UiT The Arctic University of Norway

Faculty of Science and Technology
Department of Physics and Technology

Uncertainty Guided Polygon Generation for Building Detection

Christian Salomonsen

FYS-3941 Master's Thesis in Applied Physics and Mathematics 30 SP - June 2024

Abstract

Enhanced accuracy of building detection algorithms has the potential to benefit a wide array of applications, including urban planning, environmental monitoring, and disaster response efforts. However, building extraction algorithms struggle with robustness due to among others, occlusions from vegetation and shadows of nearby tall buildings, complex building shapes, and a large distributional shift between datasets that come in varying spatial resolutions, resulting in their dependency of dataset-specific and user specified parameters. Towards addressing this shortcoming, we hypothesize that the model's uncertainty can be leveraged to increase the robustness and efficacy of these algorithms. As a first step towards evaluating this hypothesis, we propose an improved version of the current state-of-the-art that incorporates quantification of the model uncertainty. We further show that leveraging these uncertainty methods by guiding the vertex selection process through the use of a dynamic threshold improves the stability across datasets. Results on two datasets demonstrate that incorporating uncertainty has the potential to significantly improve the robustness of the previous state-of-the-art method. Additionally, the dynamic threshold, while offering a more modest improvement, showcases the potential of actively leveraging uncertainty measures to improve the model performance.

Acknowledgements

I would like to express my sincere gratitude to all those who have supported and guided me through the course of my master's program. Firstly, I extend my heartfelt thanks to my supervisor, Michael Kampffmeyer, for your unwavering guidance, insightful feedback, and encouragement. I am also deeply grateful to all my friends, whose camaraderie truly enabled me to persevere. In particular, Tobias and Iver has made this journey a very fun one; I truly appreciated the long discussions about food, politics, and everything in between. And to Rebecca, for your endless encouragement and help. Your support has meant the world to me.

Lastly, I would like to thank my loving and supporting family, who has always had a time and space for me, providing the best conditions for my success. Your belief in my has been my greatest motivation.

Contents

List of Figures

List of Tables

List of Abbreviations

ACM Active Contour Model

AFM Attraction Field Map

AGNN Attentional Graph Neural Network

AL Active Learning

ANN Artificial Neural Network

AP Average Precision

AR Average Recall

ASM Active Skeleton Model

BCE Binary Cross Entropy

BNN Bayesian Neural Network

C-IOU Complexity Aware Intersection over Union

CE Cross Entropy

CNN Convolutional Neural Network

DCNN Deep Convolutional Neural Network

DL Deep Learning

DNN Deep Neural Network

DP Douglas-Peucker

DSM Digital Surface Model

ECA Efficient Channel Attention

FCN Fully Convolutional Network

FCNN Fully Connected Neural Network

FFL Frame Field Learning

GCN Graph Convolutional Network

GGNN Gated Graph Neural Network

GIS Geographical Information Systems

GNN Graph Neural Network

IOU Intersection over Union

KL Kullback-Leibler

LSD Line Segment Detection

MAV-ATTR Mask-and-Vertices Attraction

MC Monte Carlo

MCD Monte Carlo Dropout

MCMC Markov Chain Monte Carlo

ML Machine Learning

MLP Multilayer Perceptron

MS COCO Microsoft Common Objects in COntext

MSE Mean Squared Error

MTL Multi-Task Learning

NMS Non Maximum Suppression

NN Neural Network

NR Norwegian Computing Center

ReLU Rectified Linear Unit

RGB Red, Green, and Blue

RNN Recurrent Neural Network

1 Introduction

The increase in availability of on-demand, high-resolution optical satellite imagery, combined with the widespread adoption of Deep Learning (DL) techniques for image analysis, has opened up new possibilities for various industries. Fields such as vegetation monitoring, agriculture, and forestry mapping [1, 2, 3, 4, 5, 6], urban planning, and land use [7, 8], environmental monitoring [9], and disaster management [10] have all benefited from these advancements.

In urban mapping, the use of Deep Convolutional Neural Networks (DCNNs) to extract building footprints has become a growing research focus [11, 12, 10, 13, 14, 15, 16, 17]. Despite significant progress, building detection from high-resolution satellite imagery remains a complex and challenging task. Traditional methods often depend on manual labor, such as digitizing or applying simple image processing techniques, which are time-consuming and prone to human error [18, 19, 20, 21]. The varying shapes, sizes, and orientations of buildings, along with occlusions from vegetation and shadows, make precise delineation difficult. Additionally, the dynamic nature of urban environments requires methods that can adapt to changes and provide accurate, up-to-date information [22]. DL offers the potential to automate this process, reducing labor and time while providing more precise and consistent results.

Recent developments have shown that segmentation masks can limit certain downstream applications. To address this, DL algorithms such as [12, 13, 15, 17, 23, 24, 25], have been developed to produce vector polygons directly from

segmentation masks. These vector polygons are compatible with modern Geographical Information Systems (GIS) software, whereas traditional segmentation models require an additional vectorization step that can introduce significant ambiguities near building edges. The reformulation of the problem to a vector extraction one, has the potential to tailor solutions that leverage inherent properties in images of buildings, such as walls, corners, and orientations [12, 13, 14, 17].

Due to challenges in the transformation of a binary mask into a vector representation that maintain precision and robustness, the architectures have been distinguished into two broad categories. Direct extraction of polygons in an end-to-end manner, that leverage inherent properties in the vector representation, for instance, by mimicking the graph-view of polygons employing Graph Neural Networks (GNNs) [25], or by sequentially processing each vertex using Recurrent Neural Networks (RNNs) [23, 24, 25, 15]. These methods tend to struggle with complex building shapes, not being able to detect *holes* in buildings, such as courtyards, are generally harder to train, and often require multiple iterations during inference.

The remaining category of architectures uses segmentation maps in conjunction with additional information, such as edges or vertices, before a separate step combines these into a polygon [13, 17], some methods also employ a final refinement step [12]. While these methods tend to be simpler in their processing chains, using easier polygon generation algorithms, they lose the end-to-end learned nature. Additionally, errors from the segmentation mask prediction often tend to propagate to later stages, without correction.

1.1 Key challenges

We identify two categories of key challenges. We begin by considering those related to the dataset. Buildings come in many shapes, sizes and orientations, a DL based solution needs to be robust in its ability to produce consistent predictions. However, buildings with problematic characteristics such, as rounded shapes, and holes, are often underrepresented in most datasets, and are in general hard to accurately delineate. This leads to poor performance on buildings that have complex shapes, where a human counterpart would have an easier time. Another problem arise when the building footprints are occluded by shadows from other tall structures or vegetation. While recent DL models show promising results for partly occluded buildings [15, 17], the issue prevails, and is a common pitfall for many automated algorithms.

The other category deals with issues related to the frameworks itself. The

usefulness of vector polygons are limited by their quality. Common challenges in these architectures stem from their reliance on the segmentation mask which, like the traditional methods, suffer when building edges have significant ambiguities. The work of [17] specifically address the issue of *mask reversibility*, that is, the problem with imprecise borders in the learned segmentation maps during the conversion into a vector representation, however the considerable gap in qualitative performance between their predicted segmentation masks, and polygons, suggests the issue still remains. While these new methods have shown impressive performance, they often tend to be less robust as their vectorization scheme assumes a correct prediction of a set of vertices, which are subsequently merged through different merging schemes involving hand-tuned hyperparameters.

1.2 Contribution

In this thesis, we hypothesize that both the above-mentioned challenges can be addressed by leveraging the model uncertainty. In particular, we address the issue of mask reversibility by improving the robustness of an existing model, by reformulating their network into a Bayesian Neural Network (BNN), to incorporate recent developments in Bayesian inference methods [7, 26, 27, 28, 29], we can quantify the uncertainties in the predictions, in an attempt of improving its robustness and precision in building extraction. We propose an improved version of the current state-of-the-art, that model the *epistemic* uncertainty of the architecture using Monte Carlo Dropout (MCD). We further leverage the quantified uncertainty to guide the vertex selection process by a dynamic threshold modification to the framework, with inspiration from [30]. Thus, actively leveraging the uncertainty provided through the MCD process. To evaluate our contributions, we conduct experiments on a novel, proprietary high-resolution dataset recorded by *Field*, and provided to us by the Norwegian Computing Center (NR).

To summarize, our contributions are:

1. Incorporating Monte Carlo Dropout into the current state-of-the-art [17], to quantify uncertainty in the model, and improve its robustness.

2. We further employ a dynamic threshold to guide the vertex selection process by actively leveraging the epistemic uncertainty of the model.

3. Evaluating the current state-of-the-art Deep Learning model on a proprietary, novel Norwegian dataset.

Through the work in this thesis, we are taking a first step towards validating our hypothesis, and demonstrating the potential of leveraging uncertainty in this area as well as outlining promising directions that can be explored in this domain.

1.3 Outline

We begin Part I by building a foundation in Neural Networks (NNs), in Chapter 2, exploring how a mathematical function is able to adapt to data, and thereby learn. We also discuss regularization techniques, which will be important for our method. Chapter 3 brings forth Convolutional Neural Networks (CNNs), and DCNNs to highlight how NNs process images efficiently. The chapter will also present key frameworks that in practice extract information dense representations that modern complex frameworks use. The chapter also touches on subjects that are vital in building detection, such as segmentation, and the architectures which are common for such tasks. Chapter 4 delves into related works in building footprint extraction, explaining their key novelties. The reader should pay particular attention to Section 4.4, where the framework which will be modified in our method is presented. The last chapter, Chapter 5, provides an outline of uncertainty estimation in DL, explaining concepts such as MCD, and how this produces uncertainty maps of the predictions.

Part II presents our novelty, and motivates our use of uncertainty maps to refine a building detection algorithm. It is then followed by the experiment in Part III, that begins with a description of the datasets, evaluation metrics and implementation details before presenting our results. We round this chapter off with presenting ideas for future extensions that we see fruitful.

Part I

Background and Literature Review

2

General theory of Neural Networks

NNs are computational models with an interconnected network of artificial neurons, inspired by the neurons found in the human brain. With similarities to how the brain uses electrical impulses to exchange and process information using neurons [31], Artificial Neural Networks (ANNs) consist of interconnected nodes (or artificial neurons) arranged in layers. Each connection between the nodes is associated with a weight that determines the strength of the connection. The power of NNs lies in their ability to learn patterns and relationships from data. By adjusting the weights controlling the strength of the connections between neurons during a training process, NNs can learn to make accurate predictions, classify objects, generate text and even simulate art.

