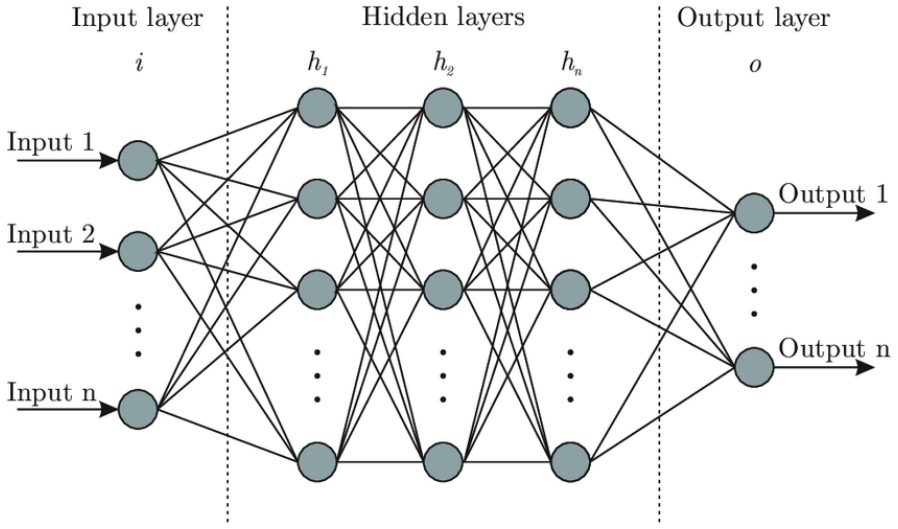


Figure 2.1: Architecture of an ANN [11]

The three types of layers in an ANN are the input layer, hidden layer, and output layer. An ANN consists of a single input layer, a single output layer, and a minimum of one hidden layer. The input layer processes the input of the ANN, such as the pixel values in an image. The hidden layer analyzes the output from the input layer or another hidden layer to identify features and use these to correlate between the given input and correct output. The output layer analyzes the output from the last hidden layer and is used to determine which class the data from the input layer belongs to. This final value is represented as a probability between 0 and 1. Each layer also consists of a bias node. The bias is a constant added to the node's input in the next layer and is used to offset the outputs of these nodes.

ANNs can be considered function estimators, where any mathematical function can be approximated with enough nodes and layers.

2.2.1 Activation Functions

When calculating the output of a node, the sum of inputs is added with the bias as seen in equation 2.1, where z is the output of the node, x is the input value of the node, w is the weight corresponding to the input value, and b is the bias.

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

The problem with using z as the input to the next layer is that the values are linear, meaning the network will act as a linear regression model. If this is the case, the network won't be able to solve more complex problems that require non-linearity. To solve this, activation functions are used on the output of the nodes.

Activation functions transform the linear output of the nodes to non-linear values. By doing this, it also decides whether a node should be activated. As the output from a node can be any real number, it could cause the weight to explode in size if the output is large enough. Most activation functions solve this by squeezing the output value into a specific range of numbers. An activation function must also be differentiable to perform backpropagation (see chapter 2.2.3). The following activation functions were used in this thesis.

Sigmoid

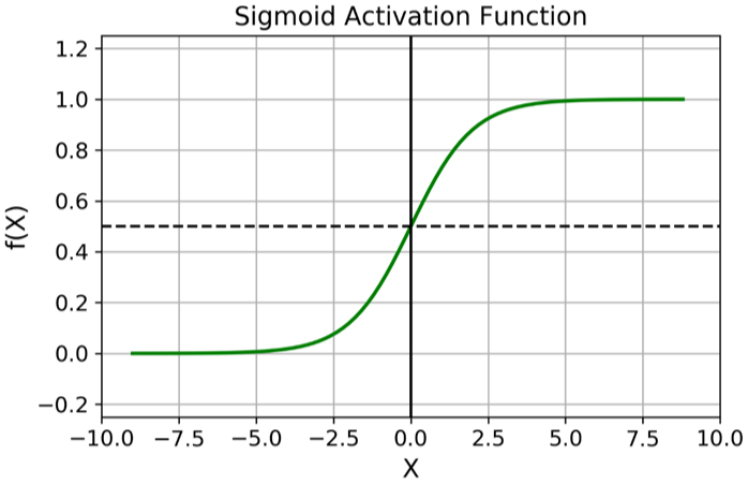


Figure 2.2: Curve of the Sigmoid activation function[12]

Sigmoid is a non-linear activation function with an S-shaped curve, as seen in figure 2.2. The activation function maps every input value to a value between 0 and 1. Because of

this, it is often used to predict the probability map of binary classification problems in the output layer. The sigmoid function is defined by equation 2.2.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

Sigmoid isn't used in the hidden layer because of the vanishing gradient problem. This problem occurs during backpropagation when trying to find the gradient of the loss function. Since the sigmoid activation function has a maximum derivative of 0.25 at $x = 0$, the derivative will be very small at very low or high x values. Because of this, the gradient of the sigmoid function can become so small that it doesn't have any impact while updating the weights of the network. This will make it difficult for the network to learn. A way to prevent the vanishing gradient problem is to initialize the weights of the layers using sigmoid activation from a truncated normal distribution with a mean of zero and a standard deviation of $\sqrt{2/(n_{in} + n_{out})}$, where n_{in} is the number of inputs and n_{out} is the number of outputs of the layer. This initialization method is called Glorot/Xavier normal initialization [13].

Rectified Linear Unit

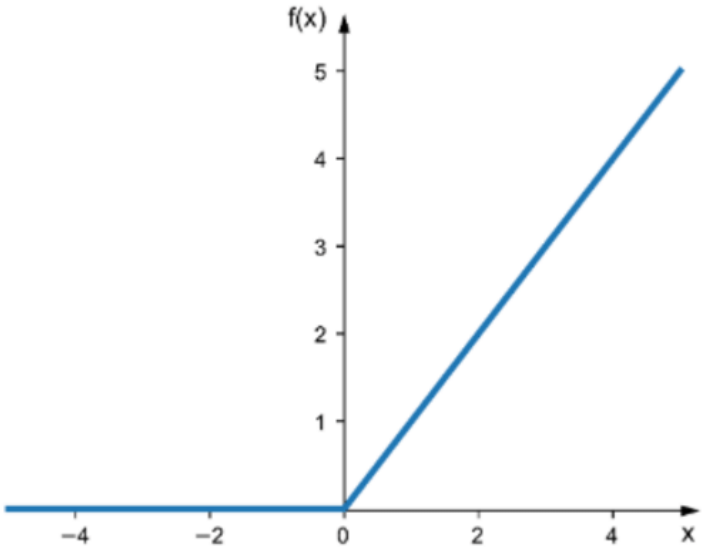


Figure 2.3: Curve of the ReLU activation function[14]

Rectified Linear Unit (ReLU) is the most commonly used activation function in the hidden layers. The function maps every input value to a value between 0 and infinity, as seen in figure 2.3. The ReLU activation function is defined by equation 2.3.

$$f(x) = max(0, x) \tag{2.3}$$

The derivative of ReLU is 0 when $x < 0$ and 1 when $x > 0$. Because of this, it doesn't suffer from the vanishing gradient problem. This also makes ReLU computationally efficient, as it is easy to compute its gradient. However, ReLU suffers from the dying ReLU problem. This happens when the output of a node stays negative during the training, which causes the activation value always to be 0. If this happens to enough nodes, it will decrease the accuracy of the network. A way to prevent the dying ReLU problem is to initialize the weights of the layers using ReLU activation from a truncated normal distribution with a mean of zero and a standard deviation of $\sqrt{2/n}$, where n is the number of inputs in the layer. This initialization method is called He normal initialization [15].

2.2.2 Loss Function

When training a network, it must receive feedback to know how well it performs. This is done with the help of a loss function. The loss function computes the difference between the current output of the network and the expected output. The goal is to minimize the loss as much as possible. The output of the loss function is used as a feedback mechanism to the optimizer, so it can correctly update the network's weights. The loss function needs to follow two properties. It has to be globally continuous and differentiable.

There are two main categories for loss functions, binary and categorical. Binary loss functions are used when only two classes are present. Categorical loss functions are used when more than two classes are present. As the thesis explores a two-class problem (building and background), three binary loss functions are tested.

Binary Cross-Entropy

Binary Cross-Entropy (BCE) is a loss function based on entropy, a measure of uncertainty. Equation 2.4 shows the formula of BCE, where n is the number of data samples, Y_i is the true label (either 0 or 1), and $\hat{Y}_i$ is the predicted output of the model (a value between 0 and 1).

$$L_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}(Y_i \cdot ln(\hat{Y}_i) + (1 - Y_i) \cdot ln(1 - \hat{Y}_i)) \tag{2.4}$$

The first term in the equation penalizes the model when the true label is 1, and the model prediction is low. The second term penalizes the model when the true label is 0, and the prediction is high. The output value of BCE is a value between 0 and infinity.

Dice Loss

Dice loss is a loss function commonly used in image segmentation problems. It is based on the dice coefficient, which calculates twice the overlap of a ground truth mask and a prediction mask divided by the total coverage of these two masks, as seen in figure 2.4.

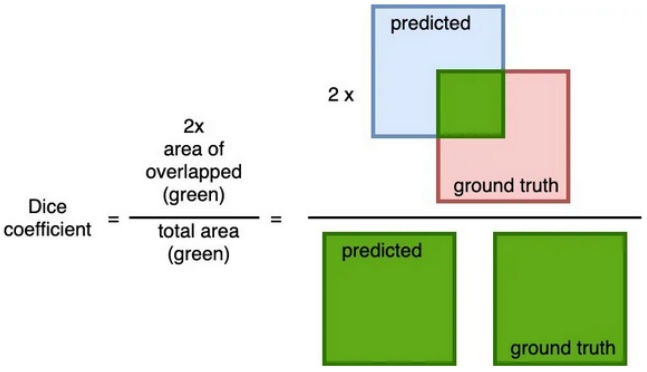


Figure 2.4: Illustration of dice coefficient[16]

The dice coefficient returns a value between 0 and 1, where 1 is a perfect prediction, and 0 is a prediction where all pixels are classified to the wrong class. As we want a decreasing loss function, the dice loss can be represented as shown in equation 2.5 where TP (True Positive) is the number of pixels correctly classified as positive, FP (False Positive) is the number of pixels incorrectly classified as positive, FN (False Negative) is the number of pixels incorrectly classified as negative and ϵ is a smoothing factor.

$$L_{Dice} = 1 - \frac{2TP + \epsilon}{2TP + FP + FN + \epsilon} \tag{2.5}$$

Jaccard Loss

Like Dice loss, Jaccard loss measures the overlap between the ground truth and predicted masks. The difference between Dice and Jaccard loss is that Jaccard loss gives equal weight to both classes, while dice loss gives more weight to the foreground class. This is especially true if the background class dominates over the foreground class. The formula for Jaccard loss is shown in equation 2.6.

$$L_{Jaccard} = 1 - \frac{TP + \epsilon}{TP + FP + FN + \epsilon} \tag{2.6}$$

2.2.3 Optimizers

As mentioned in chapter 2.2.2, the goal of any network is to minimize its loss. The optimizer does this in response to the output value of the loss function. Backpropagation is performed for every layer in the network. Backpropagation computes the gradient of the loss surface concerning the weights and biases of the network. This is done by iterating through the network, starting from the last layer and ending at the first. After getting the gradient, the optimizer algorithm computes the new weights. This is done using a type of gradient descent. Equation 2.7 shows the formula for gradient decent, where $Wnew$ is the updated weights, $Wold$ is the weight from the last iteration, γ is the learning rate, and $\nabla f(W_{old})$ is the direction of steepest descent found by the backpropagation.

$$W_{new} = W_{old} - \gamma \cdot \nabla f(W_{old}) \tag{2.7}$$

When doing gradient descent on a loss surface, the goal is to reach the lowest point on the surface, called the global minima. A problem with normal gradient descent is that it goes toward the direction of the steepest descent. Depending on where on the loss surface the starting point is, it could cause the model to become stuck at a local minima, as seen in figure 2.5.

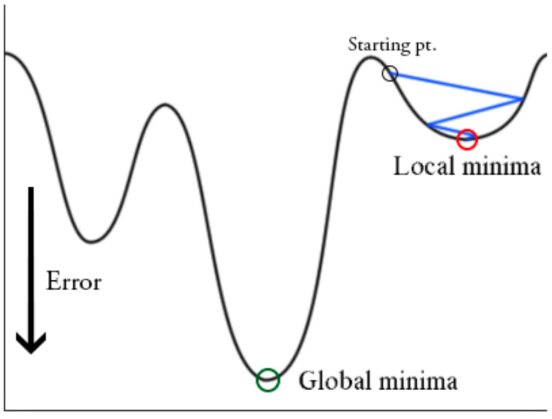


Figure 2.5: Illustration of training progress trapped in a local minima [17]

Another problem with gradient descent is that it could reach a saddle point on the loss surface, where the gradient is 0. This will cause the model to stop learning. Hyperparameters for gradient descent could solve these problems. One of these hyperparameters is momentum. Momentum is usually implemented using the exponential moving average

of the previous steps. An exponential moving average is a moving average that assigns a greater weight to the most recent values. Using momentum, it is possible to escape a small local minima and move on the loss surface when it is flat. Adam is an optimization algorithm that uses a form of momentum and is used for the proposed approach.

Adam

Adam is an algorithm for first-order gradient-based optimization of stochastic objective functions based on adaptive estimates of lower-order moments [18]. Adam is an adaptive algorithm, meaning it maintains a different learning rate for each weight in the neural network. The algorithm uses the exponential moving average of the gradient to set the learning rate. The exponential moving average is calculated using the gradient's first moment(mean) and second moment(uncentered variance).

The formula for the first moment of the gradient can be seen in equation 2.8, where m_{new} is the new first moment, m_{old} is the first moment from the previous iteration, β_1 is the first-moment exponential decay rate set at 0.9 and ∇g is the gradient of the current batch.

$$m_{new} = \beta_1 \cdot m_{old} + (1 - \beta_1) \cdot \nabla g \tag{2.8}$$

The formula for the second moment of the gradient can be seen in equation 2.9, where v_{new} is the new second moment, v_{old} is the second moment from the previous iteration, β_2 is the second-moment exponential decay rate set at 0.999 and ∇g^2 is the squared gradient of the current batch.

$$v_{new} = \beta_2 \cdot v_{old} + (1 - \beta_2) \cdot \nabla g^2 \tag{2.9}$$

As m and v have been initialized at zero, they tend to be biased towards 0. Adam fixes this by computing bias-corrected values for these in the early iterations, as seen in equations 2.10 and 2.11, where t is the current time step.

$$m_{new} = \frac{m_{new}}{1 - \beta_1^t} \tag{2.10}$$

$$v_{new} = \frac{v_{new}}{1 - \beta_2^t} \tag{2.11}$$

The weights are then updated using formula 2.12, where w_{new} is the updated weight, w_{old} is the weight from the last iteration, γ is the learning rate, and ϵ is a hyperparameter set at 10^{-7}.

$$w_{new} = w_{old} - \gamma \cdot \frac{m_{new}}{\sqrt{v_{new}} + \epsilon} \tag{2.12}$$

2.2.4 Learning Rate

The learning rate is a hyperparameter of the optimizer. It is used to control the rate of adjusting weights done by the optimizer, as seen in equation 2.7. The optimal learning rate is determined by the loss surface and optimizer used. Figure 2.6 shows the effect of different learning rates.

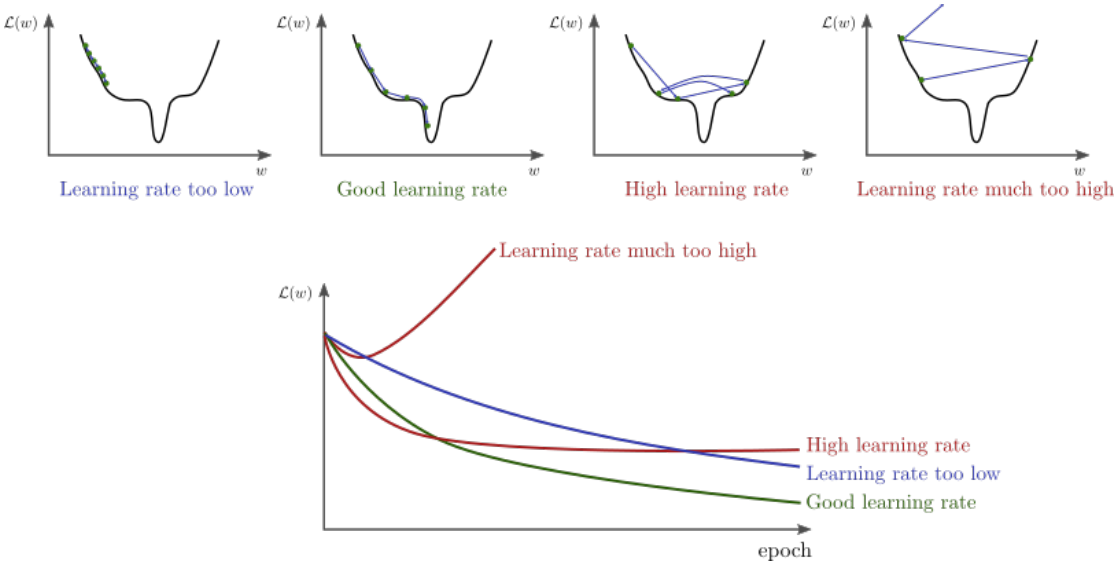


Figure 2.6: Illustration of different learning rates [19]. The top row shows the gradient descent on the loss surface. The bottom row shows the validation loss curve.

If the learning rate is too low, it will take longer to converge than the optimal learning rate. It may also get stuck in a local minima, never to be able to reach the global minima. A high learning rate may never hit the global minima, as the weights are too aggressively adjusted, causing it to jump over the minima. If the learning rate is way too high, it will diverge. This is because the model will unlearn what it has learned in previous epochs. A good learning rate steadily declines on the loss surface before flattening out and reaching convergence.

2.2.5 Epochs and Batch Size

An epoch is one training cycle using the entire training dataset. In an epoch, all data is used exactly once. The data is fed into the network in a different order for each epoch to help the network generalize. This can be done in multiple batches with a given batch size.

Batch size defines the number of training samples propagated through the neural network before updating its weights. It is a hyperparameter of gradient descent.

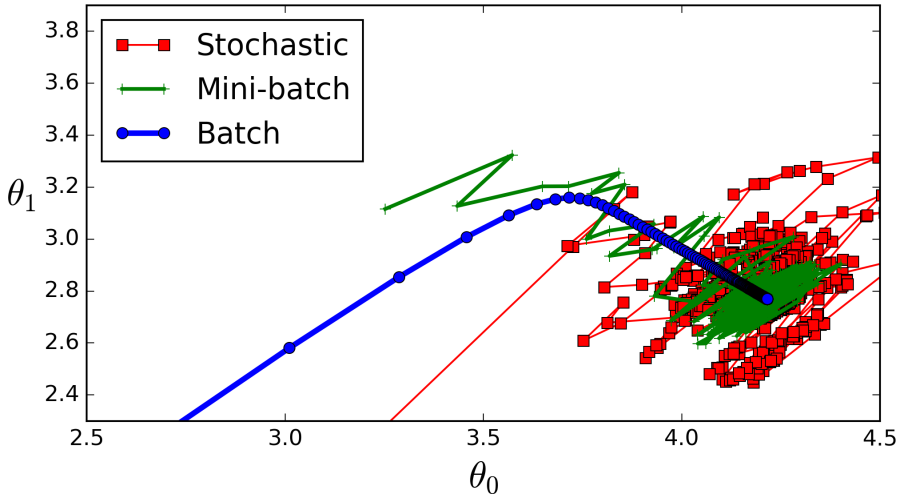


Figure 2.7: Illustration of gradient descent using different batch sizes [20]. Stochastic includes one data sample per batch, Batch includes the whole training set, and Mini-batch is somewhere between.

Figure 2.7 shows the effect of different batch sizes when finding the global minima on the loss function. A large batch size(blue) uses fewer steps to reach the minima with a more stable gradient direction. However, it is expensive on the GPU's memory. A large batch size also causes the network to have a lower ability to generalize, which can result in models being caught in a local minima [21]. Lower batch sizes(green and red) typically train the network faster because its weights are updated more frequently. However, a smaller batch size gives a worse estimation of the gradient. Because of this, the direction of the gradient will fluctuate more, as seen in figure 2.7. This isn't necessarily a negative trait, as it can free the model if trapped in a local minima.