In this section, we will dive into the fundamental concepts of NNs. Starting with its most basic building block, *the perceptron*, in Section 2.1, and explore how these models are trained to perform complex tasks through a process of learning and optimization. Section 2.2 explain how the perceptron can learn from data, before the topic of optimization algorithms is brought up in Section 2.3. At this point we extend the idea of a perceptron into a layer of multiple perceptrons, by introducing the Multilayer Perceptron (MLP) in Section 2.4. We then introduce the backpropagation algorithm (Section 2.5) before looking at activation functions and rounding the chapter off with regularization algorithms in Section 2.7.

2.1 Perceptrons

The perceptron was developed as a means to answer how information is stored or remembered, and how does this information affect the response to some stimuli by a biological system [31]. Functionally, we use it as a processing element, that given some input $x_j, j = 1, \ldots, d$ mapped to a single output variable y while associating a weight w_j between each input-to-output connection, following the format of [32],

$$\begin{aligned} y &= \sum_{j}^{d} w_j x_j + w_0 x_0, \\ &\text{or} \\ y &= \boldsymbol{w}^T \boldsymbol{x}. \end{aligned} \tag{2.1}$$

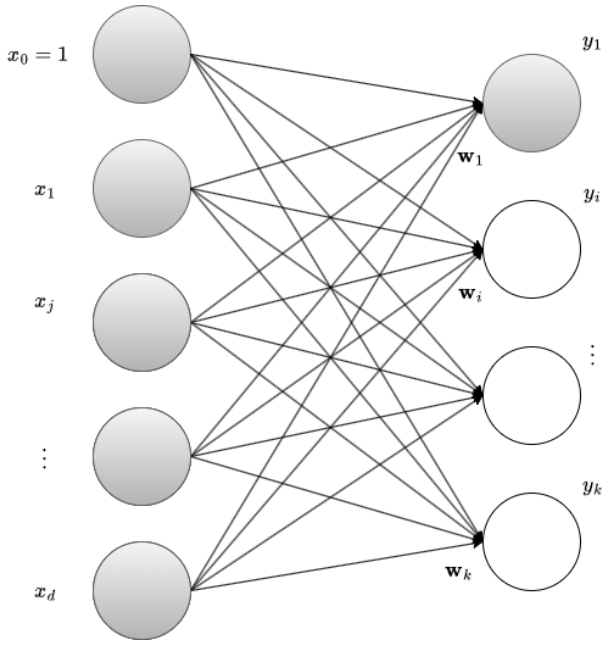

Figure 2.1: Parallel perceptrons, denoted by $y_i, i = 1, \ldots, K$, receiving inputs from $x_j, j = 1, \ldots, d$. $\boldsymbol{w}_i$ denotes the weight of the connection from the input to the output y_i. By only using the gray input to output matches, we see the form of a single perceptron. Illustration is inspired by [32].

Equation 2.1 has an additional bias unit, which we always set to 1. Then we may rewrite the perceptron as the bottom equation, having modified the input as $\boldsymbol{x} = [1, x_1, \ldots, x_d]^T$. A visualization of this structure is found by only studying the gray colored elements of Figure 2.1. A single perceptron, like the one from Figure 2.1, and given optimized weights, is able to separate two classes, by

applying for instance the sign function to its output, then choosing a class based on the sign of the output. Extending this idea to multiple classes, by using multiple perceptrons in parallel,

$$y_i = \boldsymbol{w}_i^T \boldsymbol{x} \tag{2.2}$$

where each perceptron is receiving the same input, but weighting them differently. The weights determine the information flow by gating the inputs. Now, to use this layer of multiple perceptrons in classification, we would simply assign the predicted class to be the output $\max_k y_i$. In addition to say something about the posterior probabilities, we would apply the *softmax* function in Equation 2.3 [32].

$$y_i = \text{softmax}(\boldsymbol{w}_i^T \boldsymbol{x}) = \frac{\exp \boldsymbol{w}_i^T \boldsymbol{x}}{\sum_k \exp \boldsymbol{w}_k^T \boldsymbol{x}} \tag{2.3}$$

From Equation 2.3 we see how the output of a perceptron is scaled by the outputs of the other perceptrons at the same level.

2.2 Learning perceptrons weights

To find the optimal weights for the perceptron, we use an iterative approach of incremental adjustment. We can imagine the hyperplane defined by the perceptron trying to separate K classes from initial random weights, being completely random. By quantifying how far from correctly labeling the input samples the perceptron were over several attempts, we are able to say something about the direction the weights will need to be adjusted to improve the classification accuracy.

The loss function is a statistical measure of closeness between a prediction from our model and the true value from the dataset we are attempting to model. There are not really any strict rules for what a loss function needs to look like, as long as it quantifies the error of predictions. Based on the problem, it is common to employ two different types of loss functions. If the problem deals with aligning to continuous true quantities by modelling a linear relationship between a dependent variable $\boldsymbol{y}$ from independent variables $\boldsymbol{x}_i$, the problem is a regression problem, and benefits from a loss function tailored for the task. The most common loss function used in regression is the Mean Squared Error (MSE),

$$\text{MSE}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{n}\sum_{i}(\boldsymbol{y} - \hat{\boldsymbol{y}})^2, \tag{2.4}$$

where $\boldsymbol{y}$ is the observed, true values, and $\hat{\boldsymbol{y}}_i$ is the predicted values. On the other hand, if we are predicting the class label, MSE will do a poor job of capturing the inter-class disparity compared with a suited function such as Cross Entropy (CE), which we can observe from the scalar valued output of Equation 2.4. Computing the CE between the true class ($\boldsymbol{y}^t$) of a sample ($\boldsymbol{x}^t, \boldsymbol{y}^t$), and the predicted value, $\hat{\boldsymbol{y}}^t$ gives us a measure of dissimilarity between the predicted and true class. When we are dealing with multiple classes, we encode the true class as $y_i^t = 1$ if the sample belongs to class i and 0 otherwise. The cross-entropy is computed as,

$$\text{CE}(\boldsymbol{x}^t, \boldsymbol{y}^t) = -\sum_{i} y_i^t \log \hat{y}_i^t. \tag{2.5}$$

Binary predictions use a modified version of the CE loss function, Binary Cross Entropy (BCE). While inputs to the CE function are *softmax* activations (Equation 2.3), inputs to a BCE function come from the two class special case of the softmax function, the sigmoid function,

$$\sigma(z) = \frac{1}{1 - e^{-z}}, \tag{2.6}$$

where, z for instance, to match our previous example would be $z = \boldsymbol{w}_i^t \boldsymbol{x}$. The BCE function would receive $\sigma(z)$ as input and is defined as,

$$\text{BCE}(\boldsymbol{x}^t, \boldsymbol{y}^t) = -[y^t \log \hat{\boldsymbol{y}}^t + (1 - \boldsymbol{y}^t) \log 1 - \hat{\boldsymbol{y}^t}]. \tag{2.7}$$

For our perceptron to be able to learn, we need a couple of elementary building blocks. Loss functions are one of these building blocks which we need to use in combination with the other concepts we will discuss in the remaining parts of this chapter. We will now discuss a strategy to optimize our perceptron parameters.

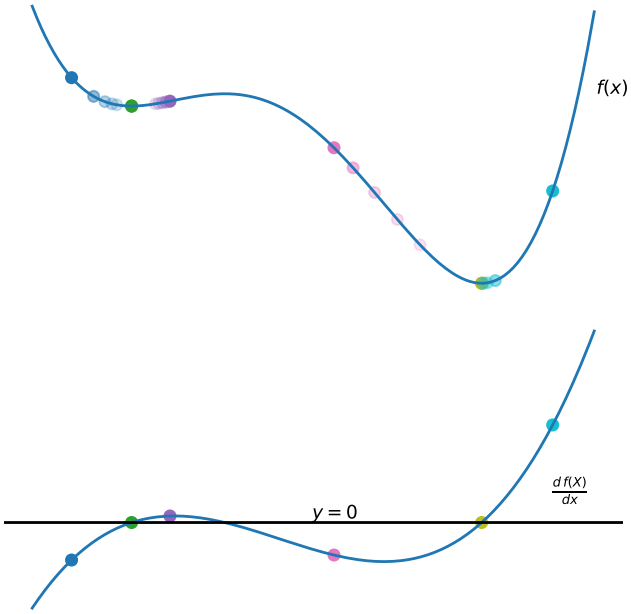

Figure 2.2: Visualization of gradient descent algorithm, for finding local minima. The top curve could for instance define a loss-space, whereas the bottom curve is its derivative. Similarly, colored points along the curve represent realizations. By applying a gradient descent updating algorithm, the position of the point move towards a local minimum. Movement illustrated by reducing the opacity of the similarly colored point.

2.3 Optimizers

Gradient descent is an algorithm used in optimizing parameters to a complex and often intractable function. We use gradient descent to optimize the parameters of a NN to the *loss space*. The ultimate goal is to navigate this loss space for a differentiable loss function, sometimes also termed *cost function*. We specifically describe gradient descent in this section, however the problem may also be formulated as a maximization problem, requiring gradient *ascent*, instead of our current minimization objective. The loss space is often highly complex as its topology is determined by the loss function used, dataset, model's parameters, number of perceptrons used, etc. To optimize the model's parameters, gradient descent formulates a way to minimize the objective function, which we demonstrate: say we have some loss function f, and would like to optimize our weights such that with the output from the model $\hat{\boldsymbol{y}}$, and ground truth $\boldsymbol{y}$ we find the global minima of f, i.e.,

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} f(\hat{\boldsymbol{y}}, \boldsymbol{y} \mid \boldsymbol{w}), \tag{2.8}$$

where $\boldsymbol{w}^*$ represent the optimal parameters. In practice, this is done using the update rule,

$$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n - \eta \frac{\partial f(\hat{\boldsymbol{y}}, \boldsymbol{y} \mid \boldsymbol{w}_n)}{\partial \boldsymbol{w}_n}, \tag{2.9}$$

using $n = 1, \ldots, N$ update steps. The hyperparameter η, in practice referred to as the learning rate, determines how to scale the gradient ∂f each step length per update iteration. An important remark to make is that we use the negative gradient as this means to update by taking a step *downhill*, since the gradient of f represents the direction of steepest increase of f. A subtle but important distinction to make is that gradient descent computes gradients of the whole dataset at once. A useful variation is called *stochastic gradient descent*, and updates the weights once per sample or over a randomly selected subset, or *batch* in the dataset. This leads to stochastic gradient descent acting as an estimate of gradient descent. Stochastic gradient descent is less computationally expensive in exchange for a lower rate of convergence. Every time the update algorithm has seen all the samples in the dataset, we say that it has reached one *epoch*, hence, gradient descent updates one step once per epoch, while for stochastic gradient descent it would depend on the batch size. Both algorithms are examples of *optimizers*, a category of algorithms used in Machine Learning (ML) to find the optimal network parameters. To get an idea of the motion produced by the optimizer, Figure 2.2 displays an example where the gradient descent algorithm is applied. For each point along the top curve, we update their position by taking a step in the opposite direction of the derivative, bottom curve, of their current position. This way, we can guarantee to always move downwards, until this is no longer possible. For instance, if a point reaches a saddle point, or local minima, where the derivative is zero, the algorithm would be stuck. For optimal training of network parameters, however, we are concerned with finding the global minima.

2.3.1 Momentum

Momentum is a modification introduced to circumvent converging to bad local minima or saddle points. To demonstrate, we first introduce an analogy for gradient descent where a person walking down a large mountain, always following the steepest path of descent, one step at a time, but with limited visibility, therefore the person is never sure when he has reached the foot of

the mountain. The analogy explains stochastic gradient descent by capturing its limitations. However, now consider a heavy boulder rolling down the same mountain. The person walking would have stopped on the saddle point of the mountain ridge between two peaks. The boulder is however able to build up momentum and moves past small troughs and ridges. Likewise, by remembering the change in weights between each update step, [33] demonstrated the use of an acceleration term, we provide an updated version of their forumlation as,

$$\begin{aligned} \boldsymbol{m}_n &= \beta m_{n-1} + (1-\beta)\frac{\partial f(\hat{\boldsymbol{y}}, \boldsymbol{y} \mid \boldsymbol{w}_n)}{\partial \boldsymbol{w}_n} \\ \boldsymbol{w}_n &= \boldsymbol{w}_{n-1} - \eta \boldsymbol{m}_n \end{aligned} \tag{2.10}$$

Their motivation was to develop an optimizer which had the convergence speed of an algorithm using the second derivative, but simpler to compute and implement for parallel processing. Momentum alleviates both problems with getting stuck in bad local minima and helps speed up convergence, besides, with momentum, the risk in using larger learning rates are fewer.

2.3.2 Adam Optimizer

With the already discussed optimizers, the issue of sensitivity to learning rate, and the slow convergence rates remain. Methods such as *AdaGrad* [34], *RMSprop* [35] and *Adam* [36] tackle this by adaptively adjusting the learning rate. In their paper [34], they emphasize difficulties for infrequent, but highly informative features, that due to their rarity, are weighted less. They devise a method which adapts the learning rate for each weight component independently, giving infrequently occurring features high learning rates, in practice, forcing the learner to notice once the rare feature occur [34]. RMSprop can be viewed as an extension of AdaGrad, which keeps a moving average, exponential decay, of the squared gradients instead of accumulating it during training. In practice, it decays the learning rate, and emphasize more recent gradient values.

Adam adapts the learning rate for each parameter, based on past gradients magnitudes, accumulating squared gradients for each parameter, subsequently reducing problems with sparse gradients. From RMSprop it borrows the idea of exponential decay of squared gradient, which prevents the learning rates from becoming too small. The algorithm is defined as,

$$
\begin{aligned}
\boldsymbol{w}_n &= \boldsymbol{w}_{n-1} - \frac{\eta \hat{m}_n}{\sqrt{\hat{v}_n} + \epsilon} \frac{\partial f(\hat{\boldsymbol{y}}, \boldsymbol{y} \mid \boldsymbol{w}_n)}{\partial \boldsymbol{w}_n} \\
\hat{m}_n &= \frac{m_n}{1 - \beta_1^n} \\
\hat{v}_n &= \frac{v_n}{1 - \beta_2^n} \\
m_n &= \beta_1 m_{n-1} + (1 - \beta_1) \frac{\partial f(\hat{\boldsymbol{y}}, \boldsymbol{y} \mid \boldsymbol{w}_n)}{\partial \boldsymbol{w}_n} \\
v_n &= \beta_2 v_{n-1} + (1 - \beta_2) \left(\frac{\partial f(\hat{\boldsymbol{y}}, \boldsymbol{y} \mid \boldsymbol{w}_n)}{\partial \boldsymbol{w}_n} \right)^2
\end{aligned}
\tag{2.11}
$$

Where m and v represent the first and second moment estimates, respectively. While $\hat{m}$ and $\hat{v}$ is introduced as bias correction that contract initialization of moment estimates, providing more stable initial steps of training. In practice, it is widely used and perform well for most datasets.

We will now look at more complex configurations of perceptrons, where we stack them into layers, to create a NN.

2.4 Multi Layer Perceptrons

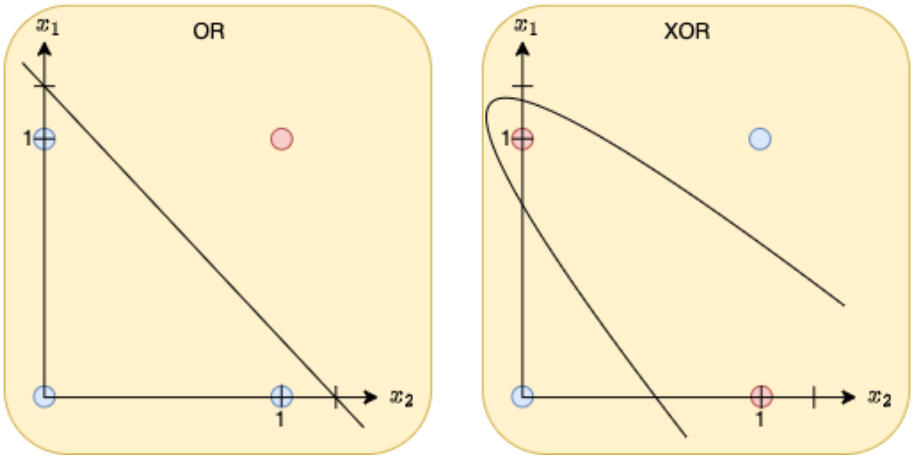

Figure 2.3: Example of two-class problem of OR and XOR. The OR function can be discriminated using a linear classifier, however the XOR problem needs a non-linear discriminant to correctly classify the points into their respective blue and red categories.

When stacking multiple perceptrons in series, interconnecting each neuron,

we construct a MLP. The MLP is useful in that perceptrons by themselves are not able to solve classification problems like the XOR problem depicted in Figure 2.3, where the discriminant needs a non-linear function to separate the blue and red dots. The stacked perceptrons between the input and final output layers in a MLP are referred to as hidden layers, and these allow the MLP to produce non-linear discrimination functions. Mathematically, we may formulate it as,

$$\begin{aligned} \boldsymbol{z}_1 &= \sigma(\boldsymbol{w}_1^T \boldsymbol{x}), \\ &\vdots \\ \boldsymbol{z}_i &= \sigma(\boldsymbol{w}_i^T \boldsymbol{z}_{i-1}), \\ &\vdots \\ \boldsymbol{y} &= \sigma(\boldsymbol{w}_L^T \boldsymbol{z}_{L-1}), \end{aligned} \tag{2.12}$$

for a network of L layers. The amount of hidden layers in the network is referred to as its depth. In practice, for a Deep Neural Networks (DNNs) it is more common to discuss the number of parameters, i.e., the amount of adjustable parameters captured in the weight matrices of the networks. MLPs are useful in that they, from the *universal approximation theorem*, are able to approximate any continuous function, given the appropriate weights [32]. This means that they have the potential to model any function to an arbitrary degree of accuracy, so long it contains enough neurons in the hidden layer.

2.5 Backpropagation

We have previously discussed how a simple perceptron can be trained using gradient descent. Based on its closeness to the ground truth, it receives a gradient update. However, MLPs have stacked layers with nonlinearities between them

While a simple perceptron can be trained using gradient descent by directly updating its weights based on the difference between its output and the ground truth, MLPs present a more complex challenge. With stacked layers and nonlinear functions, MLPs introduce a hierarchy of transformations that obscure the direct relationship between the input and the output. This mean we need a systematic approach to compute gradients of the loss function with respect to the network parameters. For this task, the backpropagation algorithm is able to efficiently propagate the gradients backwards through the network, enabling effective training of MLPs and NNs by iteratively adjusting

their weights to minimize the predition error.

We now introduce the notion of forward and backward passes. During an iteration of training, the forward pass is the part of the algorithm where we apply the model to a input $\boldsymbol{x}$, as in Equation 2.12, and produce our prediction $\boldsymbol{y}$. We then use the loss function to compare our prediction, with the ground truth $\hat{\boldsymbol{y}}$. Now we are ready to do the backwards pass. To do this, we need to compute the gradients. These represent the sensitivity of the loss function to changes in the model parameters, and are essential for guiding the optimization process. We compute the gradient of the loss with respect to the output by,

$$\delta_L = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}} \odot \sigma'(\boldsymbol{W}_L^T \boldsymbol{z}_{L-1}), \tag{2.13}$$

where $\odot$ is the *Hadamard* product, that performs element-wise multiplication, and $\sigma'(\cdot)$ is the derivative of the activation function. For each hidden layer $i = L-1, L-2, \ldots, 1$, we compute the gradient of the loss with respect to the output of the previous layer, and use it to compute the gradient with respect to the weights,

$$\delta_i = (\boldsymbol{W}_{i+1}\delta_{i+1}) \odot \sigma'(\boldsymbol{W}_i^T \boldsymbol{z}_{i-1}). \tag{2.14}$$

After computing the gradients, the next step is to update the parameters of the network accordingly using a chosen optimization algorithm, discussed in Section 2.3. For instance, using stochastic gradient descent to update each parameter:

$$\Delta \boldsymbol{W}_i = \eta \delta_i \boldsymbol{z}_{i-1}^T$$
$$\Delta \boldsymbol{b}_i = \eta \delta_i$$

While it took some time to arrive at a functioning MLP that is able to learn patterns and adapt to data, we will now investigate problems related to gradient flow in NNs.