2.2.6 Regularization

A network is trained using a training set and a validation set. The validation set is used to evaluate the model after each epoch. A validation set is used to keep track of the model's training and to see if overfitting occurs. If the performance of the validation set starts to decrease or flattens while the performance of the train set improves, overfitting

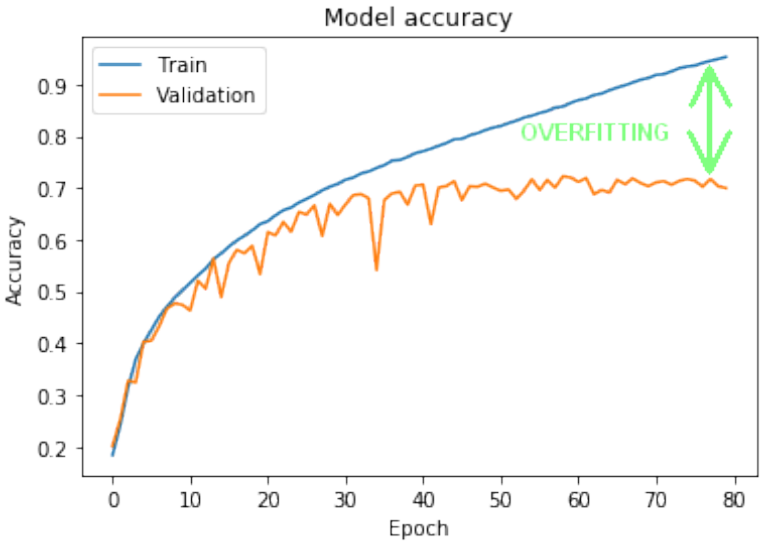


Figure 2.8: Illustration of overfitting [22]

has occurred, as seen in figure 2.8. It happens because the model learns from noise and other details in the training set rather than the underlying patterns that generalize to new data. Some of the reasons for overfitting are using a too complex model, incorrect choice of hyperparameters, and lack of training data. Some techniques to reduce overfitting are data augmentation (see chapter 2.7), early stopping (see chapter 4.2.2), and regularization techniques such as batch normalization.

Batch Normalization

When training a network, two objects of the same class with a different input distribution could be mistaken for two completely different classes. This is known as the covariate shift. Batch Normalization (BN) reduces the possibility of covariate shifts. BN improves the training speed, training stability, and performance of a network [23]. BN normalizes the input values so that the output values maintain a mean close to 0 and a standard deviation close to 1. This is done using the mean and standard deviation of the current batch of inputs, and two learnable parameters, which scale and shift the normalized values. Equation 2.13 shows the formula for BN where y is the normalized batch, x is the current batch, μ is the batch mean, σ^2 is the batch standard deviation, γ is the learnable parameter for the scale factor, β is the learnable parameter for the shift factor and ϵ is a small constant.

$$y = \gamma \cdot \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{2.13}$$

BN allows models to use much higher learning rates without negatively affecting the results. It makes working with many activation layers easier, reduces overfitting, and reduces the need for dropout layers in a network.

2.3 Convolutional Neural Networks

Convolutional Neural Network (CNN) is a class of ANN. They are specifically designed to process pixel values in images and are mainly used for image recognition. A CNN consists of an input layer, a set of feature learning layers, and a classification layer, as seen in figure 2.9.

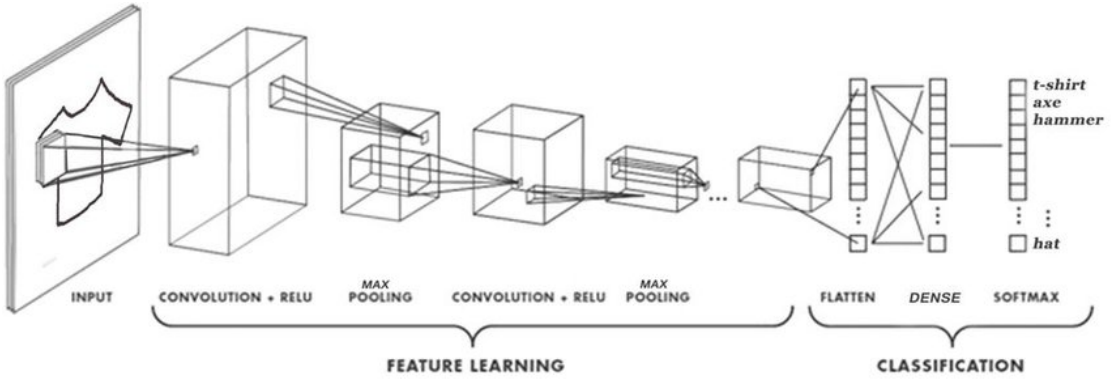


Figure 2.9: Architecture of a CNN [24]

The input layer is a 2- or 3-dimensional matrix depending on the type of image used. The size of the input layer is $hxwxd$ where h is the height, w is the width, and d is the number of channels in the input image. The feature learning layers contain convolutional layers, activation functions, and pooling layers. This set of layers learns features in the image. At the beginning of this set of layers, low-level features such as edges, colors, and textures are learned. As we go deeper, the low-level features are used to find high-level features, such as objects. The classification layer is used to decide which class the input image belongs to and is decided based on the features learned. Here the last set of feature maps is transformed into a one-dimensional array before being fed into a dense, fully connected layer, which are layers containing neurons, as explained in chapter 2.2. After this, the neuron's output is fed into the last layer, which outputs a probability vector of which class the network thinks the image belongs.

2.3.1 Convolutional Layer

Convolutional layers perform convolution on an image matrix and return a feature map. The convolution operation involves sliding a kernel over the image matrix and computing

the dot product between the kernel and the corresponding region of the image matrix. The kernel is a *hxwxd* matrix used as a pattern detector, where h is the height, w is the width, and d is the depth of the kernel. The depth of the kernel has to be the same as the depth of the image. After the dot product is computed, it is stored in a new matrix, representing the feature map. Equation 2.14 shows how direct convolution(non-flipped kernel) is performed on a 2-dimensional matrix where I' is the feature map, I is the input image, G is the kernel, w is the width of the kernel, and h is the height of the kernel.

$$I'(x,y) = I(x,y) \circledast G(x,y) = \sum_{i=0}^{w-1} \sum_{j=0}^{h-1} I(x+i, y+j) G(i,j) \tag{2.14}$$

The filters are initialized at completely random values that are updated during the training. Compared to a regular ANN, the filters can be considered weights on edges connecting neurons.

The operation is defined by two parameters, stride (s) and padding (p). Stride is the distance the kernel shifts when sliding across the input image. Padding is the width of zero padding added to the input image. The convolution starts at the upper left corner of the image matrix and shifts s columns to the right for every turn. For each turn, the dot product of the kernel and corresponding region of the image matrix is stored in a new matrice. When the kernel reaches the far right of the image matrix, it shifts down s rows before moving back to the left side of the image matrix to repeat the process. After the kernel has convolved the whole image matrix, the layer stores the matrix containing the convolution results. If the convolution is performed on a multi-dimensional image, the results from each channel are added together to form this matrix. This is the new representation of the image and is done for every kernel in the layer. Figure 2.10 illustrates convolution using a $3x3x1$ kernel done on a $7x7x1$ image, with a stride of 1 and no padding.

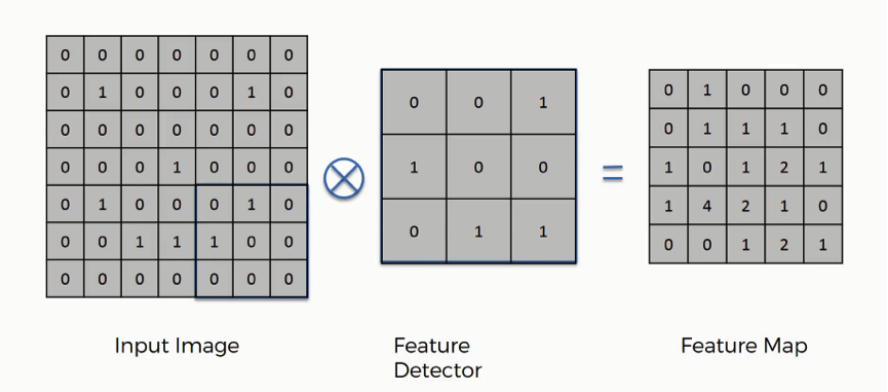


Figure 2.10: Illustration of convolution [25]

The figure shows that the feature map is smaller than the input image. This is because the kernel can't perform convolution on the edge pixels, as a part of the kernel matrix will be out-of-bounds. Zero padding is usually added to the input image before the convolution operation to keep the same size on the feature map.

The size of the convolutional layer's final output is the feature map's size times the number of kernels used in the layer. A convolutional neural network consists of multiple of these convolutional layers. The first layers normally start with only a few basic kernels to detect large notable edges in the image. As we go deeper into the network, more complex kernels are used to detect more sophisticated patterns gradually.

2.3.2 Transposed Convolution Layer

Transposed convolution layers perform transposed convolution on an input image. Transposed convolution is a technique used to upsample an image using trainable kernels. This means that the spatial dimensions of the output feature map will be larger than the spatial dimensions of the input feature map. Just as with regular convolution, the operation is defined by two parameters, stride (s) and padding (p). When doubling the spatial dimensions of an image, which is the most common thing to do in this layer, $s = 2$ and $p = 1$.

The first step of the operation involves adding $s - 1$ number of zeros between each row and column of the input image. This is followed by adding a zero padding of width $k - p - 1$ around the image, where k is the width of the kernel. The last step is to carry out a standard convolution of the resulting image matrix with a stride of 1. This process is illustrated in figure 2.11, where the input image has a spatial dimension of $4x4$, the kernel has a spatial dimension of $3x3$, $s = 2$, and $p = 1$.

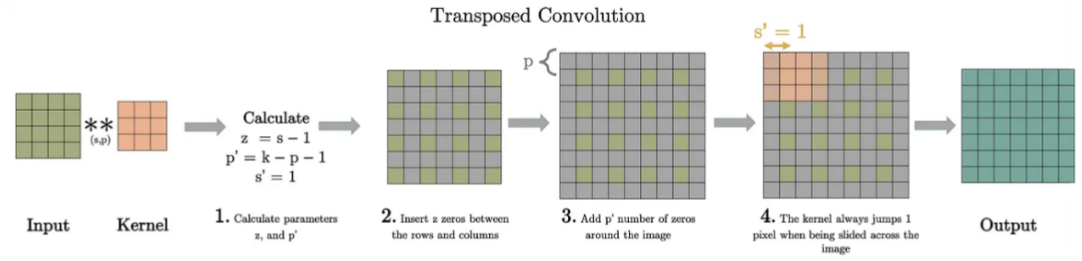


Figure 2.11: Illustration of transposed convolution [26]

The spatial dimension of the output map in this layer depends on all parameters explained and is given by equation 2.15.

$$output_{dim} = (input_{dim} - 1) \cdot s + k - 2 \cdot p \tag{2.15}$$

2.3.3 Pooling Layer

A limitation of convolution is that it provides the exact position of the feature in the feature map. Because of this, a small translation in the input will give a completely different feature map [27]. To solve this problem, pooling layers are used to introduce translational invariance. Pooling is a downsampling method used to reduce the spatial resolution of the input image. A downsampled input image will make the network more computationally efficient as we go deeper into the network where more kernels are used for the convolutional layers.

The pooling operation divides the image into multiple non-overlapping nxm regions, where nxm represents the pool size. What happens next depends on the pooling technique used. The two main pooling methods used in the field are max pooling and average pooling. The max pooling operation takes the maximum value of each region and places them in a new matrix. The goal of max pooling is to identify the most important feature in each pooling region. The average pooling operation takes the average value of each region and places them in a new matrix. Average pooling aims to create a more generalized version of the input, where each feature contributes to the downsampled image. The new matrix represents the downsampled representation of the input image. Figure 2.12 illustrates max pooling performed on a $4x4$ matrice using a pool size of $2x2$.

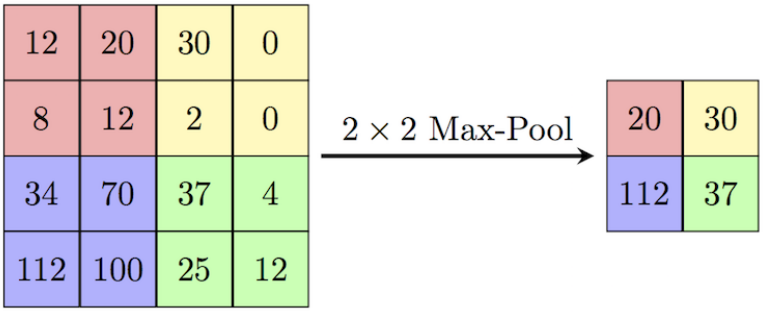


Figure 2.12: Illustration of max pooling [28]

2.3.4 Global Average Pooling Layer

Global Average Pooling (GAP) is a pooling operation that reduces the spatial dimensions of feature maps into a single value [29]. This is done by taking the average value of a feature map across all spatial locations. When done on multiple feature maps, it returns a vector of $nx1$, where n is the number of feature maps. This vector represents the contribution of each feature map. It can be used to identify the most important features in a given image that are most relevant to the final prediction.

2.4 U-Net

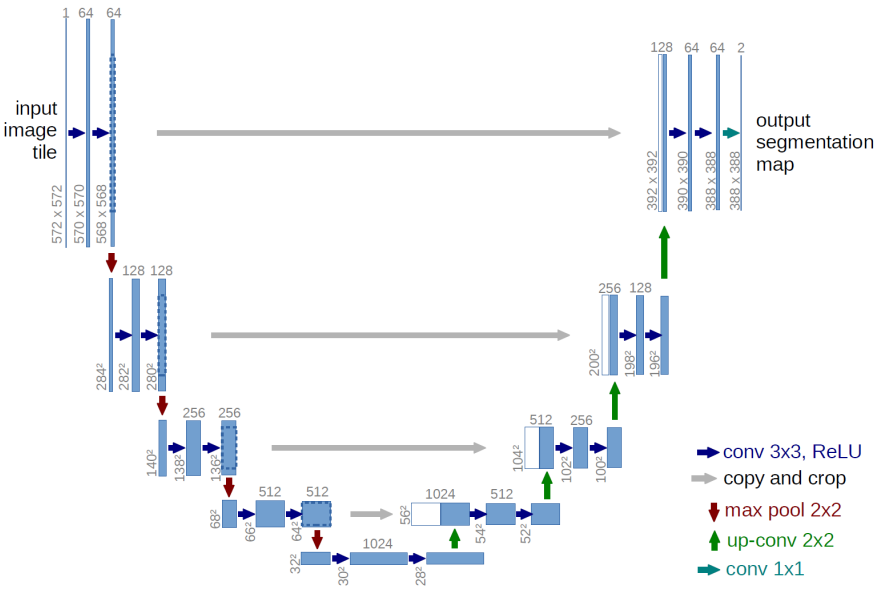


Figure 2.13: U-Net architecture[30]

U-Net is a fully convolutional network(FCN) used for semantic segmentation. U-Net gets its name from the U-shape of the architecture. The architecture was developed in 2015 to segment biomedical images [30]. As seen in figure 2.13, the architecture consists of a contracting and expansive part. The original paper uses an input size of $572x572x1$ and an output size of $388x388x2$. The channel in the output image represents the probability map to which class the individual pixels belong.

The contracting part, or the encoder as it is usually called, corresponds to the architecture of a standard CNN. The job of the encoder is to capture features in the input image. This is done in four layers, each consisting of two 3x3 convolutions using ReLU as an activation function. After the two convolution operations, max-pooling decreases the image size by a factor of 2. The number of convolution kernels starts at 64 at the top of the network and is doubled for each network depth. This means that the input image in the first depth will go from a channel size of 1 to 64. This continues to the bottom of the network, called the bottleneck. As the original U-Net paper doesn't use zero-padding in the convolutional layers, the spatial dimension of the image will be reduced by two pixels after each layer.

The expansive part is often called the decoder. The decoder's job is to precisely localize where in the image the feature from the encoder side of the network belongs. This part of the network is similar to the encoder part, whereas transposed convolution is used instead of max-pooling to increase the image size by a factor of 2. When performing transposed convolution, three-fourths of the pixels in the image from the encoder side

of the network before max-pooling is lost. To counteract this, the last feature maps in each depth at the encoder side of the network are concatenated with the output of the transposed convolution operation on the decoder side of the network at the same depth. This is called skip-connection and is mainly done to make the network better at determining the feature's location.

The final operation in the network is done for the output layer. This layer consists of a 1x1 convolution with a kernel size of 1. This is used to reduce the kernel dimensions from 64 to 1. Since this is a binary classification problem, this layer uses sigmoid as an activation function.

2.5 Context-Transfer-UNet

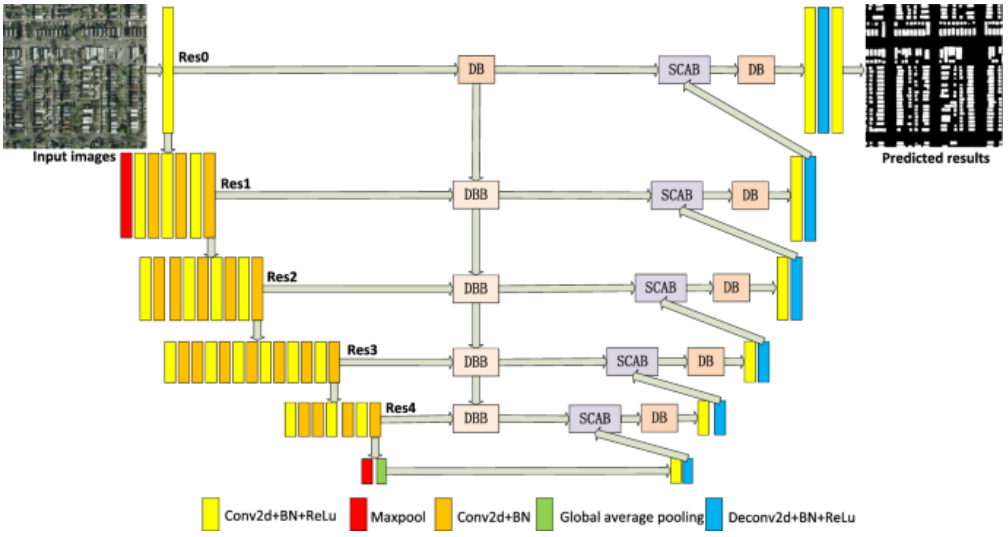


Figure 2.14: CT-UNet architecture[31]

Context-Transfer-UNet (CT-UNet) is a modification of U-Net. It was developed in 2021 to improve the segmentation of remote sensing images, compared to normal U-Net [31]. The creators of CT-UNet replace the whole encoder part of the network with a backbone using ResNet-34. CT-UNet is also one level deeper than U-Net, which lets it capture more abstract and complex features. As seen in figure 2.14, CT-UNet introduces some extra blocks in the skip-connections. The first block is called the Dense Boundary Block (DBB), which is made up of a Dense Block (DB) and a Boundary Block (BB). The last block is called Spatial Channel Attention Block (SCAB).

2.5.1 Dense Boundary Block

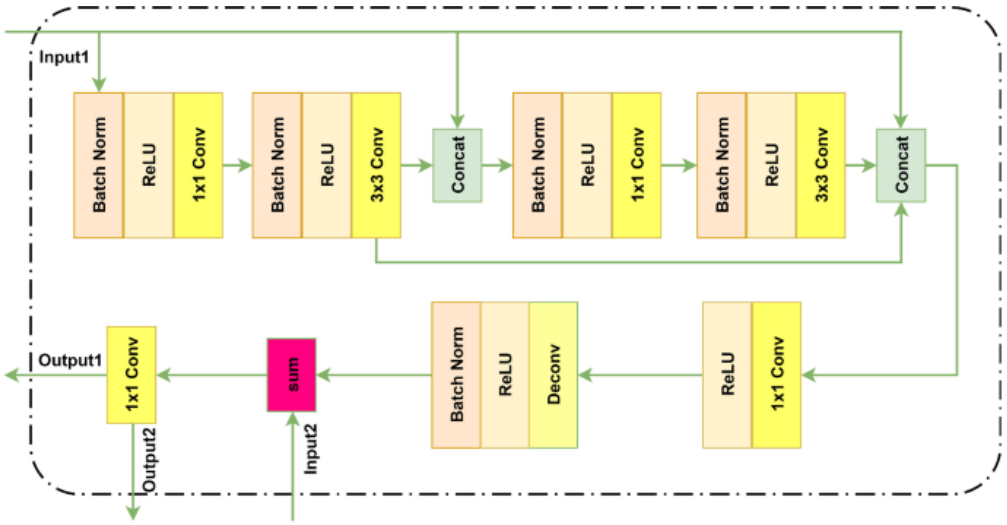


Figure 2.15: Components of the Dense Boundary Block (DBB) [31]

When a building appears similar to the background, it is easy to confuse the two classes. This becomes even clearer when the similarity occurs at the boundary of the building, which will cause the outline of the building to become fuzzy and irregular. This is the reason the creators of CT-UNet introduce the DBB. The job of the DBB is to enhance recognition capabilities and expand the distinction between the classes.