2.6 Activation Functions

The choice of activation function in a NN has significant impacts on its performance. Activation functions introduce non-linearity into the network, which

is what enables it to model complex relationships in the data. Looking back at Equation 2.12, if we were to remove the nonlinearity σ, the entire network would collapse into,

$$\boldsymbol{y} = \boldsymbol{W}_L^T Z_{L-1} = \boldsymbol{W}_L^T \cdot \boldsymbol{W}_{L-1}^T \boldsymbol{z}_{L-2} = \cdots = \boldsymbol{W}_L^T \cdot \boldsymbol{W}_{L-1}^T \cdots \boldsymbol{W}_1^T,$$

which of course is a *linear* function, and is only able to model linear relationships in the data. Furthermore, different tasks may benefit from different activation functions, and understanding them can help tailor the NN to specific applications. Some commonly used activation functions, are the sigmoid function, which was introduced earlier in Equation 2.6, which is commonly used in binary classification tasks, since it squashes input values to a range between 0 and 1. The hyperbolic tangent (tanh),

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \tag{2.15}$$

is similar to the sigmoid, but squashes its input to the range of -1 to 1, and can be preferred used in hidden layers because of its zero-centered outputs, meaning that during backpropagation, the updates to the weights are not biased towards any particular direction. Another very common choice is the Rectified Linear Unit (ReLU) activation function,

$$\text{ReLU}(x) = \max(0, x), \tag{2.16}$$

or the Leaky ReLU,

$$\text{Leaky ReLU}(x) = \begin{cases} x, & \text{if } x > 0, \\ \alpha x, & \text{if } x \leq 0. \end{cases} \tag{2.17}$$

ReLU is a common choice not only for its simplicity, but it helps mitigate a more serious problem in DNN, namely the problem of vanishing gradients.

Vanishing and Exploding Gradients. During training with backpropagation, we recall Equation 2.14, which computes gradients using the derivative of the activation function. When the derivative of the activation function $\sigma'(\cdot)$ is less than 1, the gradients can diminish exponentially as they propagate through each layer. If we consider the chain of gradients from the output layer L to the input layer:

$$\begin{aligned}
\delta_L &= \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}} \odot \sigma'(\boldsymbol{W}_L^T \boldsymbol{z}_{L-1}), \\
\delta_{L-1} &= (\boldsymbol{W}_L \delta_L) \odot \sigma'(\boldsymbol{W}_{L-1}^T \boldsymbol{z}_{L-2}), \\
\delta_{L-2} &= (\boldsymbol{W}_{L-1} \delta_{L-1}) \odot \sigma'(\boldsymbol{W}_{L-2}^T \boldsymbol{z}_{L-3}), \\
&\vdots \\
\delta_1 &= (\boldsymbol{W}_2 \delta_2) \odot \sigma'(\boldsymbol{W}_1^T \boldsymbol{x}),
\end{aligned} \tag{2.18}$$

If $\sigma'(\cdot)$ is less than 1, each multiplication in the chain reduces the gradient's magnitude exponentially with the number of layers, hence the gradients become very small in the earlier layers, which leads to minimal updates to the weights, which slows or stalls the learning process. In the opposite case, when the derivative of the activation function is greater than 1, the gradients can grow exponentially as they propagate backward through each layer. This can cause unstable weight updates during training, and cause the model to diverge from its optimization goals.

2.7 Regularization

For NNs, regularization is a set of techniques that aims to improve the NNs ability to generalize, which is its ability to perform well on new and unseen data. DNNs have a high capacity to learn complex patters from data, however this also makes them prone to overfitting, which is the term used when the model learns not only the underlying pattern but also noise and outliers in the data. We see overfitting when the network performs exceptionally well on the training data, but poorly on unseen data [37].

Broadly speaking, we may say that in regularization trades performance on the training data for the test data. These techniques are essential in modern DNNs by enforcing constraints to the learning process, to ensure the model learn the most relevant features and ignores noise.

2.7.1 Common Regularization Techniques

While there are many different ways to constrain the training process to encourage generalization, we choose to highlight three common ones.

L1 and L2 Regularization. By adding a penalty term to the loss function, proportional to the absolute value of the model parameters, we discourage the

model from relying too much on any single parameter. This is L1 regularization. If we instead penalize using the square of the model parameters we have L2 regularization. Given some cost function, we may write these as,

$$\begin{aligned}\tilde{\mathcal{L}} &= \mathcal{L} + \lambda \sum_i |w_i|, \\ \tilde{\mathcal{L}} &= \mathcal{L} + \lambda \sum_i |w_i^2|.\end{aligned} \tag{2.19}$$

For L1 and L2 regularization, respectively.

Data Augmentation is the process of applying a set of random transformations, for instance, rotations, translations, or flips. This helps the model to learn features that are invariant to the used transformations and improve the robustness of the model. Augmentation of the dataset is an efficient way to produce more data, because the best way to improve the performance of DL models, is to train it on more data [37].

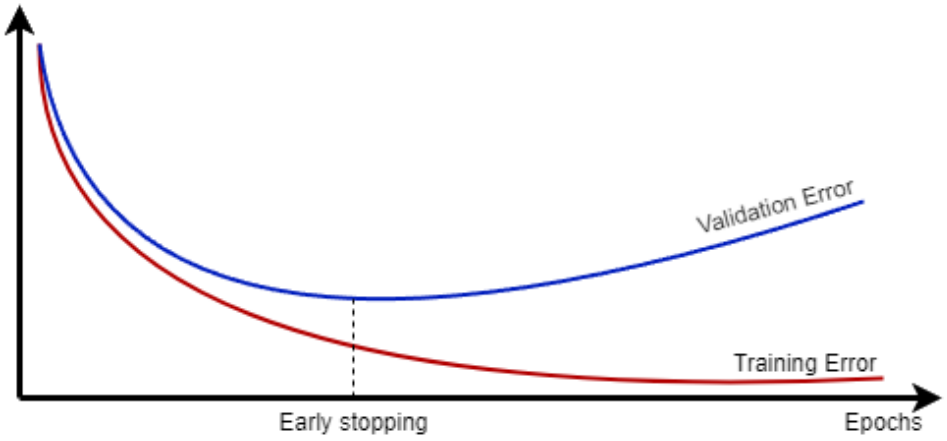

Figure 2.4: Example illustration of early stopping as a regularization technique by using a measuring performance on a validation dataset.

Early Stopping is the technique where we measure the performance of the model during training on a validation dataset that the model has not seen. When the validation error diverges, while the training error continues to converge, as seen in figure Figure 2.4, we discontinue training. This way we may directly intervene when the model overfits to the data.

2.7.2 Dropout

An effective way to prevent overfitting during training is through the use of dropout. Introduced by [38, 39], dropout involves randomly "turning off" a subset of neurons during each training iteration. This means that for each forward and backward pass, some neurons are ignored, and only a subset of the network is used. This improves the NNs ability to generalize by preventing co-adaptations, where neurons rely on specific activations from other neurons to themselves activate for a given input. As an analogy we may think of dropout as an office where each task can be performed by several employees, but no single employee is indispensable. On any given day, a random subset of the employees might take a day off. The remaining employees must still ensure that all tasks are completed, forcing each employee to be versatile and capable of handling multiple roles, rather than relying on a specific colleague to always handle certain tasks. Similarly, in training a NN with dropout, we force the network to develop redundant representations of the data, ensuring that no neuron relies too much on another [38, 39].

Dropout may be viewed as having trained an ensemble of many distinct networks with shared weights. During training, each forward pass can be thought of as sampling a different subnetwork [40]. However during inference, we approximate the effect of averaging the predictions of all possible subnetworks using "inverted dropout", where all neurons are used, but their outputs are scaled down by the dropout rate. The more accurate approximation of all possible subnetworks is by sampling a subset of NNs with dropout and averaging their predictions, however it is computationally expensive for representative sample sizes [39]. We will discuss how this more accurate MC variant can be leveraged for uncertainty estimation in Chapter 5.

3

Convolutional Neural Networks

In this chapter we discuss CNNs and DCNNs, from the basic convolutions and pooling to deep segmentation models. We begin Section 3.1 by defining the convolution operation, before discussing how a CNN is structured in Section 3.2. In Section 3.3 we present seminal CNN architectures before defining how CNNs are used in segmentation models.

3.1 Basics of Convolutions and Pooling

A convolution is a fundamental operation in CNNs, where a small matrix, a kernel or filter, slides over the input image to produce a feature map, as shown in Figure 3.1. Mathematically, the convolution involves element-wise multiplication and summation of the filter and the patch of the input image that it covers. In the context of NNs, we use this operation to allow the network to detect features such as edges, textures, and patterns in the input image. By using different filters, which in fact store the learnable weights of the CNN, the network can learn to detect various types of features. As the network goes deeper, the convolutions capture more abstract and complex patterns.

The idea of convolutions stems from signal processing, where a one-dimensional signal would be convolved with a filter to extract certain properties about the

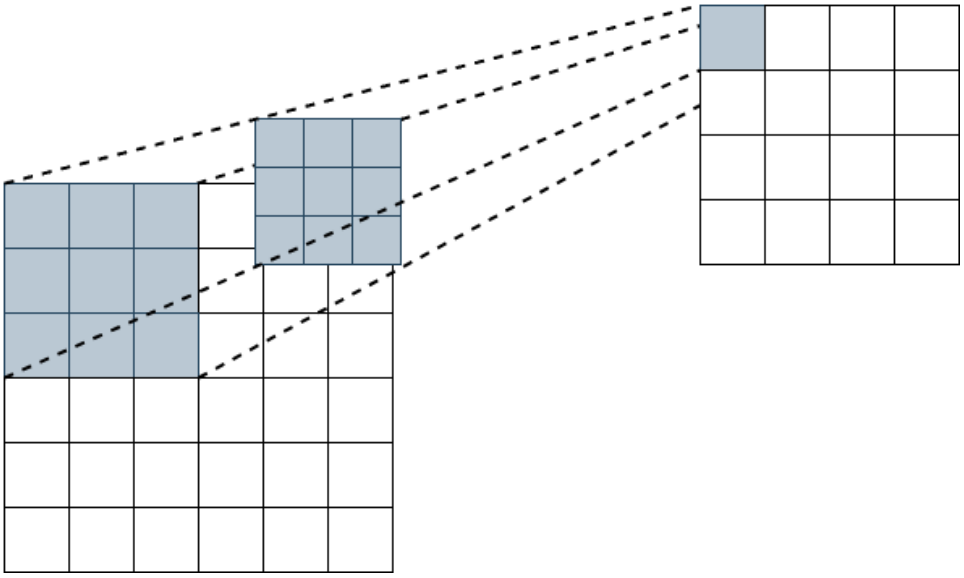

Figure 3.1: Illustration of an 6×6 image being convolved with a 3×3 kernel to produce a 4×4 feature map. The kernel slides across the image, filling in the values of the feature map iteratively.

signal. For instance, if we use a filter $\boldsymbol{h}$, with entries, $[1/3, 1/3, 1/3]$, then apply this to a signal, we have created a *moving average* filter, that averages over values of the input signal. Likewise, if we extend the idea of a average filter to say, a 3×3 kernel, then slide it over an image, we effectively blur together a 3×3 area of the input image, in our output. The particular filter is known as a *box blur* filter.