Figure 2.15 show the components of the DBB. The upper half of the figure is the DB, and the lower half of the figure is the BB.

The input of the DB is either the output from the last batch normalization layer at the current depth or the output of the SCAB at the current depth. The DB consists of two bottleneck layers, where each layer performs a BN, ReLU activation, and 1x1 convolution, followed by BN, ReLU activation, and 3x3 convolution. The 1x1 convolution is used to reduce the number of parameters, and the 3x3 convolution is used to extract the features. The input of the first bottleneck layer is the input of the DB. After the first bottleneck layer, the output from the layer is concatenated with the input and is used as input for the second bottleneck layer. Then after the second bottleneck layer, the input of the block and output of the first and second bottleneck layers are concatenated and used as the output of the DB.

The BB takes two inputs. Input1 is the output of the DB at the current depth, and input2 is the output of the DB/DBB at the previous depth. Since input1 is half the size of input2 and doesn't contain the same number of channels, it is first 1x1 convoluted before performing an up-convolution to match the dimensions. The high-level (Input1) and low-level (Input2) features are merged before they go through a final 1x1 convolution.

The BB returns two outputs. Output1 is used for the next DBB, and Output2 is used as input in the SCAB for the decoder side of the network.

2.5.2 Spatial Channel Attention Block

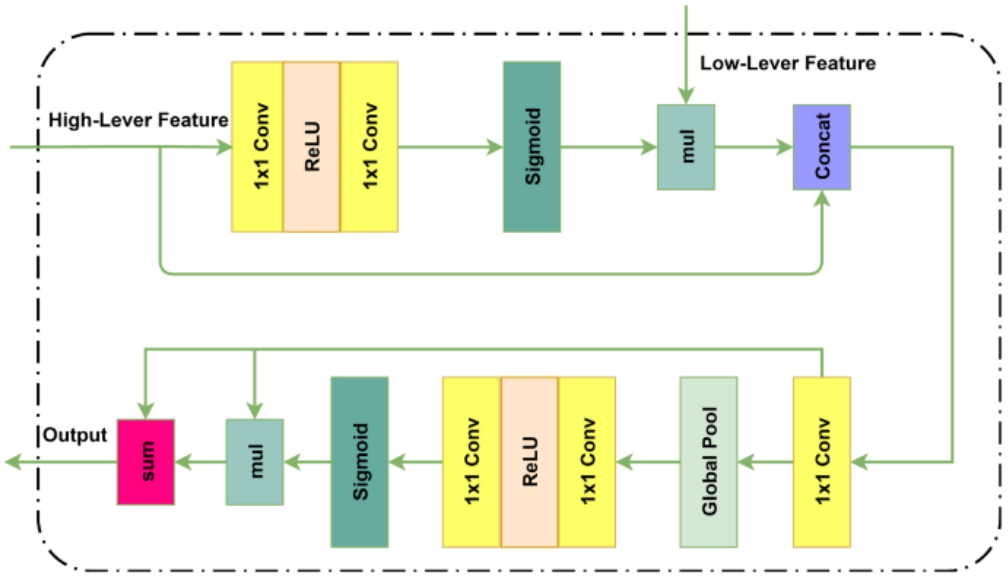


Figure 2.16: Components of the Spatial Channel Attention Block (SCAB) [31]

In an encoder-decoder network, the decoder uses simple up-scaling or up-convolution. These operations largely ignore the context information, causing intra-class inconsistency in the result. To handle this problem, the creators of CT-UNet introduce the SCAB. The job of the SCAB is to combine context space information and select more distinguishable features from space and channel. This is done by introducing low-level and high-level features on the decoder side of the network.

The components of the SCAB can be seen in figure 2.16. The upper half of the block is called the spatial attention mechanism(SAM), and the lower half of the block is called the channel attention mechanism(CAM).

The SAM uses semantic information about the high-level features to guide low-level feature selection. The high-level features are the output from the up-convolution from the previous decoder layer, and the low-level features are the output of the DB/DBB on the skip connection. The high-level features are first 1x1 convoluted to reduce the number of channels while keeping the feature map constant. Then a second 1x1 convolution occurs, which uses sigmoid as an activation function. This creates a classification probability map for each pixel normalized between 0 and 1. The probability map is multiplied with the low-level features to enhance the features of interest selectively. The output of this

multiplication is then concatenated with the high-level features before the CAM part of the block.

The CAM is designed to reassign weights based on the importance of individual channels. The output of the SAM is firstly 1x1 convoluted to restore the channels from the concatenation operation, followed by global average pooling. These average values are fed into a 1x1 convolutional layer, ReLu activation, and 1x1 convolution, followed by a sigmoid activation function. The sigmoid activation function returns a score map of the channels. The score map is multiplied by the output of the 1x1 convolution before the global average pooling. The result of this is then added together with the same 1x1 convolution output. By doing this, the channels with more discriminate features are selected.

2.5.3 Impact of the Different Blocks

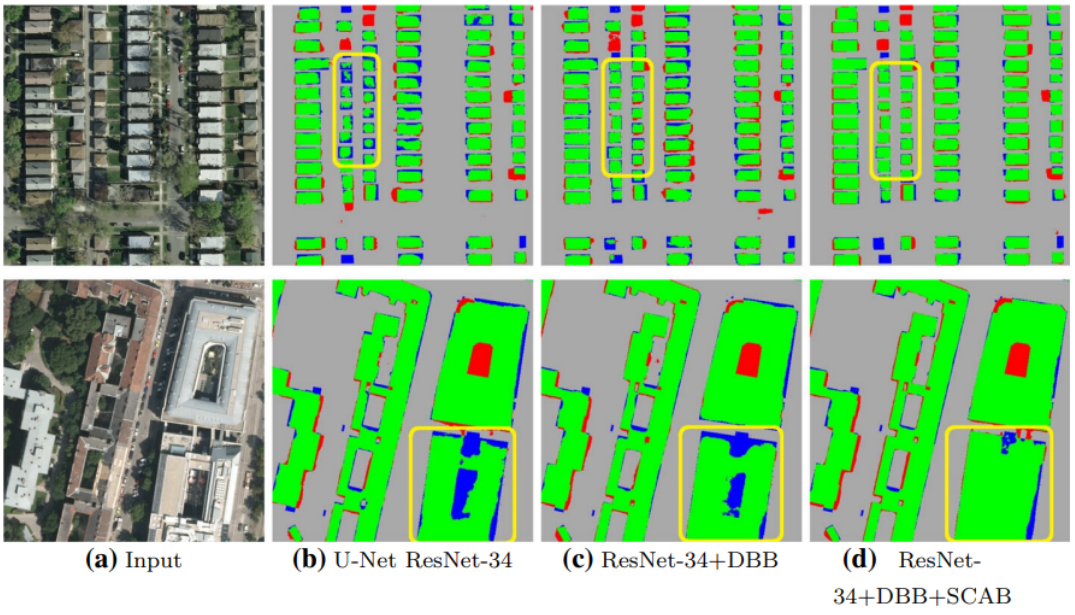


Figure 2.17: Impact of the different blocks in image segmentation. Green: true positive (TP) pixels. Gray: true negative (TN) pixels. Red: false positive (FP) pixels. Blue: false negative (FN) pixels [31].

Figure 2.17 shows the segmentation results by applying the different blocks. U-Net using ResNet-34 as the backbone is compared to CT-UNet using ResNet-34 as the backbone with DBB and DBB+SCAB. In the top row, it can be seen that U-Net mistakes some parts of the buildings as background, and in the bottom row, it can be seen that U-Net mistakes the middle of the building as background. In both rows, some of the predicted buildings have irregular edges. After the implementation of DBB, most of these wrong predictions are eliminated. SCAB pays more attention to the division of buildings and

helps fill holes in the middle of buildings that are predicted as background. This reduced the number of false negative pixels.

2.6 Transfer Learning and Backbones

Transfer learning is a method that lets us use a pre-trained model as the starting point for training a new model. These models are trained on image classification problems and replace the encoder side of the segmentation network. The pre-trained models are often referred to as backbones. Transfer learning reduces training time and increases the performance of the network. This is because the pre-trained model already knows how to classify objects and has learned general features such as edges and different shapes.

The backbones in TensorFlow are trained on ImageNet[32] using a 3-dimensional input. ImageNet is a large dataset containing over 14 million images divided into 21,841 classes. Task 1 of the thesis implements backbones with pre-trained weights. Since task 2 uses 4-dimensional images, using the pre-trained models is not possible. Because of this, the backbones are trained from scratch. The following backbones are used for the thesis.

2.6.1 ResNet50V2

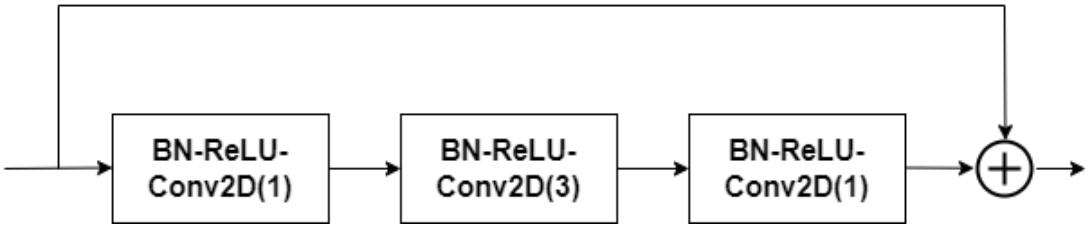


Figure 2.18: Illustration of a residual block

ResNet50V2 [33] is an improved version of the deep residual network (ResNet) architecture [34]. It is a 50-layer deep neural network that contains multiple residual blocks, as seen in figure 2.18. These residual blocks consist of a series of BN, ReLU, and convolutional layers, with a skip connection connected to the input and output of the block. The first and third convolutional layer performs a 1x1 convolution, while the second convolutional layer performs a 3x3 convolution. The output of the skip connection and the output of the last convolutional block is added together before another residual block.

Compared to most neural networks that directly try to learn the true output of a function, the residual block tries to learn the residual of a function. The residual is defined as the difference between the input and output of a function. It is represented by equation 2.16, where $R(x)$ is the residual, $H(x)$ is the predicted output, and x is the input.

$$R(x) = H(x) - x \tag{2.16}$$

This can be rewritten as shown in equation 2.17, where $R(x)$ is the predicted residual from the last convolutional block and x is the output of the skip connection.

$$H(x) = R(x) + x \tag{2.17}$$

This show that it is possible to predict the output of a function by adding the input of the residual block with the predicted residual. This way of setting up a neural network allows for deeper networks that can learn more complex features. It also addresses the vanishing gradient problem and leads to faster convergence.

2.6.2 DenseNet201

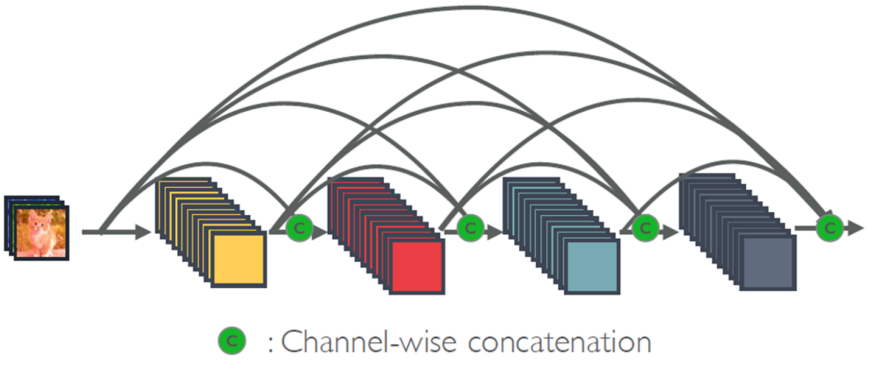


Figure 2.19: Illustration of a dense block [35]

DenseNet201 is a 201-layer deep neural network that consists of four dense blocks connected by transitional layers [36]. A dense block contains multiple sets of BN, ReLU, and 1x1 convolutional layers, followed by BN, ReLU, and 3x3 convolutional layers. The input for each of these sets of layers has an outbound skip connection that is concatenated with the output of all the following sets of layers, as seen in figure 2.19. This creates a densely connected network where all the layers have direct access to the features learned in the previous layers. This promotes the reuse of already learned features, which reduces the number of kernels and parameters needed to train the network. In the first dense block, DenseNet201 contains six sets of layers, followed by 12, 48, and 32 sets in the following dense blocks.

The input image is first fed into a 7x7 convolution, BN, and ReLU to detect large patterns followed by a 3x3 max pooling, both with a stride of 2. The output of the

pooling operation is then fed into the first dense block. A transitional layer is used after each of the first three dense blocks. This layer contains a 1x1 convolution followed by average pooling and is mainly used to reduce the spatial dimension of the feature maps. Following the last dense block, global average pooling is performed before classification.

2.6.3 EfficientNetB4

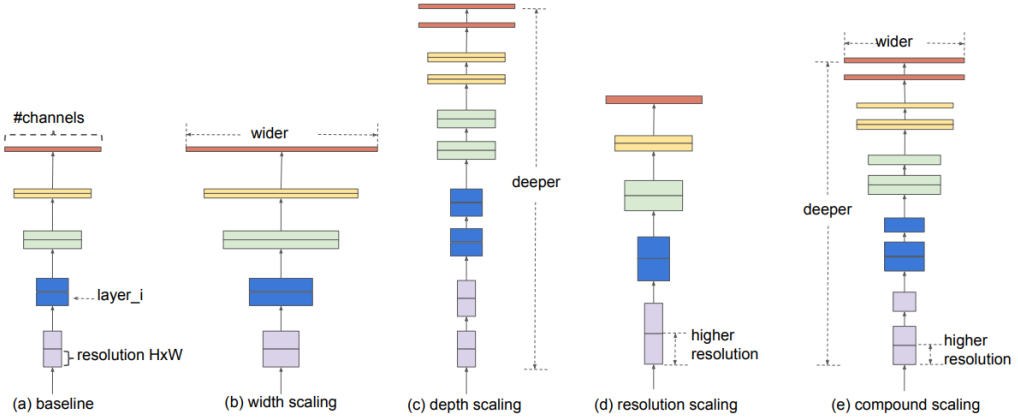


Figure 2.20: (a) shows a baseline network. (b-d) shows conventional scaling that only increases one dimension of a network, (e) shows compound scaling that uniformly scales all dimensions of a network with a fixed ratio [37].

EfficientNet is a family of neural network architectures that uses compound scaling to scale up the sizes of the networks. EfficientNetB4 is one of them. When scaling up a network, you have three choices, width scaling, depth scaling, and resolution scaling. Width scaling increases the number of kernels used in the convolutional layers. This leads to a network that tends to be able to capture more fine-grained features. Depth scaling is used to increase the depth of the network. This leads to a network that captures richer and more complex features. Resolution scaling is used to increase the resolution of the input image. This leads to a network that tends to be able to capture more fine-grained patterns. All these scaling methods result in better performance of a model. However, the performance gain diminishes for larger models. Compound scaling combines all these scaling methods to optimize the network while maintaining a balance between performance and computational efficiency. Figure 2.20 shows all mentioned scaling methods. The compound scaling method scales the network up from the baseline network, where an optimal combination of scaling has to follow equation 2.18. α, β and γ are the depth, width and resolution scaling coefficients. ϕ is a user-defined coefficient used to determine the number of available resources for the network. Increasing the ϕ value results in a larger and more computationally resource-dependent network.

$$\begin{aligned} &\text{depth:} && d = \alpha^{\phi} \\ &\text{width:} && w = \beta^{\phi} \\ &\text{resolution:} && r = \gamma^{\phi} \\ &\text{s.t.} && \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ & && \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \end{aligned} \tag{2.18}$$

The paper's authors found that the best value for α, β, and γ was 1.2, 1.1, and 1.15.

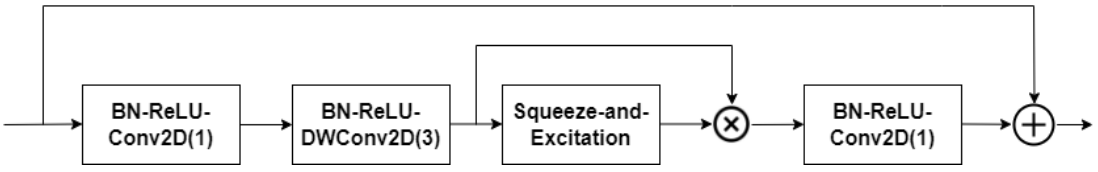


Figure 2.21: Illustration of an MBConv block

The architecture of EfficientNet is made up of multiple inverted residual blocks, or MBConv, as they are usually called. MBConv blocks are built similarly to the residual block shown in figure 2.18, where the difference is that the 3x3 convolution block is replaced by a depthwise 3x3 convolution block followed by a Squeeze-and-Excitation (SE) block, as seen in figure 2.21.

As mentioned in chapter 2.3.1, the kernel in normal convolution has the same depth as the input image. After the kernel has convolved the whole input, the results from each channel are added together to form a matrix of one dimension. This is done for every kernel in the block. The depth-wise convolution applies a different kernel to each channel of the input feature map instead. This means that if the input feature map has 64 channels, 64 different kernels with a depth of 1 will perform convolution in each channel.

The SE block consists of two different operations, squeeze and excitation. The squeeze operation squeezes the feature maps from the depth-wise convolution using global average pooling to generate channel-wise statistics. The excitation operation consists of two 1x1 convolutional layers, where the first one is followed by a ReLU activation, and a sigmoid activation follows the second one. This is done to produce weights representing each feature map's importance. These weights are then multiplied with the output from the depth-wise convolution to emphasize informative features and suppress less useful ones [38].

This operation is followed by a 1x1 convolution which combines the weighted features learned by the depth-wise convolution. Compared to regular convolution, this way of extracting features reduces the computational cost while maintaining high performance.

2.6.4 EfficientNetV2S

EfficientNetV2 is a family of neural network architecture designed to be more efficient and powerful than EfficientNet [39]. This is done by introducing a new scaling method and replacing some MBConv blocks with Fused-MBConv blocks. EfficientNetV2 is available in sizes from small (S) to large (L). The small version is used for this thesis.

The authors of the EfficientNetV2 paper claim that the compound scaling in EfficientNet is inefficient when the models become large enough (ϕ becomes large), as all parameters in the network are scaled the same. Compound scaling is replaced with a non-uniform scaling method. This scaling method gradually scales the network at later stages. The resolution scaling has also been maxed at 480x480 to reduce the resource impact on the GPU.

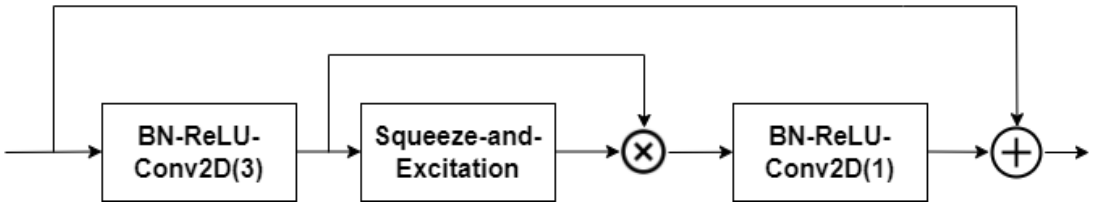


Figure 2.22: Illustration of a Fused-MBConv block

The authors also claim that depth-wise convolution is slow in the early stages of the network but effective in later stages. Because of this, the first half of MBConv blocks are replaced with Fused-MBConv. The architecture of Fused-MBConv can be seen in figure 2.22. The figure shows that the first 1x1 convolution and 3x3 depth-wise convolution is replaced with a standard 3x3 convolution. By doing this, the network trains faster with only a small parameter increase.

2.7 Data Augmentation

Data augmentation is a technique that creates modified copies of existing data to artificially increase the training set, as seen in figure 2.23.