Padding. On inspection of Figure 3.1 we see that the output feature map is downsampled from a 6×6 input size to a 4×4 output size. This is because we made a choice of starting the sliding of the kernel, to where it fit perfectly within the input image. While this works for downsampling, CNNs tend to add zeros, averages, or a clone of the value of the boundary pixel along all edges of the image, such that the kernel is able to have its center aligned with the proper edge of the input. This process is referred to as *padding*, and if we pad the input to where the center of the kernel aligns with the image, and the kernel only slides one pixel at a time, we preserve the size of the input. However, padding the input can produce boundary artifacts, where if we use for example *zero-pad* the image, the kernel will multiply and sum lower values along the borders.

Stride. Another method that affects the size of the feature output map is stride. Like a person walking, stride refers to the walking motion of the kernel across the image. The common default setting is a stride of one, where the

kernel moves itself by one pixel per iteration of the convolution. If the stride is increased to two, the kernel would walk two pixels, for the toy example in Figure 3.1, the produced feature map would end up with a shape of 3×3. The output shape of the feature map for a convolution with stride zero, is left as an exercise for the reader.

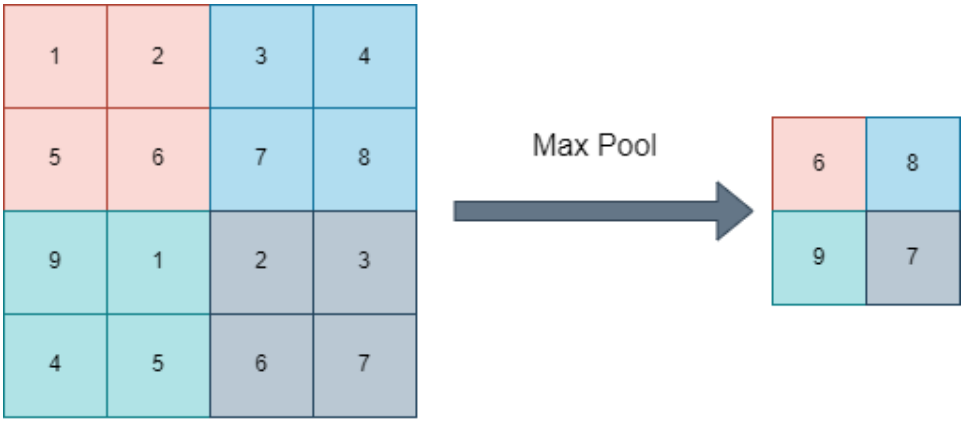

Figure 3.2: Illustration of Max Pooling using a 2×2 kernel with stride 2. The maximum values of each interaction between the input and kernel are carried on.

Pooling is a technique used to reduce the spatial dimensions of the feature maps, thus decreasing the number of parameters, and computational load in the network. The two most common types of pooling are max pooling, and average pooling. Max pooling takes the maximum value from each patch of its input, while average pooling calculates the average value. When used in CNNs, pooling operations are usually placed right after a convolutional. In addition to saving computations, they provide approximate invariance to small translations, making the network less sensitive to the exact location of features, and help in controlling overfitting by its parameter reduction, which simplifies the model [37].

3.2 Architecture Components

CNNs are typically made up of convolutional layers, which in turn, are made up of multiple filters that apply convolutions to the input, generating several feature maps. When we apply multiple filters to the same image, each filter captures different aspects of the image, where each filter in a stack are referred to as the channels of the convolutional layer. Thus, we end up with as many feature maps - different representations of the input - as there are channels in the convolutional layer. In the convolutional layer, activation functions typically follow the convolution, to introduce a non-linearity. A common choice is the

ReLU activation function. Pooling layers, which is the last step in convolutional layers, downsample the feature maps to reduce the dimensionality and computational complexity [37]. For example, a simple CNN architecture might include a sequence of convolutional layers, followed by a pooling layer. Many popular architectures will use this downsampling in conjunction with increasing the number of channels as the feature maps grows smaller, but deeper, through a series of convolutional and pooling layers. Hence, convolutional layers provide efficient ways of processing image data for NNs.

One of the key advantages of convolutional layers is parameter sharing. Unlike fully connected, classical MLP layers, where each neuron has its own weights, in convolutional layers, the same filter is used across different parts of the input. This reduces the number of parameters significantly, and allows the model to learn spatially invariant features [37]. Parameter sharing ensures that the model is more efficient and less prone to overfitting, making it well suited for grid-like data, such as images, where the same patterns can be found in different locations. CNNs also exploit the inherent spatial hierarchy in images through their use of multiple convolutional layers. Early layers typically capture low-level features such as edges and textures, while deeper layers capture high-level features such as shapes and objects. This hierarchical feature learning is important for building complex representations needed for tasks like classification and segmentation, which we will see later. By reducing the spatial dimensions and increasing the depth of the feature maps, CNNs can effectively capture spatial relationships and hierarchical structures of its inputs.

3.2.1 Backpropagation

In general terms, for CNNs, the backpropagation algorithm from Section 2.5, is modified to account for the convolution operation. Unlike Fully Connected Neural Networks (FCNNs) where each weight is updated individually, the CNN update the parameters of the filters used in the convolutional layers. A challenge in backpropagating gradients through convolutional layers, is how to compute the gradients with respect to the filter weights. The equivalent chain rule gradient computation is done by sliding the filters over the input data, and computing gradients for each position, using the *transposed convolution*, or *deconvolution*. This operation is essentially the reverse of the convolution operation used in the forward pass. But instead of applying the filters to the input data, transposed convolution involves applying the gradients to the output feature maps to compute gradients for the filter weights, allowing the CNN to update the filter weights correctly.

3.3 Deep Learning Architectures in Computer Vision

Common early CNN architectures such as *LeNet* [41], and *AlexNet* [42] demonstrate the practical use of these layers in image recognition tasks. LeNet, is one of the earliest CNNs, and used a combination of convolutional layers consisting of convolutions, the *tanh* activation function and average pooling layers, that were followed by fully connected layers, developed for recognizing handwritten digits [41]. AlexNet, won the ImageNet image classification competition in 2012, and demonstrated the power of stacking more layers, making the network deeper. They used smaller kernels than LeNet, which had 5×5 kernels, by instead using 3×3 kernels, followed by max pooling layers [42]. Both architectures have significantly advanced the field of DL in computer vision tasks.

3.3.1 Deeper architectures

Deeper networks have the potential to learn more complex representations, and achieve higher performance [43, 44, 45]. However, the *degradation problem* addressed by [43], is the counterintuitive observation that by adding layers, the network performs worse. They outline this by pointing out that the problem does not stem from overfitting, as the training error increased, suggesting that the model's capacity to learn diminishes beyond a certain point. The degradation problem indicates that the optimizer struggles to find the optimal weights as the network gets deeper, even though the *expressivity* of the network should be higher.

To address this problem, they introduce shortcut connections that bypass one or more layers. The shortcuts allow the network to learn residual functions, which can be described as the differences between the desired output and the input to a layer, rather than directly trying to learn the entire transformation. Their reasoning is based on the hypothesis that it is easier for a DCNN to learn the residual mapping than the original. The shortcut connection also improves the flow of gradients through the network, which mitigates issues related to vanishing gradients. They present the *ResNet* framework, which pushed the possible depths in networks significantly.

3.4 Methods for Semantic Image Segmentation in Deep Learning

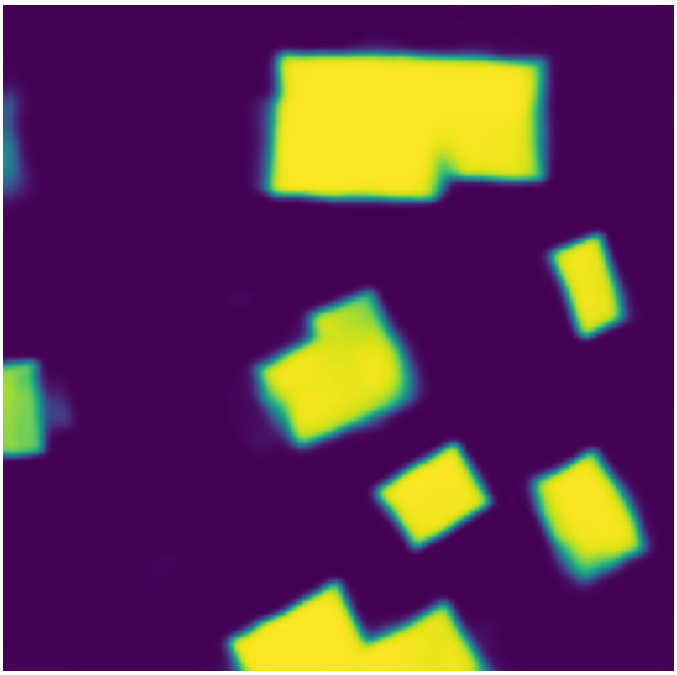

Figure 3.3: Example of semantic segmentation mask of a building.

Semantic image segmentation is a fundamental task in computer vision, that involves partitioning an image into multiple segments or regions based on certain criteria. Unlike object detection, which traditionally have focused on localization of objects with bounding boxes, semantic segmentation assigns a label to each pixel in an image, indicating the object or region that the pixel belongs to. Through pixel-wise classification, it allows for a greater understanding and analysis of the contents in the image. It is thus important in applications such as autonomous driving, by correctly detecting e.g., pedestrians or cyclists, medical image analysis, with the ability to efficiently analyze MRI data, and scene understanding. Semantic segmentation differs from instance segmentation, which in addition to classifying pixels, also distinguish between different instances of the same class.

Functionally, a semantic segmentation mask of a input image will classify each pixel to a predefined label. For instance, Figure 3.3 demonstrates a binary segmentation example of multiple buildings, where the brighter colored pixels are where the model is confident of a *foreground object*, while the model believes dark-colored pixels belong to the background. By applying a threshold, we get the class labels for each of the pixels. This general idea demonstrated on a

binary segmentation map, also extends to multiple classes, however the model then outputs its prediction mask with the same shape as the input, height × width, and the number of channels of the output tensor matches the number of classes.