Data augmentation is known to reduce overfitting and increase the performance of a model. It can be split into two categories, offline and online data augmentation. In offline data augmentation, the training set is augmented before the training starts. This augmented data is added to the original training set to create a larger set. Because of this, the same expanded training set will be used for each epoch. In online data augmentation, the training set is augmented during training. This results in a larger diversity in training

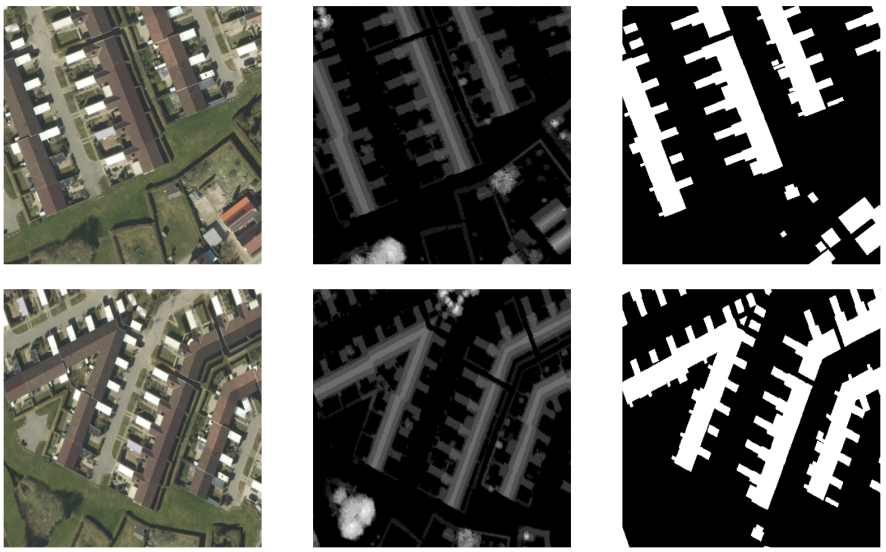


Figure 2.23: The top row shows the original data. The bottom row shows the data after augmentation.

data as the data augmentation will output unique samples for each epoch. The data augmentation methods used in this thesis combine the following techniques.

- `Rotation:` Rotates the data from -90 to 90 degrees. As rotation occurs, some pixels will be placed out of bounds. These pixels are removed from the image. Empty pixel slots in the image after rotation are filled by reflecting the adjacent pixels.
- `Width shift:` Shifts the image along the x-axis at a range of 0.7 to 1.3. Out-of-bound pixels are removed, and new pixels are filled by reflecting adjacent pixels.
- `Height shift:` Shifts the image along the y-axis at a range of 0.7 to 1.3. Out-of-bound pixels are removed, and new pixels are filled by reflecting adjacent pixels.
- `Shear:` Distorts the image along the x- or y-axis to create a perception of angles in the image. This is done by locking one axis while stretching the other at a specific angle.
- `Zoom:` Zooms in or out on the image at a range of 0.7 to 1.3. When zooming in, out-of-bound pixels are removed. When zooming out, new pixels are filled by reflecting adjacent pixels.
- `Horizontal flip:` Flips the image by reversing its columns of pixels.
- `Vertical flip:` Flips the image by reversing its rows of pixels.

2.8 Test-Time Augmentation

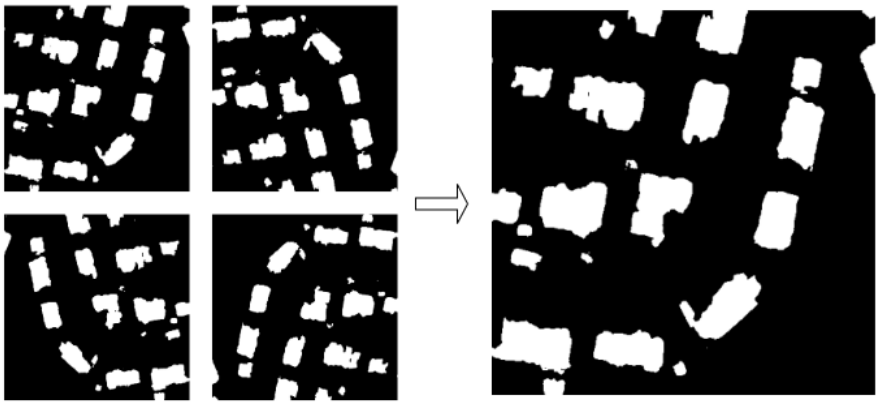


Figure 2.24: Test-Time Augmentation results performed on the original image, 90 degrees rotation, 180 degrees rotation, and 270 degrees rotation

Test-Time Augmentation (TTA) is a technique used when testing a model. The main idea of TTA is to generate multiple augmented versions of a testing image and make a prediction for each of them. All of the predictions are then combined to make a final prediction. For segmentation networks, this is done by reversing the augmentation of the predicted masks to their original position and taking the average of them, as seen in figure 2.24. This lets the network make a more confident prediction.

TTA can help the network reduce misclassification due to minor changes in the input. However, it increases the testing time and is more computationally heavy than just predicting a single image instance.

2.9 Model Ensemble

The prediction of a model is based on the patterns learned during training. Models with different architectures, hyperparameters, and training strategies will better recognize different patterns. An ensemble of models uses this as an advantage by combining the predictions of different models with a given weight for each of them. This is done so that the strength of one model will compensate for the weakness of another model.

It is shown that an ensemble of multiple models generally outperforms single models [40]. However, it requires more computational resources as multiple models must be trained, and multiple predictions on the same data must be made.

2.10 Metrics

Metrics are used to monitor the performance of a network during training and testing. Compared to loss functions, they do not have to be differentiable, as metrics don't affect the network's performance.

2.10.1 Intersection Over Union

Intersection over union (IoU), also known as the Jaccard index, is a common metric used for segmentation tasks. It measures the similarity between the predicted and ground truth masks by computing their overlap. Equation 2.19 shows the formula for IoU, where G is the ground truth and P is the prediction.

$$IoU = \frac{|G \cap P|}{|G \cup P|} = \frac{|G \cap P|}{|G| + |P| - |G \cap P|} \tag{2.19}$$

The IoU measure ranges from 0 to 1, where 0 means no overlap between the masks, and one means that the predicted mask corresponds to the ground truth mask.

2.10.2 Boundary Intersection Over Union

Boundary intersection over union (BIoU) is a metric focused on boundary quality in segmentation tasks [41]. It measures the similarity in the class boundaries between the ground truth mask and the predicted mask by computing the overlap between them. The predicted and ground truth boundaries are found using an edge detector on both masks before extracting the boundary of pixel width d. Equation 2.20 shows the formula for BIoU, where G is the ground truth, P is the prediction, and G_d and P_d are their corresponding boundary masks.

$$BIoU = \frac{|(G_d \cap G) \cap (P_d \cap P)|}{|(G_d \cap G) \cup (P_d \cap P)|} \tag{2.20}$$

Just like IoU, the BIoU measure ranges from 0 to 1.

2.10.3 Accuracy as a Metric for Segmentation

Accuracy is the most common metric for classification tasks. It is defined as the ratio of correct classifications compared to the total number of classifications, as seen in equation 2.21.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.21}$$

However, accuracy has its drawbacks when it comes to segmentation tasks. A class imbalance between the foreground and background is common in most segmentation tasks. If most pixels in an image contain the background class, a model that classifies all pixels as the background class will still achieve high accuracy, even if no foreground pixels are classified. Because of this, accuracy won't be used as a metric in this thesis.

Chapter 3

Dataset and Pre-Processing

This chapter contains the dataset used for this thesis and the pre-processing done on the dataset.

3.1 Dataset

The dataset used for the thesis is the MapAI dataset [42]. The dataset contains aerial images in RGB format, rasterized LiDAR data in float32 format, and ground truth images in binary format. The ground truth is split into two classes, where 1's represents buildings and 0's represent the background. All data samples have a resolution of 500x500 pixels.

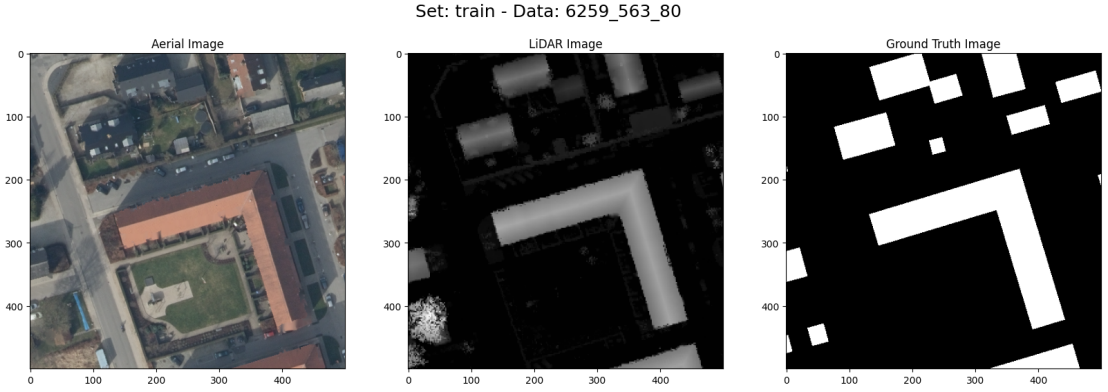


Figure 3.1: Data sample taken from the training split of the dataset

The dataset is divided into four splits. These are train, validation, task1_test, and task2_test. The train split contains 7500 data samples, and the validation split contains 1500 data samples. These two splits consist of several different locations in Denmark. Due to the area variability, it ensures that the training and validation split is diverse

with several types of environments and buildings. The task1_test split contains 1368 data samples, and the task2_test contains 978 data samples. The testing splits consist of seven different locations in Norway, comprising both rural and urban cities.

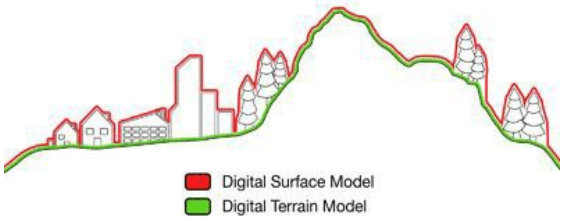


Figure 3.2: Illustration DSM and DTM [43]

The data is derived from a production setting. Because of this, some buildings will not be represented in the ground truth masks, and there will be ground truth masks that don't correspond to buildings. In addition to this, the ground truth masks in the test splits are generated using a Digital Terrain Model(DTM). As seen in figure 3.2, DTM approximates a continuous terrain surface. Due to this, the tops of the buildings in the test split will be skewed compared to the ground truths. This is not the case for the training and validation split, as the ground truths are generated using a Digital Surface Model(DSM), which captures the artificial features of the environment and doesn't skew the top of the buildings. This will result in lower performance on the test set for a model that predicts high on the validation set.

3.2 Data Pre-Processing

This section presents and discusses the pre-processing methods used on the dataset. Figure 3.3 shows the flowchart for this part of the approach.

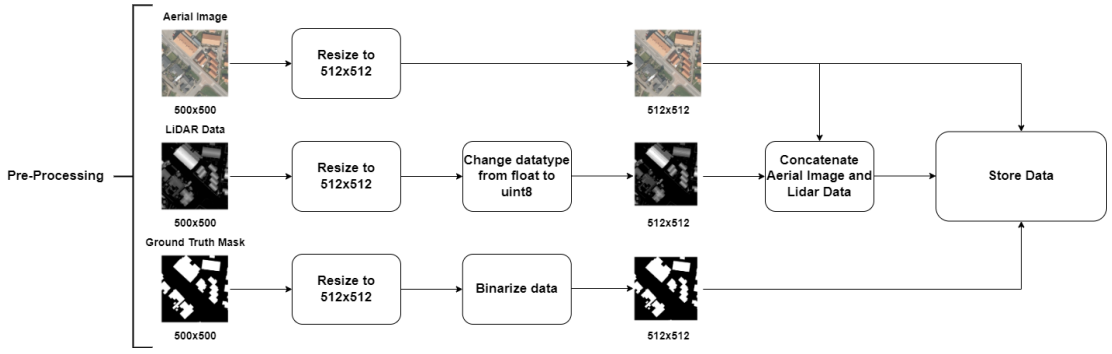


Figure 3.3: Overview of the pre-processing approach

3.2.1 Reshaping data

The encoder-decoder architecture consists of several layers. Each layer consists of either a max-pooling or transposed convolution layer, which reduces or increases the size of the image by a factor of the stride.

$$W_n = \left\lfloor \frac{W}{S} \right\rfloor \tag{3.1}$$

Equation 3.1 shows the new image size after an image with the height and width of W is pooled with the stride S. In the encoder-decoder architecture, the stride is 2. When using the original data size in the dataset, which is 500x500 pixels, the data's height and width after four max pooling blocks will be what is shown in equation 3.2.

$$500 \Rightarrow 250 \Rightarrow 125 \Rightarrow 62 \tag{3.2}$$

After this, when performing transposed convolution, the size of the data will be what is shown in equation 3.3

$$62 \Rightarrow 124 \Rightarrow 248 \Rightarrow 496 \tag{3.3}$$

As seen in the two equations, this causes a mismatch in data size between the encoder and decoder side of the network. This will make it impossible to concatenate data through skip connections. Because of this problem, the data is reshaped to 512x512 pixels.

3.2.2 Merging RGB and LiDAR data

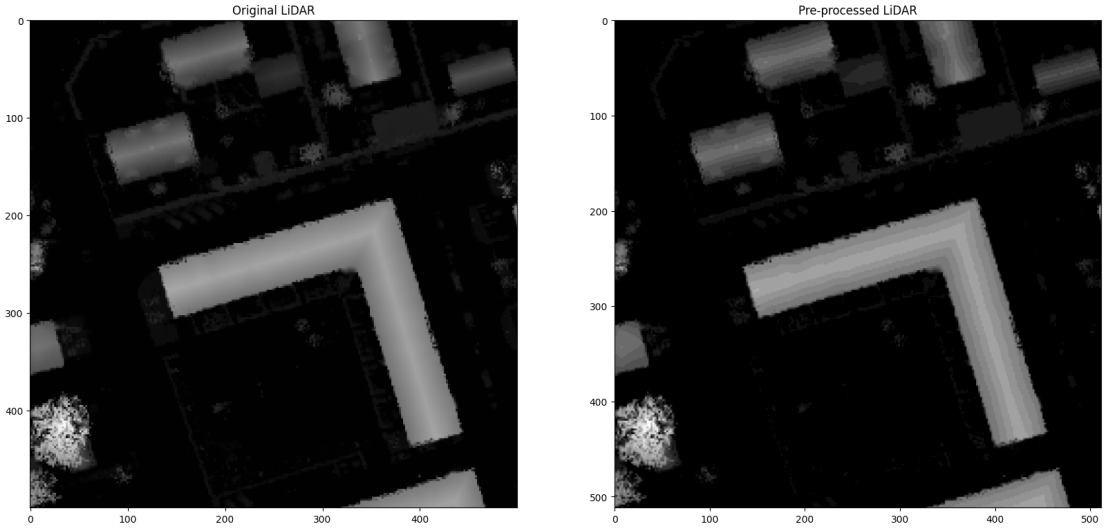


Figure 3.4: Comparison between original and pre-processed LiDAR data

Feeding neural networks built on the encoder-decoder architecture with multispectral images is possible. A multispectral image is an image that contains data from more than three spectral bands. Since task 2 of the thesis allows for using both RGB images and

LiDAR data, a subset of 4 channel data structures is created for the train, validation, and task2_test split. Since Tensorflow only allows for 8-bit data formats when an image has four channels to load batches from the directory, the LiDAR data format is changed from float32 to uint8.

As seen in figure 3.4, some information is lost during the pre-processing of the LiDAR data. Mainly the smoothness of the contours on the roofs of the buildings and data points close to the ground, such as some fences and cars. However, as the building's location, shape, and general height are the same, this should not affect the training result to a large degree. After the LiDAR data is pre-processed, it is concatenated with the 3-channel RGB image to form a 4-channel data structure.

Chapter 4

Approach

This chapter contains and discusses the proposed approach for the thesis. This includes the different architectures used for creating the models and the training methods. Figure 4.1 shows the flowchart of the training approach.

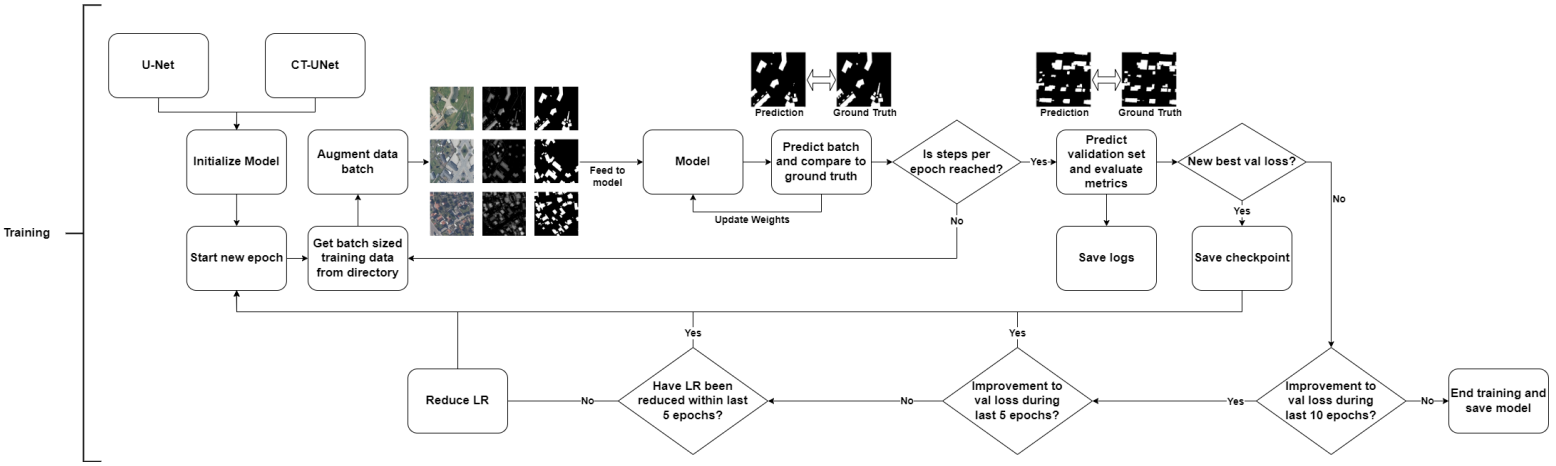


Figure 4.1: Overview of training approach

4.1 Model Architecture

This section presents and discusses the model architectures used for this thesis. The model architectures used are U-Net and CT-UNet. They were combined with the following backbones: ResNet50V2, DenseNet201, EfficientNetB4, and EfficientNetV2S.

4.1.1 Placement of BN layers

A highly discussed topic in the field is if the BN layer should be placed before or after the activation function. The authors of the original paper state that the BN should occur before the non-linearity activation function. However, when using a ReLU activation function after BN, all negative values outputted by the BN layer will be changed to zero. After some testing with BN before and after the activation function, as seen in appendix B, it is concluded that BN after the activation function performs better for the proposed approach. This is the case for both U-Net and CT-UNet.

4.1.2 U-Net

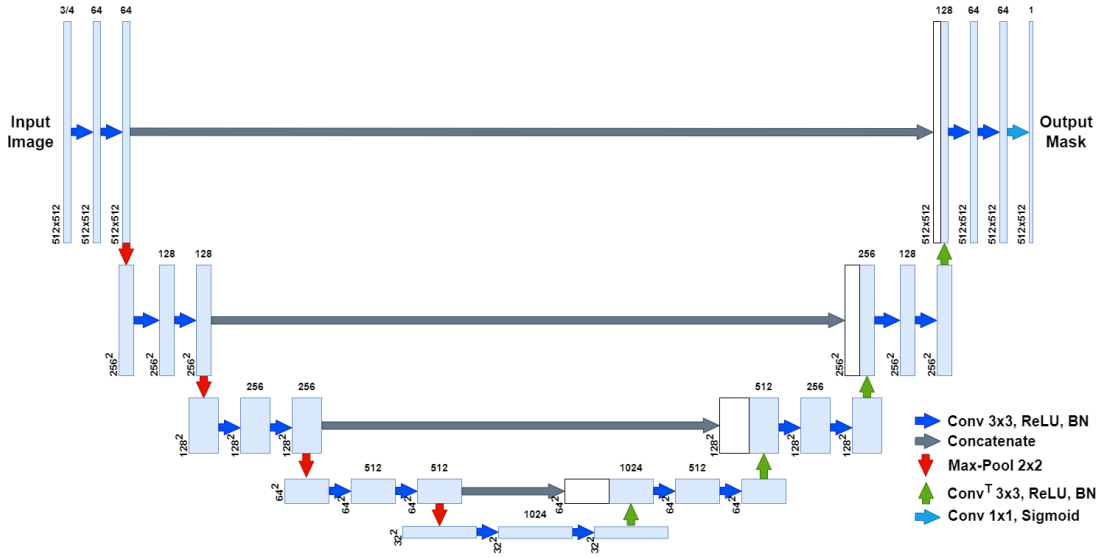


Figure 4.2: Illustration of U-Net architecture used in the approach

A modified version of the original U-Net architecture explained in chapter 2.4 is used for this thesis, as seen in figure 4.2. The input layer has a spatial dimension of 512x512 and a channel size of 3 or 4, depending on which task is performed.