3.4.1 Semantic segmentation Architectures

While there exist many DL architectures which are able to do semantic segmentation, we will in this section highlight three of the seminal ones that have pioneered the field. The final one is also used to extract features in our later experiments.

Fully Convolutional Network The architecture Fully Convolutional Network (FCN), proposed by [46] extended the abilities of regular CNNs for semantic segmentation by generating segmentation masks at the same image resolution as the input image. Similar to a CNN used for regular classification, the architecture use convolutional layers to create a dense representation of the data that capture spatial relationships and local patterns, but replaces the final fully connected layers from the dense representation with a upsampling procedure. The deconvolutions converts the low-resolution feature maps back to the original image size to produce pixel-level segmentation maps. They do this by using deconvolutions, which reverses the downsampling done by the earlier convolutional and pooling layers. The transposed convolution is upsampled by inserting zeros between the pixels of the dense feature map, before convolving with a filter, which upsamples the input. As the upsampled representations could be very coarse, they used skip connections to merge feature maps from earlier layers, during upsampling, which helps with preserving details by combining low-level spatial information with high-level semantic information.

U-Net The U-Net is a architecture that was originally developed for biomedical image segmentation by [47], but has since been widely adopted, due to its ability to produce precise and detailed segmentations. It is characterized by its U-shaped architecture, that consist of a contracting path, the encoder, and an expansive path, the decoder. This design allows the network to capture context and localization efficiently. The encoder is a usual CNN feature extractor, that creates a dense representation of the data, like the CNN used in the FCN. Also, similar to the FCN, the decoder, use deconvolutions to upsample the representations in the same stepwise manner as the encoder, in addition to receiving skip connections for every upsampling. This improved its localization ability, maintaining high-resolution information through the network. The detailed fusion of features helps in producing finer segmentation outputs with more

accurate boundaries compared with the simpler skip connections used in FCN. The architecture's ability to generalize from limited data made it well suited for use in the medical field, where access to detailed data is restricted.

High-Resolution Network (HRNet) Is a state-of-the-art framework designed for tasks that need high-resolution representations, proposed by [48]. They achieve this by maintaining a high-resolution feature representation through the network, unlike the encoder-decoder bottleneck architectures mentioned above, and others such as [49]. It is still able to capture low-level features by using multiple parallel branches, where each correspond to a different resolution. The branches interact and exchange information by using multi-scale fusion, where features from different resolutions are combined multiple times. This ensures that the high-resolution representation is enriched with representations from lower levels, while efficiently preserving spatial relations.

4

Existing Architectures for Building Footprint Extraction

We well now investigate the existing architectures that have been developed with the means of improving building footprint extraction. We begin this chapter by discussing the need for tailored model outputs, and a shift away from binary segmentation maps, here we will distinguish two main categories of solutions, before discussing a few selected models more thoroughly. The reader should pay extra attention to Section 4.4, which is later modified and evaluated in our experiments in Chapter 8.

4.1 Polygonal Extraction: A Shift From Segmentation

Extracting building footprints from overhead images is a complex and challenging task with significant implications for urban planning, agriculture, environmental monitoring, and disaster management. The goal is to relieve the manual labor involved, and provide up-to-date information about rapidly changing urban environments [10]. However, this task is complicated due to factors such

as occlusions from vegetation and nearby tall structures, varying shapes and sizes.

Traditionally, the task has been approached as a pixel-wise semantic segmentation problem [12], using established frameworks such as the U-Net [47], and HR-Net [48] to detect buildings. Instance segmentation has also been used in methods like the *Mask R-CNN* [50], where the network initially extract bounding boxes, before classification and producing a binary segmentation mask for each bounding box. Although these methods have shown promise, they often produce inaccurate building masks, with irregular boundaries [12, 17].

In recent years, there has been a shift towards extracting building footprints in a polygon, vector format. From the paradigm shift, the need for refined and regular representations of buildings became apparent. While segmentation models generally are able to maximize their accuracy, this does not translate to useful representations. By this, we mean that a binary mask output may quantitatively produce good accuracies, however on inspection of the predictions, walls are represented by irregular curves, due to low confidence along the borders of the masks. Subsequently, while there exist methods for generating polygonal representations from binary segmentation masks, for instance using contour detection by the marching squares algorithm [51], followed by DP simplification [52], these representations often produce highly complex polygons, with too many redundant vertices [17, 13, 15, 12].

The methods which output polygonal representations can be broadly categorized in to two approaches: end-to-end learned methods and hybrid methods.

4.2 End-to-End Learned Polygonal Extraction

Directly learning building vector representations from overhead images leverage the power of DL to perform the entire extraction process in a single, unified model. While this category is broad, the developed methods tend to build on architectures that process information using graphs or recurrent representations. For instance, a polygon consists of connected vertices, hence RNNs could be used in recurrently predicting vertices in a polygon, one by one. This is the procedure of *Polygon-RNN* [23], and *Polygon-RNN++* [24], which from a predicted bounding box use a CNN-RNN architecture to sequentially produce vertices of the object outline. *PolyMapper* is another CNN-RNN method which specifically is designed for building extraction, with the addition of road predictions. By viewing a polygon as an undirected graph where every vertex has at most two edges, it makes sense to employ GNNs, as is done in the *Curve-GCN* [25]. The

Curve-GCN, as the name suggests, use a Graph Convolutional Networks (GCNs) to simultaneously predict all vertices from an initial polygon generated using a CNN feature extractor. Their GCN iteratively offset each vertex into their final position. Interestingly, Polygon-RNN, Polygon-RNN++, and Curve-GCN are in addition to detection of buildings by themselves, also able to interactively, with human supervision, speed up the labeling process for creating accurate ground truth labels. When compared with more recent methods, they can be difficult to train and due to their recurrent nature, need multiple inference iterations. Additionally, none of the aforementioned architectures are able to correctly vectorize buildings with holes.

Recently, [15] proposed *PolyWorld*, which from a set of CNN-extracted vertices and visual feature descriptors - that associate embedded information into each vertex - before a Attentional Graph Neural Network (AGNN) predicts an equal set of vertex offsets and matching descriptors. The matching descriptors are then used in a optimal connection network that use the *Sinkhorn* algorithm [53], to produce a permutation matrix that connects vertices, forming polygons. As we will see for hybrid models, connecting accurately predicted vertices without knowing which vertex connects to another is a recurring issue, with creative solutions. PolyWorld [15] is at the time of writing, only surpassed by [17] in terms of performance, but is able to produce regular and simple polygons, but fail in certain cases where buildings have complex shapes [17].

4.3 Hybrid Methods

The other category, are characterized by needing multiple steps. A common paradigm is to initially predict binary segmentation masks and sometimes additional information, before vectorizing these. Some methods add an extra refinement process; refining vertex positions, or simplifying polygons by removing vertices [17]. These methods address the issues with inaccurate segmentation borders, by leveraging Multi-Task Learning (MTL), that simultaneously predict multiple tasks inherent to the problem, e.g., vertices, orientations, line segments. To regularize the contributions from the segmentation maps, that are known to have inherent issues. For instance, Frame Field Learning (FFL) by [13], predicts a *frame field* in addition to the segmentation mask, using what mathematically is a degree four vector field, where two of the vectors are constantly opposite. This has the effect of always having one of the vectors tangent for building edges, and both vectors tangent for corners, effectively constraining polygons to more rigid shapes, avoiding rounded corners. To combine the predicted frame field with their segmentation map, they devise a modified Active Contour Model (ACM) [54], termed the Active Skeleton Model (ASM), that modifies ACM to work with frame fields instead of contours [13].

Opposed to end-to-end learned methods, FFL is easier to train, and while the predictions are performed once, the ASM needs active optimization during inference, causing a speed penalty.

Another method that addresses the *mask reversibility* [17] in building extraction, that is, the issues caused by imprecise borders produced by segmentation maps, is proposed by [12]. Their method also leverage MTL, however they specifically segment building area and border as separate tasks, to force discrepancy between the area and border. Additionally, the model also outputs an edge orientation. A distinct selection module filters the three outputs of the model to create an initial polygon, later refined using their vertex refinement method, that using a ResNet backbone trained in conjunction with a Gated Graph Neural Network (GGNN), where the initial polygon positions are used in extracting corresponding "depth-wise" features of the backbone embedding, to embed visual descriptor for each vertex. These are then used by their GGNN to offset each vertex. While the addition of a GGNN and additional feature extractor during polygon refinement brings additional complexity during training, their approach in addressing the issue of mask reversibility by predicting the contour of the building is a clever way to improve the borders produced by conventional segmentation models. Additionally, the vertex selection network leverage the edge orientation prediction by discarding the vertex if its previous consecutive vertex have a significantly differing orientation, which provide an efficient method for accurately filtering redundant vertices. While their model is able to achieve competitive results, it also struggles with buildings that have numerous short edges, and severely occluded [12].

4.4 Hierarchical Supervision

In this section, we introduce the *Hierarchical Supervision* (HiSup) architecture by [17], which we will later modify to incorporate uncertainty into, as we outline in our method in Chapter 6.

Hierarchical Supervision attempts to leverage inherent information in overhead images, which they divide into three categories: low-level information of building corners, represented by vertices, mid-level edge information, that connect the vertices, and represents building walls, and high-level shape information gathered in a semantic binary map. Their motivation, like the aforementioned architectures, is dealing with imprecise border maps generated from the segmentation output, which they identify as a limiting factor for hybrid models [17].

4.4.1 Addressing Mask Reversibility

In addressing mask reversibility, they regularize each task of their MTL architecture by enabling *cross-level interactions* between these. The mid-level geometries use Attraction Field Maps (AFMs) from [55, 56], to learn the edge connections between the low-level vertices, and emphasize building edges for the high-level segmentation network through cross-level interactions. Thus, each learning task interacts with the mid-level geometry using Efficient Channel Attention (ECA) [57], which builds on the attention mechanism, where the model learns to use features from the mid-level F_{afm}, to weight channels in each of the other levels that the network should pay extra attention to. This enforces that each of the levels learn the same thing, but implicitly weighs the *opinion* of the middle level more, since the high and low level does not project their features onto it. The authors implement ECA by the equations,

$$\begin{aligned} F^{e}_{\text{seg}} &= \sigma(\text{C1D}(\text{GAP}(F_{\text{mid}} + F_{\text{seg}}))) \cdot F_{\text{seg}} + F_{\text{seg}}, \\ F^{e}_{\text{ver}} &= \sigma(\text{C1D}(\text{GAP}(F_{\text{mid}} + F_{\text{ver}}))) \cdot F_{\text{ver}} + F_{\text{ver}}. \end{aligned} \tag{4.1}$$

In Equation 4.1, F^{e}_{seg}, and F^{e}_{ver}, refer to the *enhanced* high, and low-level geometries, respectively, while $\text{C1D}(\cdot)$ is the 1-D convolution. $\text{GAP}(\cdot)$ refer to global average pooling.

4.4.2 Learning Line Segments

In accurately capturing line segments in polygons, [17] employ a tactic devised by [55, 56], in their work in Line Segment Detection (LSD). Which is an important topic, with applications in scene understanding, and 3D reconstruction, in addition to object detection. Their motivation stems from that LSD is hard to formulate in a way that can leverage the full potential of DCNNs [55].

They highlight the factual similarity between region representation and boundary contour representations, and leverage this by reformulating LSD as the equivalent region coloring problem. Thus allowing semantic segmentation models to simply predict AFMs. To introduce AFMs, we first investigate the region-partition map of an image, which assigns every pixel to only one line segment. Imagine this as coloring each pixel of an image in the same color if their closest line segments are the same. Now, for every pixel p, and its corresponding projection p^{*} onto its assigned line segment in the region-partition map, we define the 2D attraction vector for the pixels to be,

$$\mathcal{A}(p) = p^* - p. \tag{4.2}$$

Hence, the AFM stores the x and y coordinates of each attraction vector in a two channel image. While [55] goes on to demonstrate how to collapse this representation back into a line segment, the authors of [17] found that by predicting the AFM, thus discarding the next step of LSD, worked best. HiSup further use the AFM for their mid-level geometrical features, and inflicting its understanding of the scene to the other levels, effectively regularizing them in accordance with itself.

4.4.3 Learning Hierarchical Representations

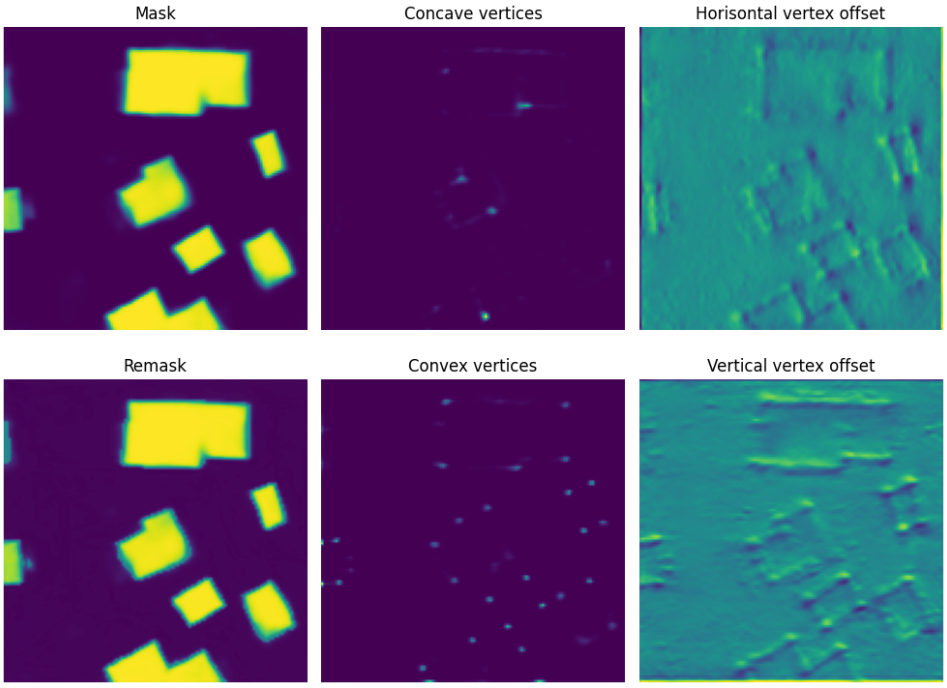

Figure 4.1: Example output from *Hierarchical Supervision* [17]. The left column displays the two segmentation masks predicted. Where the segmentation mask on top highlighted as *Mask* is the pure segmentation prediction from the model, which is discarded during polygon generation. *Remask* is the segmentation mask produced from the AFM intermediate prediction, and is kept for generation of polygons. The center column contain predicted vertices. The model distinguishes concave from convex vertices. The final column shows the vertex offset maps, which are also distinguished into horizontal and vertical offsets. The polygon generation MAV-ATTR uses a combination of these maps to produce polygons.

The HiSup framework train three simultaneous tasks, two of which produce segmentation maps. While this may come across as confusing, during inference, they discard the task which only focus on segmenting the image into a binary map, by instead constructing a new feature representation from the predicted AFMs, effectively only using AFMs as an intermediate stage. This new feature representation is concatenated with the initial feature map extracted from the backbone before predicting its own segmentation map, which they argue, will enhance building contours. To align this segmentation mask with the original one, which is discarded during inference, the *mask* loss of their model is the sum of the BCE loss from each predicted segmentation map. While we question the discarding of the original segmentation map, their ablation study demonstrates that it produces better building contours [17]. From the ground truth annotations, we may compute the true AFM, which during training is compared with the predicted AFM using an l_1 loss function.

To learn the vertex representation from the feature map enhanced using ECA with the mid-level geometries, the network predicts a convex and concave vertex heatmap, which is compared with the ground truth vertex positions using CE. They use a offset map extracted directly from the backbone, with two channels, one for each direction - horizontal, and vertical - which is multiplied with the ground truth vertex heatmap and compared with the product of the true offset and true vertex heatmap using l_1 loss. This provides an effective method to achieve sub-pixel accuracies, because the vertex heatmap only extracts pixel position of vertices. The different *final* outputs of the model can be viewed in Figure 4.1.

4.4.4 Polygon extraction

To extract vertices from the vertex and offset maps, the authors propose a polygon extraction and simplification routine, which they name Mask-and-Vertices Attraction (MAV-ATTR). This process consist of an initialization and simplification process, where the initial vertices come from the predicted heatmap $\boldsymbol{H}$ with the offsets of the vertices $\boldsymbol{O}$, and the predicted building mask $\boldsymbol{S}$. The initial regions, are extracted by applying a threshold $\tau_v = 0.008$, followed by Non Maximum Suppression (NMS), and a gathering of the extracted regions, from the vertex maps, during this process the offsets are also used to translate the vertices. We denote the set of refined vertices V.

Given the above described initial polygon representation, the authors have a 4-step simplification process, as the initial polygon is simply the contour of the segmentation map, with many redundant intermediate vertices. The process is described as,

1. Matching each vertex x_k^i, for the k-th vertex of the i-th initial polygon to the closest vertex in V, using the Euclidean distance,

$$\pi(k,i) = \arg_j \min_{\boldsymbol{v} \in V} \|\boldsymbol{x}_k^i - \boldsymbol{v}_j\|_2, \tag{4.3}$$

   where $\pi(k,i)$ maps indices from the boundary pixels of the initial polygon to the closest vertex in V.

2. Then finding all non-minimal pixels with unique indices, hence only storing the pixels with the minimal distance to a vertex, for the pixels that share a vertex.

3. The remaining pixels are removed if their distance to a vertex is greater than a threshold of $\tau_d = 5$.

4. Adjacent edges of polygons that are parallel to a threshold of 10° are merged.

In summary, the work presented in [17] demonstrates a capable architecture which leverages multiple levels of input geometries to better predict separate tasks in a MTL setting. Learning multiple levels of geometries tailored for the task's specific nuances, has been previously explored in different works [13, 12]. However, engineering sensible features which better the accuracy of the network is far from trivial. Their approach has demonstrated higher accuracies than previous works, while being able to retain simple polygons with acceptable numbers of redundant vertices used in the final polygon. We build further on their ideas in later chapters, where we leverage uncertainty to further address consistent building shapes, and a more robust architecture. We also modify the MAV-ATTR polygon generation method described above by guiding the fixed threshold τ_d using uncertainties.

5

Uncertainty Estimation in Deep Learning

This chapter will present and discuss the theory of uncertainty estimation in DL. We will indulge in concepts justifying the use of uncertainty estimation and implementation choices which will later be used in our own methodology presented in Chapter 6. First, we introduce uncertainty in deep learning, why it is of importance in general, and for our subgenre of building detection in Section 5.1. Second, we present Bayesian approaches to uncertainty in Section 5.2, before Section 5.3 elaborates on how dropout is used in Bayesian approximation.

5.1 Uncertainty in Deep Learning

Understanding and quantifying uncertainty in DL models is essential for accurate building and object detection. By evaluating model uncertainty, we can significantly enhance decision-making, ensuring both reliable and precise delineations. This is particularly crucial in areas like urban planning and disaster management, where incorrect delineation of building boundaries can have severe consequences [10]. Uncertainty estimation helps identify areas where the model may be prone to errors, allowing for targeted improvements and increased robustness against outliers. It also enhances model reliability

by indicating when predictions can be trusted and when further validation is necessary. Moreover, incorporating uncertainty measures makes our systems more robust, allowing them to handle diverse and complex environments with greater confidence.

Epistemic and Aleatoric uncertainties. Imagine you are baking a cake for a party, but you can't quite remember if you added the sugar. There is a fundamental difference between not knowing because you simply forgot - a knowledge problem - and not knowing because you're unsure how the cake will rise - a randomness problem. Similarly, in philosophy, the notions of *epistemic* and *aleatoric* uncertainty distinguish between our uncertainty due to a lack of knowledge, which we could theoretically resolve with enough information, termed epistemic uncertainty, and uncertainty due to inherent randomness in outcomes, termed aleatoric uncertainty [58]. As is highlighted by [29], in our field aleatoric uncertainty refers to the inherent noise in the data, while epistemic refers to the uncertainty in the model parameters. In understanding why a DNN makes one prediction over another, we need methods in modelling the epistemic uncertainty of our models.

5.2 Bayesian Approaches to Uncertainty

Bayesian methods offer a well established framework for quantifying uncertainty by capturing inherent ambiguity in predictions. However, the use of fully Bayesian approaches in large NNs are computationally demanding and suffer from challenges related to scalability [26]. To address this, research has been done into approximate Bayesian inference in DL models. One such approach, which we will delve into further in the following section, involves leveraging dropout as a Bayesian approximation. In this section, we provide a brief overview of Bayesian methods in uncertainty estimation, and their significance in the field.