The encoder side of the network performs two 3x3 convolutions using ReLU as an activation function followed by BN for each network depth. Zero-padding is added to the feature maps for each convolution so that the output shape is the same as the input shape. This is because we want the same spatial dimension for the network's input and output. The kernels are initialized using a Gaussian normal distribution to reduce the probability of encountering the dying ReLU problem. The first depth of the network uses 64 kernels for each convolution. This is doubled for each depth. After the two sets of operations, 2x2 max-pooling is performed to reduce the spatial dimensions of the feature maps.

For the decoder side, as opposed to the original U-Net paper, 3x3 transposed convolution is used instead of 2x2 transposed convolution. This deviation is because 2x2 transposed convolutions are known to create a checkerboard pattern in the upscaled feature map, which can introduce artificial edges and reduce the performance of a network. The transposed convolutions are followed by ReLU activation and BN. This is so that the upscaled image and concatenated feature map from the encoder side of the network have the same range and representation of values, as each depth of the encoder side of the network ends with BN. After the feature maps have been concatenated, two sets of 3x3 convolutions are performed, just like the encoder side of the network.

The network performs a final convolution after the last set of 3x3 convolutions. This convolution uses a single kernel of size 1x1 followed by a sigmoid activation function.

When implementing the U-Net architecture with one of the four backbones, the backbone replaces the encoder side of the network. The last feature map of each spatial dimension from the backbone is used for the concatenation with the decoder feature map.

4.1.3 CT-UNet

A modified version of the original CT-UNet architecture explained in chapter 2.5 is used for this thesis.

The encoder side of the network consists of one of the four backbones explained in chapter 2.6. The input size is 512x512 with 3 or 4 channels depending on the task performed. At the bottom of the network, the spatial dimension of the feature maps is 16x16, with the number of channels the backbone outputs. The first change of the modified CT-UNet network is that the GAP block at the bottom of the network is removed. This is because the spatial dimension of the feature map will be reduced to 1x1 and cause a mismatch of dimensions in the decoder part of the network. It was attempted to replace the GAP with a SE block, as explained in chapter 2.6, but the network performed better without any GAP or SE block.

The ReLU activation function and BN block change places in the network's decoder part. The bottom layer uses 512 kernels for the convolution operations. This is then halved for each following layer until the top of the network is reached, which uses 16 kernels for each convolution operation. The last de-convolution block of the original CT-UNet architecture is removed as it doesn't make sense to upscale the feature map from 512x512 to 1024x1024. The convolution and transposed convolution operations in the decoder part of the network use a kernel size of 3x3.

In the original CT-UNet architecture, the DBB uses the previous stage DB/DBB block output as its second input. This input contains half the number of channels as the current block. Because of this, it is fed into a 1x1 convolution so that the number of channels matches the DBB block. At the end of the DBB block, the features are pooled using max-pooling to match the dimensions of the SCAB and the next DBB block. The SCAB block remains mostly the same as the author's architecture, where the only difference is that the SCAB block at the top of the network performs a 1x1 convolution on the input from the DB block. This is because their number of channels needs to match.

All convolution blocks using ReLU activation are initialized with He normal, and the blocks using sigmoid activation are initialized using Glorot normal.

4.1.4 Hyperparameters

The choice of hyperparameters was decided on by training multiple models with different hyperparameters and choosing the hyperparameter that gave the best results on the validation set. A more in-depth explanation of these tests can be seen in appendix B.1 and appendix B.2.

The best initial learning rate for U-Net is 0.0001, with ADAM as the optimizer. The best-performing loss function is Dice loss.

For CT-UNet, the best initial learning rate is 0.00005, with ADAM as the optimizer. The best-performing loss function is Dice loss.

Due to the memory limitations of the GPU, the batch size was set so that the training progress used all available memory, which was 32GB. More of this is explained in chapter 4.1.5.

4.1.5 Model Overview

The amount of memory a model uses while training is determined by the complexity of the model, the number of parameters of the model, and the batch size used while training.

Networks containing many layers and/or parameters are heavy on the GPU as it requires memory to store the weights of each layer in the model. A high batch size also requires more memory as the model has to store the gradients for each sample in the batch. Due to this, the models with the most parameters and layers use smaller batch sizes than those with fewer parameters and layers. This can be seen in the following table, which contains all the models used for this thesis.

Architecture	Backbone	Batch Size	Parameters
U-Net	None	6	34,540,737
U-Net	EfficientNetB4	12	18,253,672
U-Net	EfficientNetV2S	12	19,069,561
U-Net	ResNet50V2	12	17,541,441
U-Net	DenseNet201	12	24,746,881
CT-UNet	EfficientNetB4	10	17,831,856
CT-UNet	EfficientNetV2S	12	19,413,761
CT-UNet	ResNet50V2	12	17,803,401
CT-UNet	DenseNet201	10	24,831,177

Table 4.1: Models used for the thesis

4.2 Training

This section presents the methods used while training the models.

4.2.1 Data Generator and Augmentation

Training large models with thousands of training and validation images is expensive and requires a lot of Video RAM (VRAM). When using a large dataset, most of this VRAM will be used for storing the images in arrays. Keras data generators are used for training to avoid this unnecessary waste of VRAM. Instead of loading the whole dataset into an array, the data generator only loads one batch at a time into memory. This is done with the `flow_from_directory` function.

Online data augmentation is performed on each batch when loading batches from the directory, as explained in chapter 2.7.

4.2.2 Callbacks

In Keras, a callback is an object that performs actions during various stages of training. All callbacks used in this thesis are executed after an epoch.

ModelCheckpoint

Training a model for this thesis takes around 10 hours on the setup used. If the training unexpectedly stops, the progress is lost. Because of this, the `ModelCheckpoint` callback

function is used, which saves a checkpoint of the model for the best-performing epoch. This is done by monitoring the validation loss at the end of the epoch. If the new validation loss is lower than the previous one achieved during training, the resulting model and its weights are saved. In some cases, the final checkpoint also performs better than the last epoch of the training, another reason for using this callback function.

EarlyStopping

Since multiple models are used for this thesis, it is hard to set a specific number of training epochs to train the models. Training for too many epochs will lead to overfitting the training set, while training for too few epochs may lead to underfitting the model and not getting the best results possible. Because of this, the `EarlyStopping` callback is used. This callback function monitors the validation loss at the end of the epoch. If the validation loss has not decreased for the last ten epochs, the training will stop, and the model will be saved.

ReduceLROnPlateau

As the training nears its convergence, it is beneficial to reduce the learning rate. This is so the model can more easily reach the global minima for the loss function. The `ReduceLROnPlateau` callback function is used to do this. This callback function monitors the validation loss at the end of the epoch. If the validation loss has not decreased for the last five epochs and the learning rate has not decreased over the previous five epochs, the learning rate is decreased by a factor of 0.1.

CSVLogger

The last callback function used is `CSVLogger`. The callback function streams each epoch result to a CSV file. This file contains the epoch number, learning rate, training accuracy, training IoU, training loss, validation accuracy, validation IoU, and validation loss. These log files are used for the training graphs in this thesis.

Chapter 5

Experimental Evaluation

This chapter presents and discusses the experimental part of the thesis. This includes the experimental setup, results of the different experiments, and analysis of the experiments.

5.1 Experimental Setup

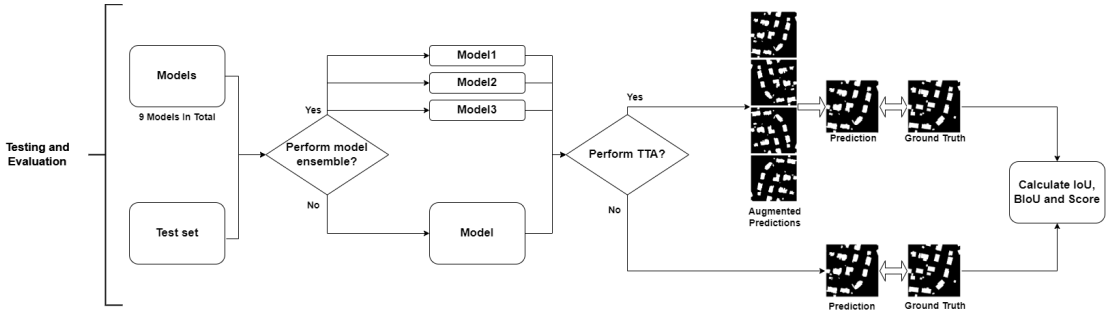


Figure 5.1: Overview of the experimental setup

Figure 5.1 shows the flowchart for the experimental setup. The figure shows that the experimental setup is split into four parts. These are evaluation, single model prediction, ensemble model predictions, and TTA.

5.1.1 Evaluation

The models are evaluated using IoU and BIoU, as explained in chapter 2.10, as well as the average of these two metrics as seen in equation 5.1.

$$Score = \frac{IoU + BIoU}{2} \tag{5.1}$$

These metrics are used as they are the official metrics used in the MapAI competition. The functions for calculating the evaluation metrics were copied from the MapAI competition GitHub repository [44].

The IoU and BIoU are calculated using all the predicted masks for the test dataset and the ground truth masks. The IoU and BIoU represent the mean IoU and mean BIoU values of all the images in the test dataset.

5.1.2 Single Model Prediction

The single-model prediction takes a single trained model and predict's once on the whole testing dataset. After receiving the predicted masks, a threshold of 0.5 is set. If the predicted pixels in the mask have a value less than or equal to the threshold, the pixel values are changed to 0 (background). If they are larger than the threshold, the values are changed to 1 (building). After this, the predicted masks are evaluated using the ground truth data.

5.1.3 Ensemble Model Prediction

An ensemble of models consists of 3 different models. All of the models predict once on the whole dataset. After receiving the predicted masks, a threshold of 0.5 is set, just like the single model prediction. When this is done, the model ensemble can start. A list of float values ranging from 0 to 1, with an interval of 0.1, is initialized. This list represents the weights a model can have in the ensemble. A triple for loop is then started, where all the loops iterate through the list. These loops represent the weight of the three different models. If the sum of the weight equals one, the predictions of the three different models are multiplied by the weights and added together. After this, the pixel value of each predicted test image is changed to 0 if the pixel value is less than or equal to the threshold and changed to 1 if the pixel value is greater than the threshold. The predicted mask is then evaluated with the ground truth data. The weights and metrics are saved if the score is better than any previously seen score in the ensemble.

The function returns the best combination of weights and the IoU, BIoU, and score for the ensembled models using these weights.

5.1.4 TTA

The TTA is performed by predicting the original, horizontally, vertically, and horizontally vertical flipped versions of the image. The predicted masks are then converted back

to their original position before the average of the masks is calculated. This mask of average pixel values represents the TTA-predicted mask. Instead of a threshold of 0.5, a threshold of 0.3 is used. Every pixel value less than or equal to 0.3 is changed to 0, and every pixel value above 0.3 is changed to 1. Using a lower threshold, the model is more likely to predict that the pixel is part of the foreground if it appears in at least one of the predicted masks instead of all.

5.2 Experimental Results

This section presents the results of the experiments performed. All experiments have been performed on task 1 and task 2 of the thesis.

The following experiments have been performed.

- `Single-model experiments:` Done to determine which model performs the best.
- `Single-model experiments using TTA:` Done to determine if TTA improves the predictions of the different models.
- `Model ensemble:` Done to determine if an ensemble of models performs better than a single model prediction.
- `Model ensemble using TTA:` Done to determine if TTA improves the prediction on ensembled models.

5.2.1 Task 1 Experiments

This subsection contains the experiments done on task 1 of the thesis.

Training and Validation Results

All models shown in table 4.1 were trained on RGB aerial images for this task. The training progress for the models are shown in figure 5.2.

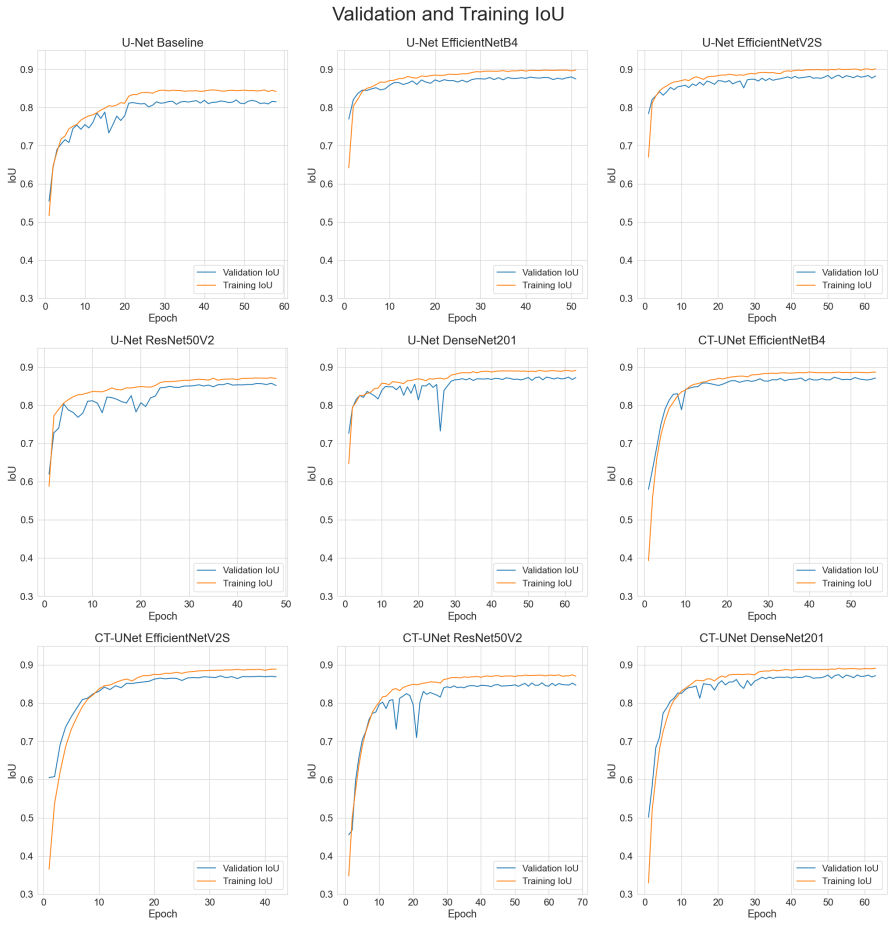


Figure 5.2: Validation and Training IoU during training

The figure shows that the training is stable for almost every model except U-Net DenseNet201 and CT-UNet ResNet50V2, where we see sudden large drops in validation IoU during training. However, this only lasts three epochs at most before the validation IoU reaches a new high. The initial epochs for CT-UNet architecture models are a little slower than those using U-Net. This is likely due to the lower learning rate for the CT-UNet models.

Table 5.1 shows the training and validation IoU after the final epoch. The table shows that U-Net EfficientNetV2S performs best for training and validation IoU, resulting in 90.1% and 88.22% IoU, respectively.

Model	Epoch	Train IoU	Validation IoU
U-Net Baseline	57	0.842	0.8146
U-Net EfficientNetB4	51	0.8978	0.8745
U-Net EfficientNetV2S	63	**0.901**	**0.8822**
U-Net ResNet50V2	48	0.8701	0.8516
U-Net DenseNet201	63	0.8911	0.8724
CT-UNet EfficientNetB4	56	0.8868	0.8713
CT-UNet EfficientNetV2S	42	0.889	0.8691
CT-UNet ResNet50V2	68	0.8706	0.8469
CT-UNet DenseNet201	63	0.8914	0.872

Table 5.1: Training and Validation IoU after the last epoch

On average, between the models, there is a difference of 2.06% in validation and training IoU. This indicates that a bit of overfitting occurs.

Single Model Results

Table 5.2 shows the results achieved for each model on the test dataset, with and without using TTA.

	Single Prediction			TTA		
Model	**IoU**	**BIoU**	**Total Score**	**IoU**	**BIoU**	**Total Score**
U-Net Baseline	0.7471	0.5690	0.6581	0.7529	0.5734	0.6632
U-Net EfficientNetB4	0.7836	0.6132	0.6984	0.7898	0.6183	0.7041
U-Net EfficientNetV2S	0.7675	0.6018	0.6847	0.7763	0.6100	0.6931
U-Net ResNet50V2	0.7617	0.5789	0.6703	0.7677	0.5829	0.6753
U-Net DenseNet201	0.7632	0.5990	0.6811	0.7685	0.6045	0.6865
CT-UNet EfficientNetB4	**0.7865**	**0.6148**	**0.7007**	**0.7914**	0.6189	**0.7052**
CT-UNet EfficientNetV2S	0.7824	0.6113	0.6968	0.7904	**0.6199**	0.7052
CT-UNet ResNet50V2	0.7609	0.5952	0.6781	0.7690	0.6022	0.6856
CT-UNet DenseNet201	0.7612	0.5977	0.6795	0.7680	0.6045	0.6863

Table 5.2: Results from single model predictions with and without TTA

The table shows that CT-UNet EfficientNetB4 performs best for single-image prediction, giving an IoU of 78.65%, BIoU of 61.48%, and a score of 70.07%. This is also the case for the TTA prediction, where we see an IoU of 79.14%, BIoU of 61.89%, and a score of 70.52%. Across all models, on average, TTA improves the IoU and BIoU by 0.67% and 0.6%, respectively.

Weighted Model Ensemble Results

Due to time limitations, the weighted model ensemble was performed on the six best models with respect to validation IoU. Table 5.3 shows the five best ensembles achieved using these models.

Model 1	Model 2	Weight	IoU	BIoU	Score
U-Net DenseNet201	CT-UNet EfficientNetB4	0.5, 0.5	**0,7954**	**0,6219**	**0,7086**
U-Net DenseNet201	CT-UNet EfficientNetV2S	0.5, 0.5	0,7938	0,6217	0,7078
U-Net EfficientNetV2S	CT-UNet EfficientNetB4	0.5, 0.5	0,7931	0,6210	0,7071
CT-UNet EfficientNetB4	CT-UNet EfficientNetV2S	0.5, 0.5	0,7937	0,6202	0,7069
CT-UNet EfficientNetB4	CT-UNet DenseNet201	0.5, 0.5	0,7930	0,6208	0,7069

Table 5.3: Top 5 model ensembles

The table shows that the best ensembles were achieved using two models with equal weight. The best ensemble was a combination of U-Net DenseNet201 and CT-UNet EfficientNetB4, which achieved an IoU of 79.54%, BIoU of 62.19%, and a score of 70.86%. It is worth mentioning that all top 5 ensembles were achieved using two models with equal weights.

Table 5.4 shows the top 5 model ensembles using TTA.

Model 1	Model 2	Model 3	Weight	IoU	BIoU	Score
U-Net DenseNet201	CT-UNet EfficientNetB4	CT-UNet EfficientNetV2S	0.4, 0.3, 0.3	**0,8011**	**0,6295**	**0,7153**
U-Net EfficientNetB4	U-Net DenseNet201	CT-UNet EfficientNetV2S	0.2, 0.4, 0.4	0,8001	0,6292	0,7146
U-Net EfficientNetB4	U-Net DenseNet201	CT-UNet EfficientNetB4	0.2, 0.4, 0.4	0,8000	0,6275	0,7138
U-Net EfficientNetV2S	U-Net DenseNet201	CT-UNet EfficientNetB4	0.3, 0.4, 0.3	0,7995	0,6280	0,7137
U-Net EfficientNetV2S	U-Net DenseNet201	CT-UNet EfficientNetV2S	0.2, 0.4, 0.4	0,7988	0,6284	0,7136

Table 5.4: Top 5 model ensembles using TTA

The best-performing model ensemble using TTA contains U-Net DenseNet201, CT-UNet EfficientNetB4, and CT-UNet EfficientNetV2S, with weights of 0.4, 0.3, and 0.3, respectively. This ensemble of models achieved an IoU of 80.11%, BIoU of 62.95%, and a score of 71.53%. This ensemble of models is used as the proposed solution for task 1. Across the top 5 ensembles, on average, TTA improves the IoU and BIoU by 0.61% and 0.74%, respectively. It is worth mentioning that U-Net DenseNet201 appears in all the top 5 ensembles using TTA, which means it catches features that the other models don't.

5.2.2 Task 2 Experiments

This subsection contains the experiments done on task 2 of the thesis.

Training and Validation Results

All models shown in table 4.1 were trained on RGB aerial images concatenated with LiDAR data for this task. The training progress for the models is shown in figure 5.3.

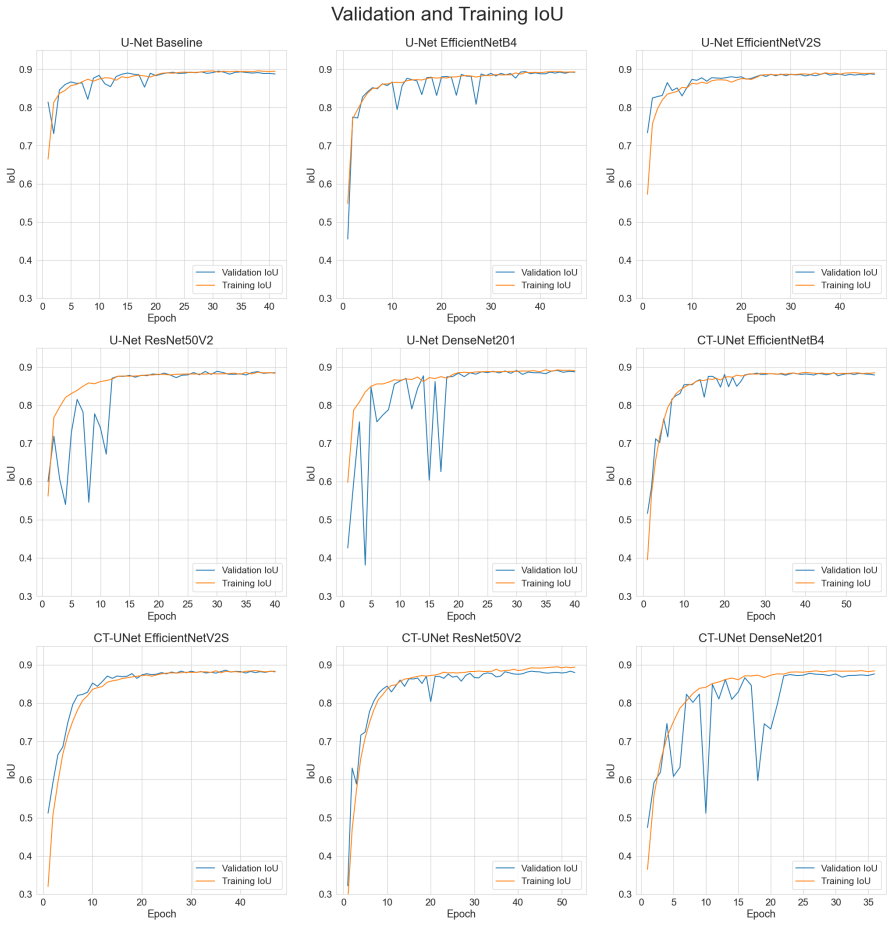


Figure 5.3: Validation and Training IoU during training

The figure shows that the training progress is very unstable for U-Net ReseNet50V2, U-Net DenseNet201, and CT-UNet DenseNet201, as the validation IoU suddenly drops drastically during training. However, the validation IoU manages to catch up to the training IoU at the end of the training. Like task 1, the CT-UNet models use a little more time before reaching the training IoU plateau due to the lower learning rate.

Table 5.5 shows the training and validation IoU after the final epoch. The table shows that U-Net Baseline performs best on the training dataset, resulting in an IoU of 89.46%.

U-Net EfficientNetB4 performs best on the validation dataset, resulting in an IoU of 89.18%.

Model	Epoch	Train IoU	Validation IoU
U-Net Baseline	41	**0.8946**	0.8845
U-Net EfficientNetB4	47	0.893	**0.8918**
U-Net EfficientNetV2S	47	0.8897	0.8866
U-Net ResNet50V2	40	0.8837	0.8853
U-Net DenseNet201	40	0.8903	0.8877
CT-UNet EfficientNetB4	57	0.8851	0.8792
CT-UNet EfficientNetV2S	47	0.8838	0.8821
CT-UNet ResNet50V2	53	0.8944	0.8799
CT-UNet DenseNet201	36	0.8845	0.8765

Table 5.5: Training and Validation IoU after the last epoch

On average, between the models, there is a difference of 0.51% in validation and training IoU. This indicates that almost no overfitting occurs.

Single Model Results

Table 5.6 shows the results achieved for each model on the test dataset, with and without using TTA.

	Single Prediction			TTA		
Model	**IoU**	**BIoU**	**Total Score**	**IoU**	**BIoU**	**Total Score**
U-Net Baseline	0.8836	0.7824	0.8330	0.8873	0.7891	0.8382
U-Net EfficientNetB4	0.8804	0.7720	0.8262	0.8847	0.7788	0.8317
U-Net EfficientNetV2S	0.8766	0.7798	0.8282	0.8799	0.7838	0.8319
U-Net ResNet50V2	0.8875	0.7859	0.8367	0.8897	0.7903	0.8400
U-Net DenseNet201	**0.8909**	**0.7907**	**0.8408**	**0.8930**	**0.7939**	**0.8435**
CT-UNet EfficientNetB4	0.8688	0.7661	0.8174	0.8721	0.7705	0.8213
CT-UNet EfficientNetV2S	0.8794	0.7783	0.8288	0.8820	0.7823	0.8322
CT-UNet ResNet50V2	0.8788	0.7775	0.8281	0.8825	0.7823	0.8324
CT-UNet DenseNet201	0.8751	0.7704	0.8227	0.8779	0.7737	0.8258

Table 5.6: Results from single model predictions with and without TTA

The table shows that U-Net DenseNet201 performs best for single-image prediction, giving an IoU of 89.09%, BIoU of 79.07%, and a score of 84.08%. This is also the case for TTA prediction, where we see an IoU of 89.30%, BIoU of 79.39%, and a score of 84.35%. Across all models, on average, TTA improves the IoU and BIoU by 0.31% and 0.46%, respectively.

Weighted Model Ensemble Results

Due to time limitations, the weighted model ensemble was performed on the six best models with respect to validation IoU. Table 5.7 shows the five best ensembles achieved using these models.

Model 1	Model 2	Model 3	Weight	IoU	BIoU	Score
U-Net Baseline	U-Net DenseNet201	-	0.5, 0.5	**0,8954**	0,7963	**0,8459**
U-Net EfficientNetV2S	U-Net ResNet50V2	U-Net DenseNet201	0.2, 0.4, 0.4	0,8921	**0,7965**	0,8443
U-Net Baseline	U-Net ResNet50V2	-	0.5, 0.5	0,8938	0,7941	0,8440
U-Net EfficientNetV2S	U-Net DenseNet201	-	0.5, 0.5	0,8934	0,7944	0,8439
U-Net EfficientNetB4	U-Net ResNet50V2	U-Net DenseNet201	0.1, 0.4, 0.5	0,8931	0,7938	0,8434

Table 5.7: Top 5 model ensembles

The table shows that the best ensembles were achieved using two models with equal weight. The best ensemble was a combination of U-Net Baseline and U-Net DenseNet201 which achieved an IoU of 89.54%, BIoU of 79.63%, and a score of 84.59%.

Table 5.8 shows the top 5 model ensembles using TTA.

Model 1	Model 2	Model 3	Weight	IoU	BIoU	Score
U-Net Baseline	U-Net DenseNet201	CT-UNet EfficientNetV2S	0.5, 0.3, 0.2	**0,8964**	**0,8009**	**0,8486**
U-Net Baseline	U-Net EfficientNetB4	U-Net DenseNet201	0.6, 0.1, 0.3	0,8962	0,8009	0,8486
U-Net Baseline	U-Net EfficientNetV2S	U-Net DenseNet201	0.5, 0.2, 0.3	0,8961	0,8010	0,8485
U-Net Baseline	U-Net ResNet50V2	U-Net DenseNet201	0.6, 0.1, 0.3	0,8962	0,8008	0,8485
U-Net Baseline	U-Net EfficientNetV2S	U-Net ResNet50V2	0.5, 0.2, 0.3	0,8954	0,7999	0,8476

Table 5.8: Top 5 model ensembles using TTA

The best-performing model ensemble achieved using TTA contains U-Net Baseline, U-Net DenseNet201, and CT-UNet EfficientNetV2S, with weights of 0.5, 0.3, and 0.2, respectively. This ensemble of models achieved an IoU of 89.64%, BIoU of 80.09%, and a score of 84.86%. This ensemble of models is used as the proposed solution for task 2. Across the top five ensembles, on average, TTA improves the IoU and BIoU by 0.25% and 0.57%, respectively. It is worth mentioning that U-Net Baseline dominates in all the top 5 ensembles using TTA.

5.3 Analysis

The experiments show that ensembles of models perform better than single model prediction for both tasks. This is because the ensemble reduces the variance of the predictions by using different architectures. As models have different strengths and weaknesses, the strength of one model is used to compensate for the weakness of another. The use of multiple models also captures a wider range of features, as all the models have learned different features based on the backbone of the model.

The experiments also show that TTA predictions perform marginally better than single-instance predictions. This is because TTA reduces the uncertainty for the predictions, captures a diverse representation of the input image, and is better at localizing edges between the background and foreground. By implementing both of these methods, the predictions are more robust and accurate.

Figure 5.4 and figure 5.5 show the effect of the different prediction methods on task 1 and task 2 of the test dataset, respectively. These images are predicted using the best performing models and ensembles from chapter 5.2.1 and 5.2.2.

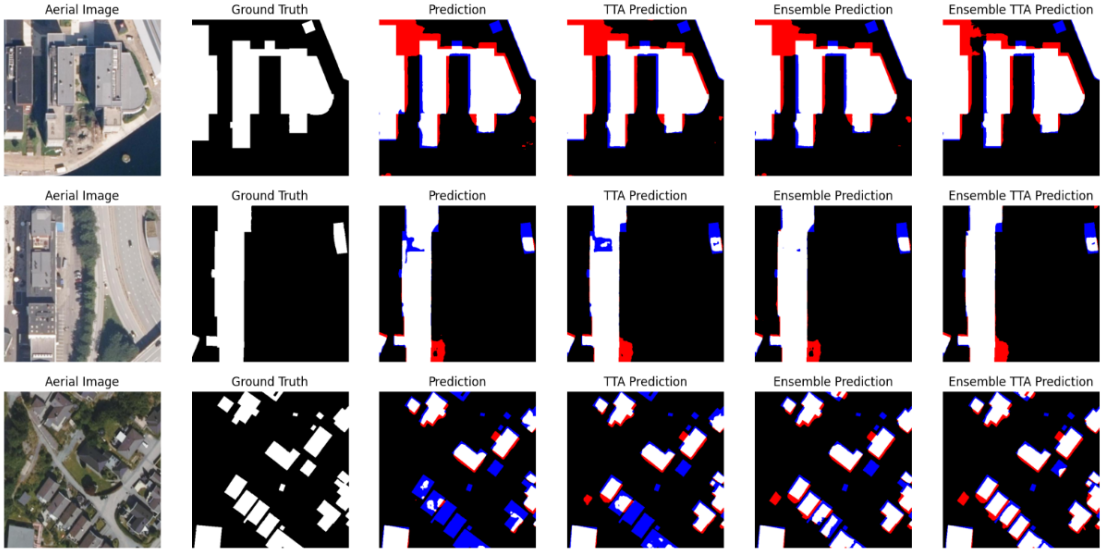


Figure 5.4: Predicted masks from task 1 using the test set. White: True Positives, Black: True Negatives, Red: False Positives, Blue: False Negatives.

As seen in figure 5.4, the prediction in the top row shows that the model thinks the left and right buildings are connected. There are also some specs of false positive pixels around in the predicted mask. When predicting with TTA or ensemble, most of these specs are removed. The ensemble using TTA manages to tell the two buildings apart and remove all random specs of false positives. In the middle row, it can be seen that the hole of false negatives in the main building is filled as we progress using more advanced prediction methods. The single model prediction in the bottom row has problems classifying the bottom row of buildings as foreground. TTA helps the model fill in two of the buildings. When deploying the ensemble, we see that most of the bottom row of buildings are correctly classified with some irregular edges. These edges are then straightened when using TTA.

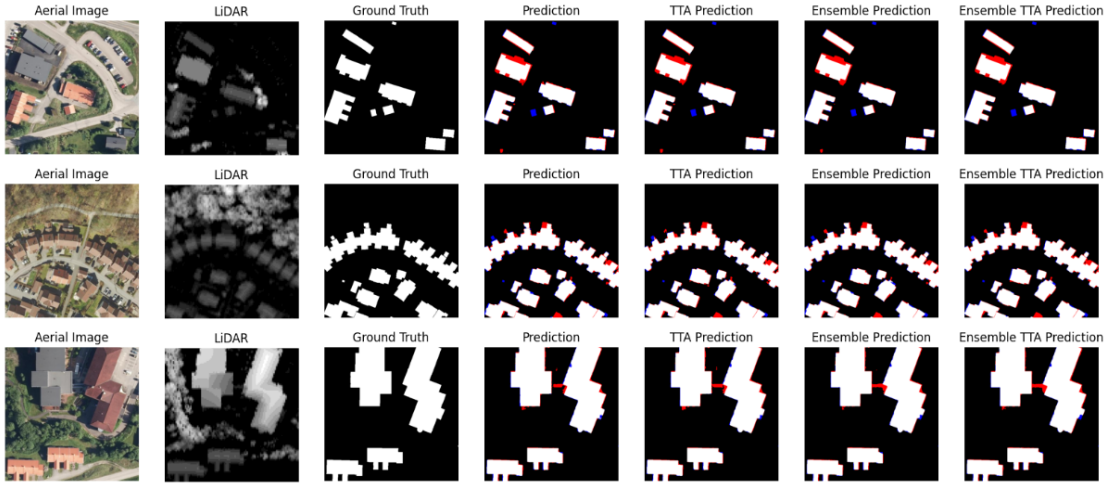


Figure 5.5: Predicted masks from task 2 using the test set. White: True Positives, Black: True Negatives, Red: False Positives, Blue: False Negatives.

Looking at figure 5.5, we see only minor changes in the predicted masks compared to task 1. The shape of all predicted buildings is the same, where the only difference is that tiny specs of false positives are removed as we move towards the more advanced methods. This is mainly due to the models weighing the LiDAR data more than the RGB data, as we see that the predictions correlate to the shape and location of the buildings in the LiDAR data.

As seen in table 5.1 and 5.2, when only using RGB images, the U-Net architecture performs the best on the validation set while the CT-UNet architecture performs the best on the test set. After introducing LiDAR data, U-Net performs the best for the validation and test set, as seen in table 5.5 and 5.6. However, there is a large gap between the performance of the test and validation set for both tasks, as seen in table 5.9, which shows the metrics using the same weighted ensembles using TTA on both sets. The ensembles used in the table are the best-performing ensembles using TTA for task 1 and task 2, as seen in table 5.4 and 5.8.

	Validation		Test	
	Task 1	Task 2	Task 1	Task 2
IoU	0.9306	0.9384	0.8011	0.8964
BIoU	0.8503	0.8639	0.6295	0.8009
Score	0.8904	0.9012	0.7153	0.8486

Table 5.9: Metrics for the validation and test set using the best performing model ensemble with TTA for each task

The difference in metrics is most likely due to how the ground truth masks have been generated. The ground truths of the validation set have been generated using DSM, and ground truths for the test set have been generated using DTM, which skews the top of the buildings, as explained in chapter 3.1. Another factor could be that the buildings and environment in Denmark look a little different than the buildings and environment in Norway. As the training and validation set contains images from Denmark and the test set contains images from Norway, this could also cause a drop in performance between the sets due to bias. Figure 5.6 shows a comparison of predicted masks with and without the use of LiDAR data. These predictions are made on the validation set.

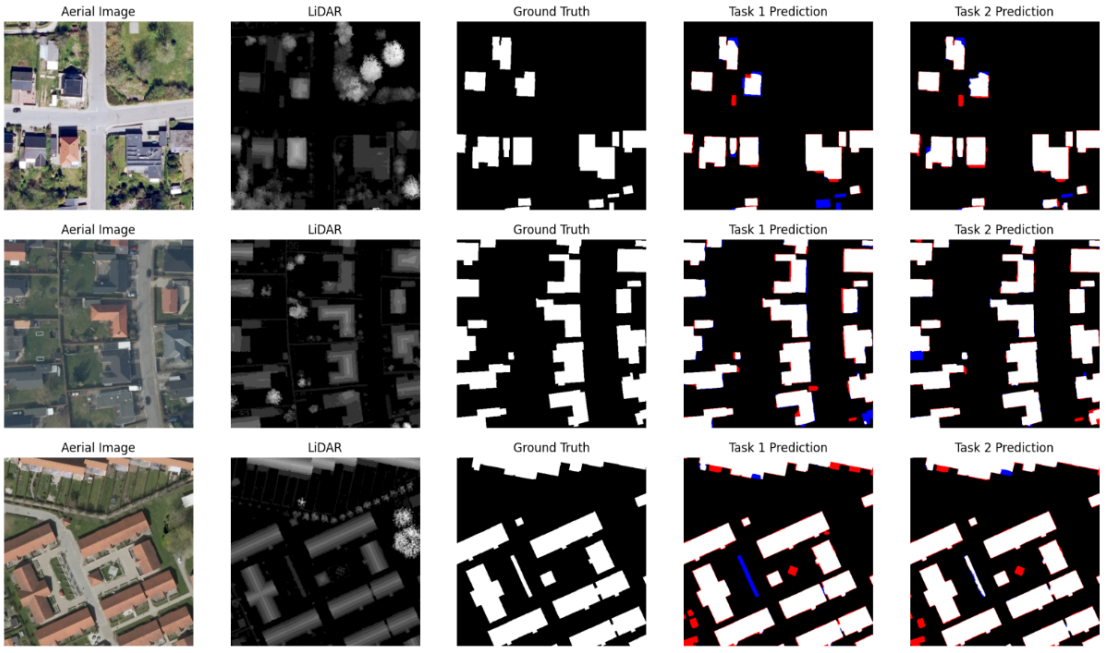


Figure 5.6: Predicted masks from tasks 1 and 2 using the validation set. White: True Positives, Black: True Negatives, Red: False Positives, Blue: False Negatives.

The top row shows that the LiDAR data prediction manages to classify most of the buildings hidden by trees in the bottom right corner. In the middle row, the prediction using only the RGB image manages to classify a white truck as a building and classify the building in the bottom right corner as background. These misclassifications are corrected when using LiDAR data. The LiDAR prediction introduces a new set of false negatives, as seen in the middle left part of the prediction. This is likely due to the part of the house not appearing in the LiDAR data. In the bottom row, we see that the LiDAR prediction manages to classify a bicycle shed that the RGB prediction classifies as background. There are also some false positives at the top building, which the LiDAR prediction correctly classifies.

Chapter 6

Conclusions

The thesis aimed to develop a machine-learning method that managed to generate segmentation masks of buildings accurately. This was split into two parts where the first part required that the methods be developed using only aerial images, and the second part required that the method be developed using LiDAR data with or without aerial images. This was achieved by creating nine models for each task. The models were created using two different architectures, U-Net and CT-UNet, and four different backbones, EfficientNetB4, EfficientNetV2S, ResNet50V2, and DenseNet201. Data augmentation, transfer learning, model ensemble, and test time augmentation were used to generate a more robust solution.

Four experiments were performed for each task, single model prediction, single model prediction using TTA, model ensemble prediction, and model ensemble prediction using TTA. These experiments concluded that ensembles of models performed better than just single models. TTA also marginally improved the result of both of these techniques. CT-UNet performed better for task 1, while U-Net performed the best for task 2. This is likely because the DBB and SCAB are less needed when using LiDAR data, as it is much easier for the network to distinguish between the foreground and background classes.

As the thesis is based on the MapAI competition, another objective was to have the method challenge the top competitors of the MapAI competition. Table 6.1 shows the metrics of the proposed method compared to the top 3 groups of the competition.

		Task 1			Task 2			
Placement	**Team**	**IoU**	**BIoU**	**Total Score**	**IoU**	**BIoU**	**Total Score**	**Final Score**
1	FUNDATOR	0.7794	0.6115	0.6955	0.8775	0.7857	0.8316	**0.7635**
2	HVL-ML	0.7879	0.6245	0.7062	0.8711	0.7504	0.8108	**0.7585**
3	DEEPCROP	0.7902	0.6185	0.7044	0.8506	0.7461	0.7984	**0.7514**
-	Our suggested method	0.8011	0.6295	0.7153	0.8964	0.8009	0.8486	**0.7820**

Table 6.1: The proposed approach compared with the top 3 groups of the MapAI competition

As seen in the table, the suggested method excels in every metric of the competition. Compared to the best group for task 1, it can be seen that the proposed method improves the IoU, BIoU, and score by 1.32%, 0.5%, and 0.91%, respectively. Compared to the best group for task 2, it can be seen that the proposed method improves the IoU, BIoU, and score by 1.89%, 1.52%, and 1.7%, respectively. This causes the final score to improve by 1.85% compared to 1st place in the competition.

Most of the top groups used U-Net for their solution [45] [46]. The main difference between theirs and our solution is that they use weaker backbones (EfficientNetB1, ResNet26d, ResNet34) but have models with more parameters, which means that their decoders are more complex. It was tried to use more complex decoders by increasing the number of kernels in the convolutional layers. However, this gave worse results than the suggested method, which means the models were too complex for the dataset. In summary, we surpass every group in the competition by finding a balance between the encoder and decoder side of the network and by using two different architectures.

6.1 Future Directions

This section contains some recommendations for future work.

More generalized dataset

As the models may be biased toward buildings in Denmark, a broader dataset containing data from multiple countries could help the models generalize more. This could increase the results of the test sets.

Edge masks

As seen in table 6.1, the BIoU is lower than the IoU, which indicates that the models have a more challenging time classifying the edges of the buildings. A solution to this could be to train the models on edge masks, where the mask only highlights the edges of the buildings. These models could be used in the ensembles in combination with the models trained on the building masks. One could simply use an edge detector on the building masks to get the edge masks.

Post-processing

Post-processing methods such as morphological opening or closing were attempted for this work. However, both methods introduced false positives and negatives. Another way to post-process the masks could be to remove all predicted buildings with an area under a certain amount of pixels, not laying on the border of the predicted mask. By doing this with a low pixel area, misclassified objects such as cars and sheds will be removed from the predictions.

Better hyper-parameter tuning

The hyper-parameters for U-Net and CT-UNet were only tuned on RGB images. These hyper-parameters were then used for the models using RGB images and LiDAR data. Tuning the models used for task 2 on the RGB and LiDAR set could help improve the predictions for task 2.

The hyper-parameters for U-Net have been tuned without any backbone, and the hyper-parameters for CT-UNet have been tuned with only EfficientNetB4 as the backbone. The hyper-parameters found in these experiments are then used for every backbone for that architecture. Finding and using the best hyper-parameter for each of the nine models used in the thesis would likely increase the results. This is, however, a time-consuming task.

Appendix A

Poster

This appendix contains the poster presented at UiS on the 1st of June, 2023. The pages have been scaled down from A3 to A4 to fit the format of the thesis better.

An Efficient Machine Learning Approach
Images and

Made by Erik Finnesand, Supervised by

Introduction

Buildings are important for population information, policy-making, and city management. Various computer vision technologies such as object detection, classification, and segmentation have proven helpful in urban planning, disaster management, and city modeling scenarios, where segmentation has proven the most precise method as it highlights areas of interest.

Training data is derived from real-world satellite images. This poses some challenges as the images may have varying qualities, contain noise, and consist of large class imbalances. Smaller buildings are harder to detect as they may be obstructed by other objects. Shadows and water reflection may also be mistaken as buildings. These variables make acquiring accurate segmentation masks of building challenging.

This work is based on the NORA MapAI competition hosted last year. The competition is split into two tasks. Task 1 requires the participant to develop machine learning models for generating accurate segmentation masks of buildings using only aerial images. Task 2 requires the participant to do the same with the inclusion of LiDAR data.The main objective of this work is to create a solution that challenges the top groups attending the MapAI competition.

Dataset

The official MapAI competition dataset is used. It consists of 4 splits:

- Train: 7500 Data samples
- Validation: 1500 Data samples
- Test 1: 1368 Data samples
- Test 2: 978 Data samples

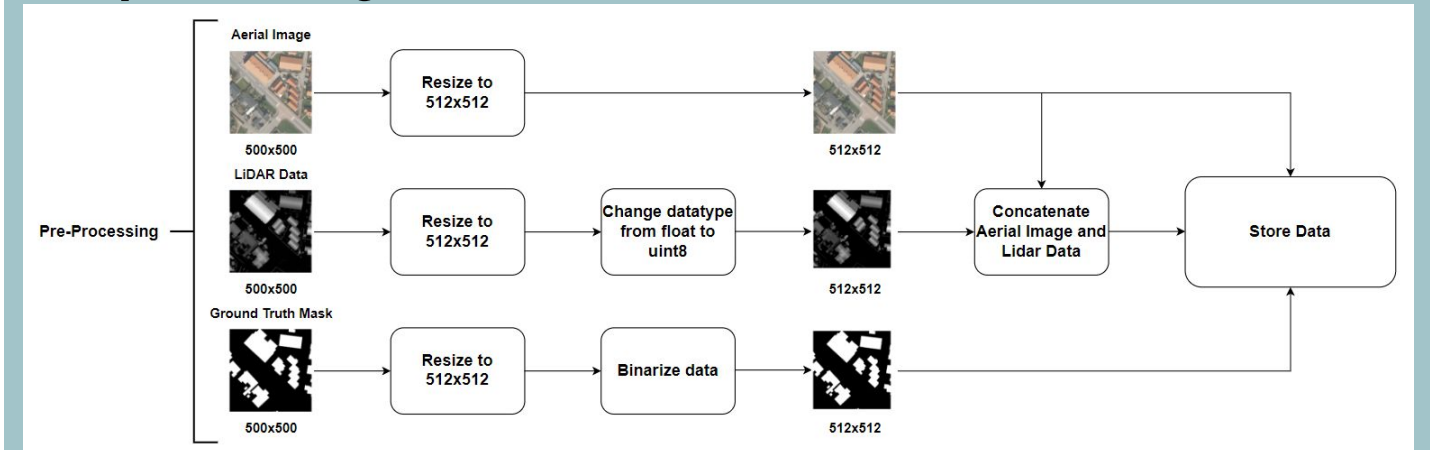
The train and validation split contains samples from Denmark. Both test splits contains samples from Norway. As the data is derived from a production setting there may be differences between the data and the ground truth mask. The ground truth mask for the test set has also been generated using DTM, which skews the top of the buildings.

Data sample taken from train set

for Building Segmentation Using Aerial LiDAR Data

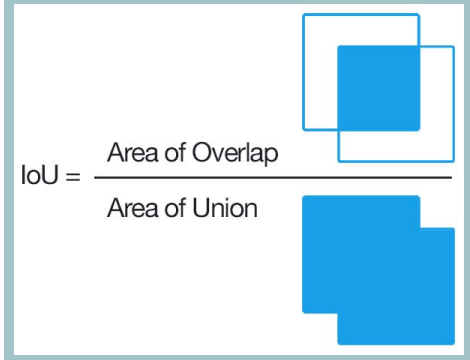
Mina Farmanbar and Muhammad Sulaiman

Pre-processing

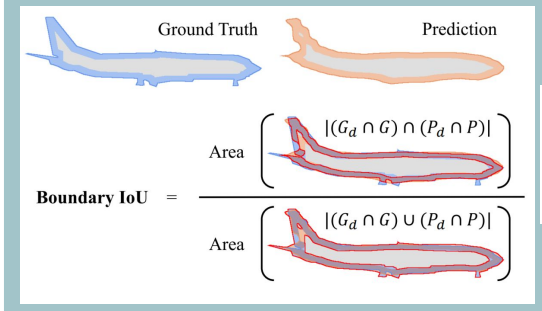


Metrics

IoU [3]



BIoU [4]



Score

$$Score = \frac{IoU + BIoU}{2}$$

Training

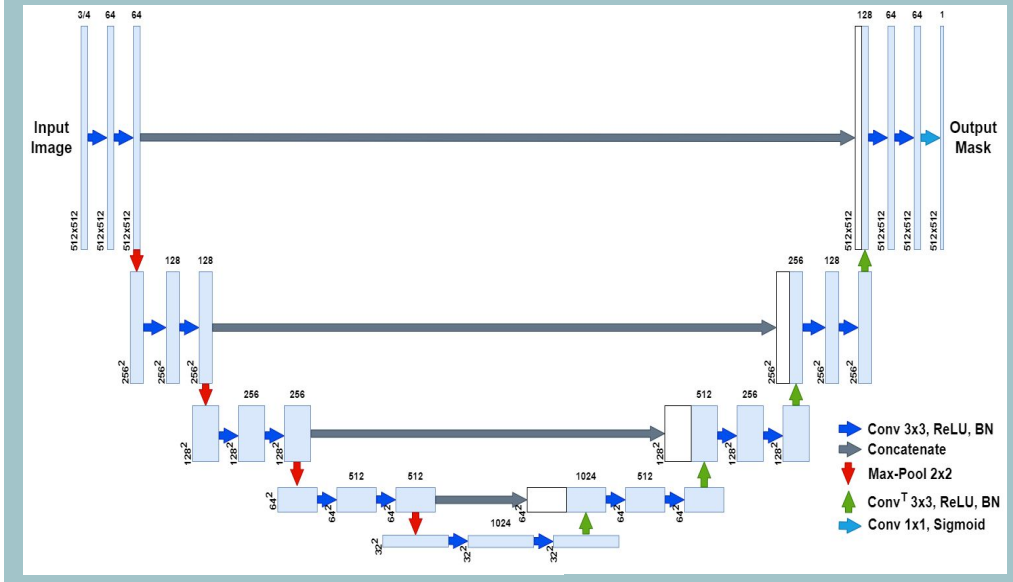
Architecture	U-Net	CT-UNet
Loss	Dice loss	
Optimizer	ADAM	
Initial LR	0.0001	0.00005
Batch size	6 or 12	10 or 12
Callbacks	EarlyStopping, ReduceLROnPlateau and ModelCheckpoint	

Architecture	Backbone	Batch Size	Parameters
U-Net	None	6	34,540,737
U-Net	EfficientNetB4	12	18,253,672
U-Net	EfficientNetV2S	12	19,069,561
U-Net	ResNet50V2	12	17,541,441
U-Net	DenseNet201	12	24,746,881
CT-UNet	EfficientNetB4	10	17,831,856
CT-UNet	EfficientNetV2S	12	19,413,761
CT-UNet	ResNet50V2	12	17,803,401
CT-UNet	DenseNet201	10	24,831,177

Methodology

U-Net [1]

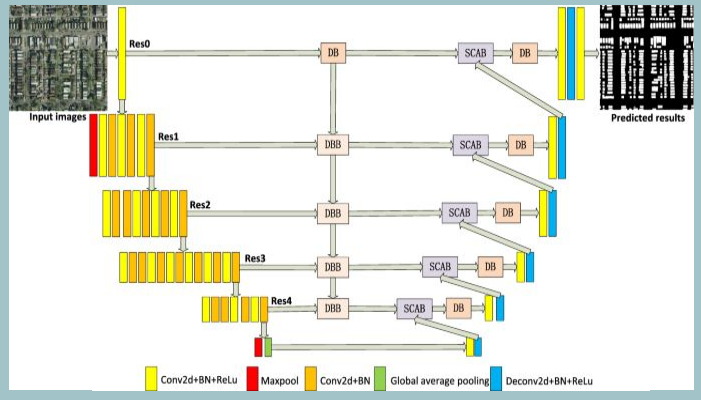
U-Net is a Encoder-Decoder architecture.
Encoder: Finds features. Decoder: Localizes these features.

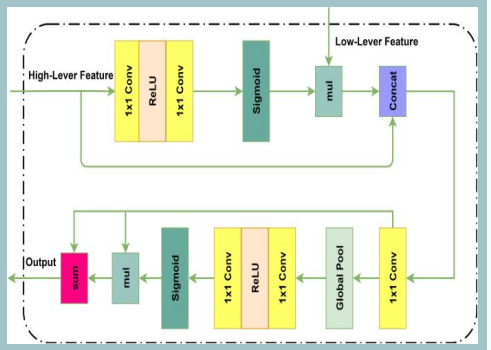


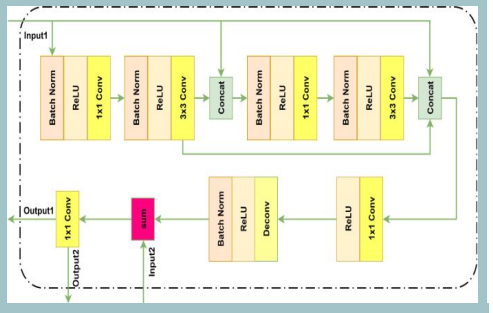
Context-Transfer-UNet [2]

DBB: Enhance recognition capabilities and expand the distinction between the classes.

SCAB: Combine context space information and select more distinguishable features from space and channel.







Transfer learning and backbones

Transfer learning is a method that lets us use a pre-trained model as the starting point for training a new model, called backbones. These models are trained on image classification problems and replace the encoder side of the segmentation network. These model are trained on ImageNet.

Online Data Augmentation

Data augmentation is a technique used to artificially increase the training set by creating modified copies of already existing data. The augmentation is performed when loading batches during training.

Model Ensemble

The main idea of a model ensemble is to combine predictions from multiple models. As the models have been trained using different architectures and backbones, the strength of one model may compensate for the weakness of another.

Test Time Augmentation (TTA)

The main idea of TTA is to generate multiple augmented versions of a testing image and make a prediction for each of them. All the predictions are then combined to make a final prediction.

References

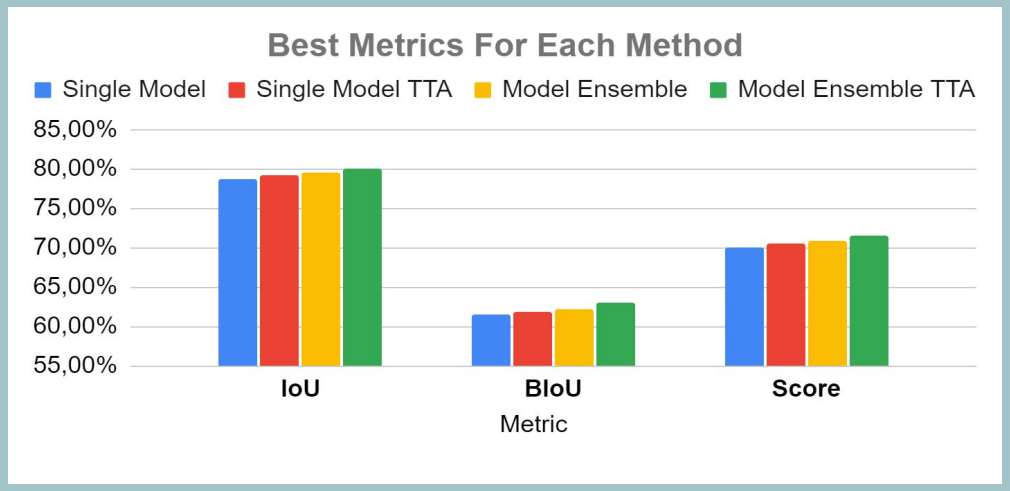
[1] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Medical Image Computing and ComputerAssisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18, pages 234–241. Springer, 2015.

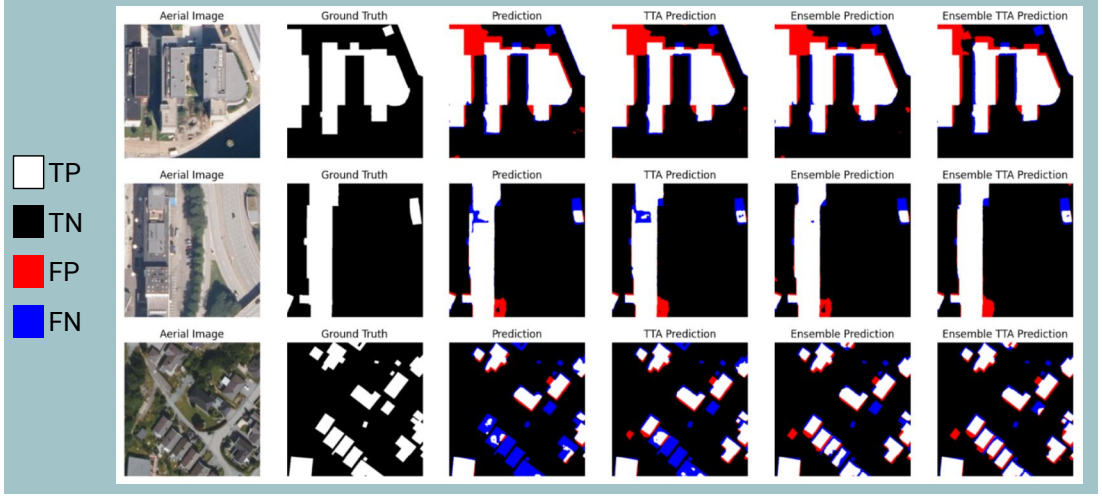
[2] Sheng Liu, Huanran Ye, Kun Jin, and Haohao Cheng. Ct-unet: Context-transferunet for building segmentation in remote sensing images. Neural Processing Letters, 53:4257–4277, 2021.

[3] https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

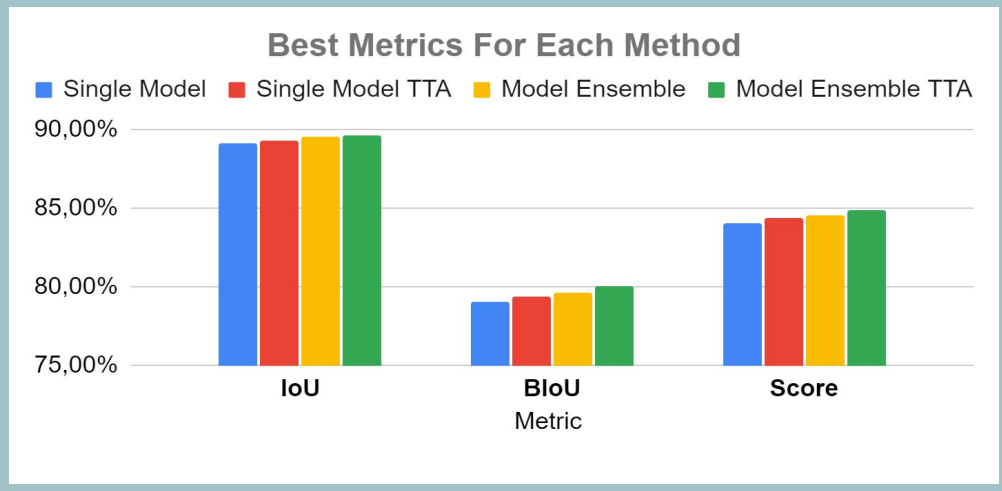
[4] Bowen Cheng, Ross Girshick, Piotr Dollár, Alexander C Berg, and Alexander Kirillov. Boundary iou: Improving object-centric image segmentation evaluation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 15334–15342, 2021.

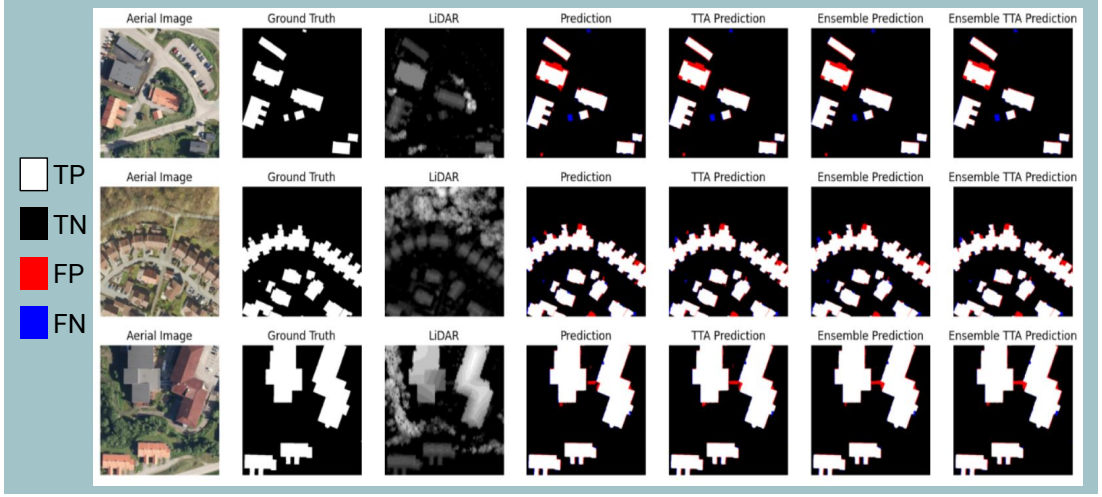
Task 1 Results





Task 2 Results





Conclusion

Because of techniques such as spatial attention, transfer learning, backbones, data augmentation, test time augmentation, and ensembles, our method improves the score of Task 1 by 0.91% and Task 2 by 1.7% compared to the best groups in each category. Our final score improved by 1.85% compared to the best group.

		Task 1			Task 2			
Placement	**Team**	**IoU**	**BIoU**	**Total Score**	**IoU**	**BIoU**	**Total Score**	**Final Score**
1	FUNDATOR	0.7794	0.6115	0.6955	0.8775	0.7857	0.8316	**0.7635**
2	HVL-ML	0.7879	0.6245	0.7062	0.8711	0.7504	0.8108	**0.7585**
3	DEEPCROP	0.7902	0.6185	0.7044	0.8506	0.7461	0.7984	**0.7514**
-	Our suggested method	0.8011	0.6295	0.7153	0.8964	0.8009	0.8486	**0.7820**

Appendix B

Preliminary Experiments

Preliminary experiments were done to determine the best hyperparameters for U-Net and CT-UNet. These experiments were done on task 1 of the thesis.

B.1 U-Net Preliminary Results

All the preliminary tests for U-Net are done without any backbones.

B.1.1 Batch Normalization Placement

This experiment determines whether BN should be placed before or after the activation function. An initial learning rate of 0.0001, dice loss, and a batch size of 6 are used.

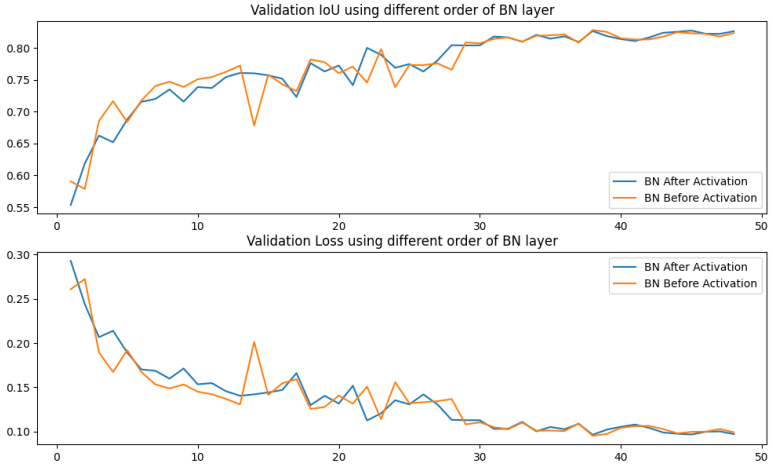


Figure B.1: Validation IoU and Loss during training using different orders of BN

BN after the activation function yields a 0.0029 higher validation IoU and more stable training progress. Because of this, it is decided that BN should be performed after the activation function for the U-Net models.

B.1.2 Initial Learning Rate

This experiment determines the best initial learning rate for the models. Dice loss and a batch size of 6 are used.

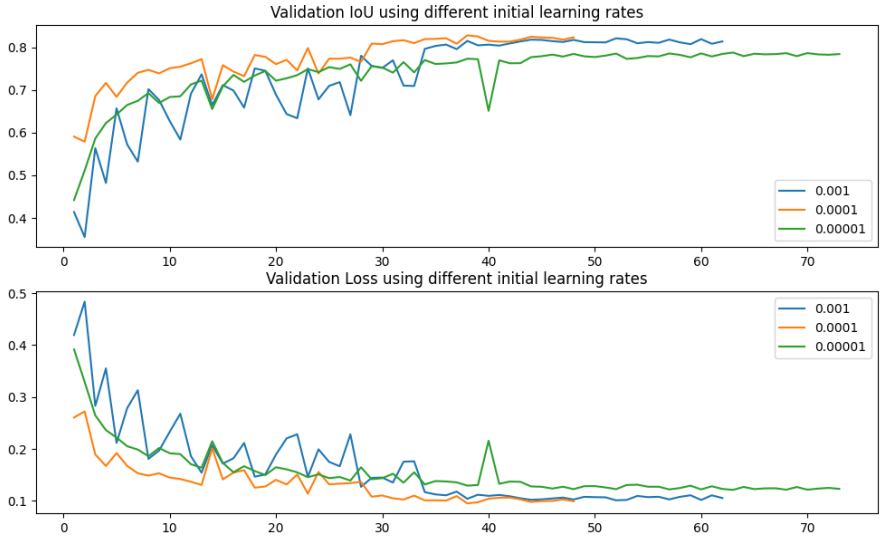


Figure B.2: Validation IoU and Loss during training using different initial LR

The figure shows that an initial learning rate of 0.0001 yields the highest validation IoU. This is also the initial learning rate that converges the quickest. Because of this, it is decided that the initial learning rate for the U-Net models should be 0.0001.

B.1.3 Loss Function

This experiment determines the best loss function for the models. An initial learning rate of 0.0001 and a batch size of 6 are used.

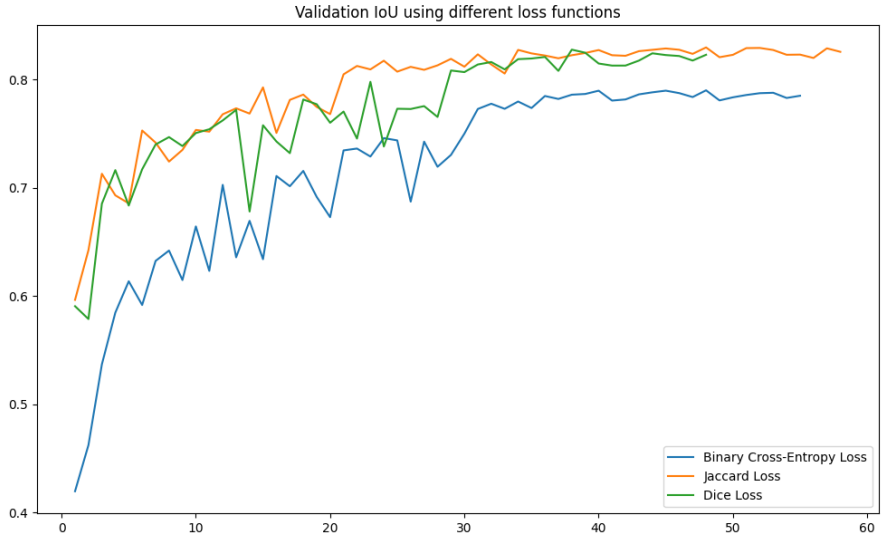


Figure B.3: Validation IoU and Loss during training using different loss functions

Dice and Jaccard loss performs quite the same regarding validation IoU, where Dice loss has a validation IoU 0.0001 higher than Jaccard loss. Dice loss finished the training ten epochs before Jaccard loss. Because of this, it is decided that the loss function for the U-Net models should be Dice loss.

B.2 CT-UNet Preliminary Results

All the preliminary tests for CT-UNet are done with EfficientNetB4 as the backbone. They are done on task 1 of the thesis to determine the best possible hyperparameters for the CT-UNet architecture.

B.2.1 Batch Normalization Placement

This experiment determines whether BN should be placed before or after the activation function. An initial learning rate of 0.0001, dice loss, and a batch size of 6 are used.

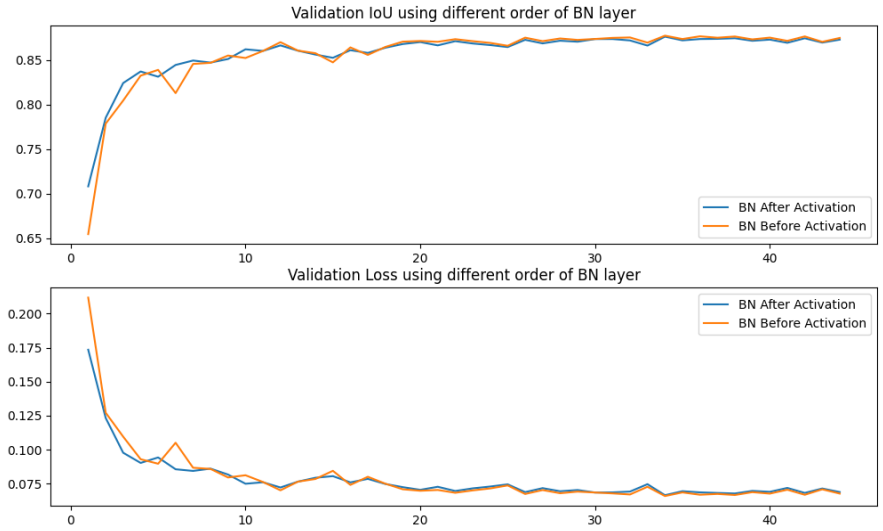


Figure B.4: Validation IoU and Loss during training using different orders of BN

BN before the activation function yields a 0.0019 higher validation IoU and BN after the activation function gives a more stable training progress. As the results are very similar, we look at the achieved metrics on the test set to determine which method should be used.

Model	IoU	BIoU	Score
CT-UNet BN After ReLU	0.7785	0.6050	0.6918
CT-UNet BN Before ReLU	0.7612	0.5907	0.6759

Table B.1: Metrics for the test set different orders of BN

The table shows that using BN after the activation function yields higher metrics on the test set. Because of this, it is decided that BN should be performed after the activation function for the CT-UNet models.

B.2.2 Initial Learning Rate

This experiment determines the best initial learning rate for the models. Dice loss and a batch size of 6 are used.

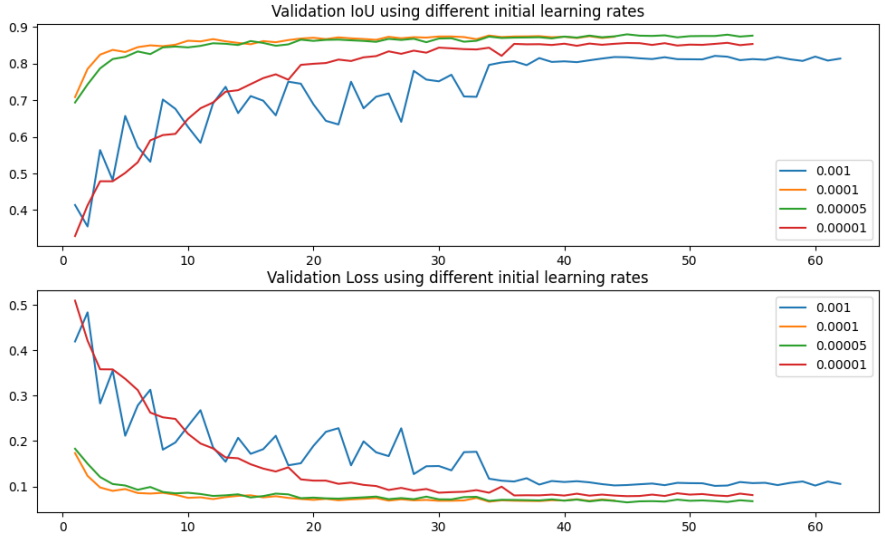


Figure B.5: Validation IoU and Loss during training using different initial LR

The figure shows that an initial learning rate of 0.00005 yields the highest validation IoU. Because of this, it is decided that the initial learning rate for the CT-UNet models should be 0.00005.

Appendix C

Code and Instructions

This appendix explains the Python scripts used in this thesis and how to compile and run the scripts.

The code can be found on the project GitHub page: https://github.com/erikx50/A-Machine-Learning-approach-for-Building-Segmentation-using-laser-data.

The best-performing model ensembles can be found in this Google drive: https://drive.google.com/drive/folders/1FsF-B6xUvm2ZP7gcQ5ke7vl7XHlGSCsj.

C.1 Scripts

The scripts are found in the code folder.

- `Load_Dataset.py`
  Downloads the MapAI dataset and creates a folder where the dataset is stored.
- `Preprocess_Dataset.py`
  Preprocesses the MapAI dataset and creates subfolders containing the preprocessed data in the dataset folder.
- `Train_Model.py`
  Train a selected model on a chosen task.
- `Test_Model.py`
  Tests and prints a model's IoU, BIoU, and Score. Users can choose between the validation and test sets. Users can also choose to enable TTA predictions.

- `Model_Ensemble.py`
  Tests and prints the IoU, BIoU, and Score of an ensemble of models. Users can choose between the validation and test sets. Users can also choose to enable TTA predictions. This script can run three types of ensemble methods.

  1. 3 model ensemble with set weights.
  2. Find the best weights for the ensemble on that specific set using three models.
  3. Find the best ensemble and weights on that specific set using multiple models.

- `UNet.py`
  It contains the code for the U-Net architecture.

- `CTUNet.py`
  It contains the code for the CT-Unet architecture.

- `Loss_Metrics.py`
  It contains the code for IoU and Dice coefficient metric and loss function.

- `eval_functions.py`
  It contains the code of the evaluation functions used for the MapAI competition. This code is taken from the official MapAI competition GitHub: https://github.com/Sjyhne/MapAI-Competition/blob/master/competition_toolkit/competition_toolkit/eval_functions.py.

- `utils.py`
  It contains the code for loading the dataset when testing and test time augmentation.

C.2 Compile and Run

Before running the scripts, install the required Python packages from `requirements.txt`. These required packages are:

Packages	**Version**
datasets	2.8.0
numpy	1.23.5
opencv_python	4.6.0.66
Pillow	9.4.0
tensorflow	2.11.0
tifffile	2023.2.3
tqdm	4.64.1

Start by running `Load_Dataset.py`. This script will download the dataset from HuggingFace and create a new folder in the root folder called "dataset." Each dataset split

will be placed in its own folder, where each folder contains a folder for aerial images, LiDAR data, and ground truths.

The next step is to pre-process the data. This is done by running the `Preprocess_Dataset.py` script. This script will create new folders in the dataset folder for each split.

If the user wishes to train a model, the script `Train_Model.py` has to be run. The user will be asked which GPU to run the training progress on, which train set to use, which type of model to use, and what name to save the model as. If this is the first time running the script, a model folder will be created in the root folder. In this folder, the model will be saved in one of two subfolders, task1 or task2, depending on the task. For the train set selection, RGB represents task 1, and RGBLiDAR represents task 2.

If the user wishes to test a single model, the script `Test_Model.py` has to be run. The user will be asked which GPU to run the testing progress on, which set the model was trained on, which set to use when testing the model (validation or test), if TTA should be performed, and the model's file name. This file will print the metrics of the selected model.

If the user wishes to test an ensemble of models, the script `Model_Ensemble.py` has to be run. The user will be asked for the same prompts as `Test_Model.py` and which type of ensemble method to run.

1. `Ensemble with set weights.`
   Input the name of the three models to use for the ensemble. After the models have predicted on the set, input the weights of the different models separated with a comma.

2. `Find best weights from a set of 3 models.`
   Input the name of the three models to use for the ensemble. After the models have predicted on the set, every different combination of weights will be tried. The function will print out the best combination of weights and the achieved score.

3. `Find best 3 model ensemble using multiple models.`
   Input as the name of the models separated with a comma. The script will create every possible 3-model combination and try every possible combination of weights using these ensembles. The top 10 ensembles will be printed with their weight and achieved metrics.

Appendix D

NORA Annual Conference 2023

The NORA Annual Conference 2023 was held in Tromsø on the 5-6th of June. After submitting an abstract to the conference, I got accepted to hold an oral presentation of the work done in this thesis.

At the end of the conference, I received the best oral presentation award.

Figure D.1: Certificate for best oral presentation at the NORA Annual Conference 2023

Bibliography

[1] Mapai: Precision in building segmentation. https://www.nora.ai/competition/mapai-precision-in-building-segmentation/. Accessed: 2023-04-24.

[2] Python. https://www.python.org/. Accessed: 2023-03-28.

[3] Tensorflow. https://www.tensorflow.org/. Accessed: 2023-03-28.

[4] Keras. https://keras.io/. Accessed: 2023-03-28.

[5] Datasets. https://pypi.org/project/datasets/. Accessed: 2023-03-28.

[6] Huggingface. https://huggingface.co/. Accessed: 2023-03-28.

[7] Numpy. https://numpy.org/. Accessed: 2023-03-28.

[8] Opencv. https://opencv.org/. Accessed: 2023-03-28.

[9] Matplotlib. https://matplotlib.org/. Accessed: 2023-03-28.

[10] Xiaoye Liu. Airborne lidar for dem generation: some critical issues. *Progress in physical geography*, 32(1):31–49, 2008.

[11] Facundo Bre, Juan Gimenez, and Víctor Fachinotti. Prediction of wind pressure coefficients on building surfaces using artificial neural networks. *Energy and Buildings*, 158, 11 2017. doi: 10.1016/j.enbuild.2017.11.045.

[12] Sigmoid activation function. https://insideaiml.com/blog/Sigmoid-Activation-Function-1031. Accessed: 2023-04-11.

[13] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[14] Mihai RADU, Ilona COSTEA, and Valentin Stan. Automatic traffic sign recognition artificial inteligence - deep learning algorithm. pages 1–4, 06 2020. doi: 10.1109/ECAI50035.2020.9223186.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[16] Segmentation of organs in medical images with artificial intelligence. https://www.plainconcepts.com/segmentation-organs-with-artificial-intelligence/. Accessed: 2023-04-11.

[17] Hemayet Chowdhury, Md Imon, Anisur Rahman, Aisha Khatun, and Md Saiful Islam. A continuous space neural language model for bengali language. 01 2020.

[18] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[19] Stanford cs231. https://cs231n.github.io/neural-networks-3/#anneal. Accessed: 2023-03-29.

[20] Batch size image. https://towardsdatascience.com/gradient-descent-clearly-explained-in-python-part-2-the-compelling-code-c21ee26fbc28. Accessed: 2023-03-29.

[21] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[22] Convolutional neural network 3: convnets and overfitting. https://aigeekprogrammer.com/convnets-and-overfitting/. Accessed: 2023-04-11.

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

[24] Nistala V.E.S. Murthy. Freehand sketch-based authenticated security system using convolutional neural network. *International Journal of Engineering and Advanced Technology*, 9, 12 2019. doi: 10.35940/ijeat.B4412.129219.

[25] Convolutional neural networks (cnn): Step 1- convolution operation. https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-1-convolution-operation. Accessed: 2023-04-06.

[26] What is transposed convolutional layer? https://towardsdatascience.com/what-is-transposed-convolutional-layer-40e5e6e31c11. Accessed: 2023-04-10.

[27] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[28] Max-pooling / pooling. https://computersciencewiki.org/index.php/Max-pooling_/_Pooling. Accessed: 2023-04-10.

[29] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[30] 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.

[31] Sheng Liu, Huanran Ye, Kun Jin, and Haohao Cheng. Ct-unet: Context-transfer-unet for building segmentation in remote sensing images. *Neural Processing Letters*, 53:4257–4277, 2021.

[32] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, pages 630–645. Springer, 2016.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[35] Review: Densenet — dense convolutional network (image classification). https://towardsdatascience.com/review-densenet-image-classification-b6631a8ef803. Accessed: 2023-04-13.

[36] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[37] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[38] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[39] Mingxing Tan and Quoc Le. Efficientnetv2: Smaller models and faster training. In *International conference on machine learning*, pages 10096–10106. PMLR, 2021.

[40] Truong Dang, Tien Thanh Nguyen, Carlos Francisco Moreno-García, Eyad Elyan, and John McCall. Weighted ensemble of deep learning models based on comprehensive learning particle swarm optimization for medical image segmentation. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 744–751. IEEE, 2021.

[41] Bowen Cheng, Ross Girshick, Piotr Dollár, Alexander C Berg, and Alexander Kirillov. Boundary iou: Improving object-centric image segmentation evaluation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15334–15342, 2021.

[42] Sander Jyhne, Morten Goodwin, Per-Arne Andersen, Ivar Oveland, Alexander Salveson Nossum, Karianne Ormseth, Mathilde Ørstavik, and Andrew C Flatman. Mapai: Precision in building segmentation. *Nordic Machine Intelligence*, 2:1–3, 9 2022. ISSN 2703-9196. doi: 10.5617/NMI.9849. URL https://journals.uio.no/NMI/article/view/9849.

[43] Ketut Wikantika. Three dimensional city building modellingwith lidar data (case study: Ciwaruga, bandung). 2018.

[44] Mapai-competition/eval_functions.py. https://github.com/Sjyhne/MapAI-Competition/blob/master/competition_toolkit/competition_toolkit/eval_functions.py. Accessed: 2023-02-13.

[45] Lars Martin Hodne and Eivind Hovdegård Furdal. Team fundator: Weighted unet ensembles with enhanced datasets. *Nordic Machine Intelligence*, 2(3), 2022.

[46] Satheshkumar Kaliyugarasan and Alexander Selvikvåg Lundervold. Lab-net: Lidar and aerial image-based building segmentation using u-nets. *Nordic Machine Intelligence*, 2(3), 2022.