Bayesian inference is a method where Bayes' theorem is used to update the probability for a prior belief as more evidence or information becomes available. It is a fundamental approach to statistics for incorporating prior knowledge, along with new data, to draw conclusions.

Bayes' theorem can be expressed as:

$$P(\theta|X) = \frac{P(X|\theta)P(\theta)}{P(X)} \tag{5.1}$$

where $P(\theta|X)$ is the posterior probability of the model parameters θ given

the data X. $P(X|\theta)$ is the likelihood of the data given the model parameters. $P(\theta)$ is the prior probability of the model parameters. $P(X)$ is the marginal likelihood or evidence.

The theorem in Equation 5.1 gives us a way to update our beliefs about the model parameters as we observe more data. For NNs, Bayesian inference allows us to measure the uncertainty in the model parameters, and its predictions. However, applying Bayesian inference directly to DNNs is challenging as the design choices are non-trivial [26].

Challenges of Bayesian Neural Networks. While BNNs modify classical NNs by using a distribution over the weights, instead of point estimates, which provides a statistically founded way to incorporate uncertainty, in a framework robust to overfitting. This approach suffer from an intractable posterior, due to the need for integrating over all possible weight configurations, where for instance, in variational inference, we align an approximate posterior distribution $q(\theta)$ with the true posterior distribution $P(\theta|X)$, by minimizing the Kullback-Leibler (KL) divergence, which can be expensive [59, 26].

Challenges of Bayesian Neural Networks. While BNNs modify classical NNs by using a distribution over the weights instead of point estimates, which provides a statistically founded way to incorporate uncertainty and offers robustness to overfitting, this approach suffers from an intractable posterior. The need for integrating over all possible weight configurations makes exact inference computationally prohibitive. For instance, in variational inference, an approximate posterior is aligned with the true posterior, but this process can be computationally expensive and complex to implement [26].

Approximate Bayesian Inference. To make Bayesian methods more tractable, [27, 26, 29, 28] have developed approximate inference techniques. Variational inference and Markov Chain Monte Carlo (MCMC) methods are commonly used approaches, but can still be computationally expensive and hard to implement [26]. The use of dropout as a Bayesian approximation, have been a practical alternative which remains relatively inexpensive, which we will discuss in detail in the next section. Dropout, which traditionally was used as a regularization technique [38, 39], can be interpreted as performing approximate Bayesian inference in NNs, and is an efficient way to estimate uncertainty.

5.3 Dropout as Bayesian Approximation

By leveraging dropout during both training and inference, we can approximate the posterior distribution over the model's weights, allowing us to quantify the uncertainty in its predictions. This approach is known as MCD, and has been shown to be effective in several applications, while being computationally efficient and allows for quantifying model uncertainty [28, 60]. In this section, we will delve deeper into MCD, exploring its theoretical foundations, implementation, and applications in uncertainty estimation for deep learning models. Note that we will follow the notations and theoretical justification presented in both [26], and [28], which provide a simple, yet informative description of MCD.

The idea behind MCD is fairly simple. We are interested in the posterior distribution $p(\boldsymbol{W}|\boldsymbol{X}, \boldsymbol{Y})$ over the models weights $\boldsymbol{W}$. We assume to have training data $\boldsymbol{X}$, and ground truth labels $\boldsymbol{Y}$. As previously discussed, this distribution is intractable, therefore we use variational inference to approximate the posterior distribution, using $q(\boldsymbol{W})$ through minimizing the KL divergence,

$$\mathrm{KL}(q(\boldsymbol{W}) \,||\, p(\boldsymbol{W}|\boldsymbol{X}, \boldsymbol{Y})) = \sum_{\boldsymbol{W} \in \mathcal{W}} q(\boldsymbol{W}) \log \left(\frac{q(\boldsymbol{W})}{p(\boldsymbol{W}|\boldsymbol{X}, \boldsymbol{Y})} \right). \tag{5.2}$$

Then, for every layer i, we have,

$$\begin{aligned} \boldsymbol{W}_i &= M_i \cdot \mathrm{diag}(z_i), \\ z_{i,j} &= \mathrm{Bernoulli}(p_i), \quad \text{for} j = 1, \ldots, K_i \end{aligned} \tag{5.3}$$

which defines the variational distribution for every layer. In Equation 5.3, the unfamiliar M_i is the variational parameters, or weights, before applying dropout. $z_{i,j}$ denotes Bernoulli distributed random variables, with probabilities determined by the dropout probability p_i, which determines the probability of removing the contributions j of the weight matrix per layer.

These modifications make sense for BNNs, that are NNs where we have placed a prior distribution over its weights. A common implementation is by using Gaussian prior distributions over the weights in each layer, $p(\boldsymbol{W}_i)$, where,

$$\boldsymbol{W}_i \sim \mathcal{N}(0, \boldsymbol{I}),$$

and assuming point estimates for the bias vectors b_i. Predictions are for a new

input $\boldsymbol{x}^*$ generated by integrating over all possible weight configurations,

$$p(\boldsymbol{y}^*|\boldsymbol{x}^*, \boldsymbol{X}, Y) = \int p(\boldsymbol{y}^*|\boldsymbol{x}^*, \boldsymbol{W})\, p(\boldsymbol{W}|\boldsymbol{X}, Y)\, d\boldsymbol{W} \approx \int p(\boldsymbol{y}^*|\boldsymbol{x}^*, \boldsymbol{W})\, q(\boldsymbol{W})\, d\boldsymbol{W}, \tag{5.4}$$

where we have approximated the integral using the approximate posterior, that we subsequently approximate using MC integration,

$$p(\boldsymbol{y}^*|\boldsymbol{x}^*, \boldsymbol{X}, Y) \approx \frac{1}{T} \sum_{t=1}^{T} p(\boldsymbol{y}^*|\boldsymbol{x}^*, \hat{\boldsymbol{W}}_t), \tag{5.5}$$

using $\hat{\boldsymbol{W}}_t \sim q(\boldsymbol{W})$, using T MC samples. The procedure of Equation 5.5 is defined as MCD [26].

5.3.1 Implementation and Applications

Following the above description, variational inference in BNNs can be implemented simply by adding dropout layers to a NN. Where the stochasticity introduced by dropout in each forward pass may be viewed as a realization from the posterior distribution over the model's weights, where they collapse into distinct subnetworks. During inference, we use Equation 5.5 which perform MC integration over the approximating distribution [26]. By training a network using dropout, we enforce a constraint on the network, which forces it to learn redundant representations of the data. This is an effective regularization technique as have been discussed in Subsection 2.7.2.

During inference MCD performs multiple stochastic forward passes with different subsets of neurons activated. In practice, this means that an ensemble of subnetworks makes their distinct predictions, before averaging the result using Equation 5.5. This way we are able to model the epistemic uncertainty of the network by also finding the variance of the ensemble's predictions. For object detection tasks where segmentation maps are used, on image inputs, we gain insight into the exact areas the model is uncertain about.

The study conducted by [39] propose the use of weight averaging with dropout disabled during inference, to get fast and accurate predictions, by scaling down the weights of the network with the dropout probability p_i, to approximate a combination of the models. However, [28] demonstrate that their large model

benefits to a larger extent by MC sampling. Besides, the weight scaling approach is unable to model uncertainty.

5.3.2 Bayesian Convolutional Neural Networks

An exciting consequence of the findings discussed in previous sections, is the simple procedure of dropout, with which little modification to existing DCNNs, several of which were already using "standard dropout". Highlighted in [26], their modifications to *LeNet* were able to significantly outperform the standard dropout procedure. This work was later extended to even deeper frameworks of encoder-decoder segmentation models [28] that also highlighted the use of uncertainty maps for scene understanding. They provide a model termed *Bayesian SegNet*, a Bayesian adoption of the earlier *SegNet* architecture [49].

Bayesian SegNet [28] introduced key contributions to the field. One important insight was that a fully Bayesian network, which applies dropout layers after each convolutional layer, is not necessarily the most effective approach. Instead, they demonstrated that strategically applying dropout in specific layers, the central 6 in the bottleneck, proved most beneficial for evaluation performance.

Another significant contribution of Bayesian SegNet [28] was its use of uncertainty maps for scene understanding. By leveraging dropout during inference, the model can generate multiple predictions and quantify the uncertainty in these predictions, with averaging and variance. These uncertainty maps provide valuable information about the confidence of the model in different regions of the image, which is important for applications like autonomous driving and medical imaging where understanding model confidence is vital. They found that along borders of objects, the uncertainties were large. Additionally, when the model predicts wrong, the uncertainty is generally high, they observe an inverse relationship between class accuracy and the uncertainty. This approach to uncertainty was further explored in the paper [29]. They discuss the types of uncertainties — aleatoric and epistemic — that are important in computer vision tasks, and how these uncertainties can be quantified and utilized to improve model performance and robustness.

Part II

Proposed Methodology

6

Uncertainty Guided Building Extraction

Understanding the limitations of recent deep learning models, such as the *Hierarchical Supervision* (HiSup) [17] architecture, is crucial for advancing the accuracy and robustness of building footprint extraction. This chapter aims to dissect specific shortcomings and propose innovative strategies to address them. Section 6.1 will discuss shortcomings and limitations in the current state-of-the-art, such as low confidence borders of the binary map which is an inherent issue with segmentation maps, where occlusions additionally magnify this effect. In improving the robustness of the framework we address the usage of uncertainty. We further demonstrate our approach to leverage the uncertainty by devising a modified version of the MAV-ATTR algorithm from [17] in Section 6.3.

6.1 Error Propagation of Vague Borders

Facilitating accurate polygonization require precise segmentation maps. As shown in Figure 4.1 the current state-of-the-art approach, while producing accurate segmentation maps, struggles to produce high-confident predictions along the borders of buildings. Accurately annotating the borders of buildings is a difficult task, where we expect there to be some margin of error. Other conditions that will have an impact on the accuracy of the border can be occlu-

sion from vegetation, and parts of buildings which are not accounted for in the ground truth annotations, e.g., balconies, as well as sides of buildings which are slanted away from the imaging sensor, such that its features are even harder to capture from single images. This margin of error can be imagined to propagate into the DL models during training, which in turn predicts borders with lower confidence. The errors caused by the ambiguity in the border prediction consequentially may cause large errors for downstream tasks. In the case of HiSup, one component that is particularly dependent on the precise boundary segmentation is MAV-ATTR, as described in Subsection 4.4.4. In particular, it suffers from low confidence borders as it excludes certain vertices from being included in the final polygon due to the distance between the contour of the segmentation map and a vertex being too great.

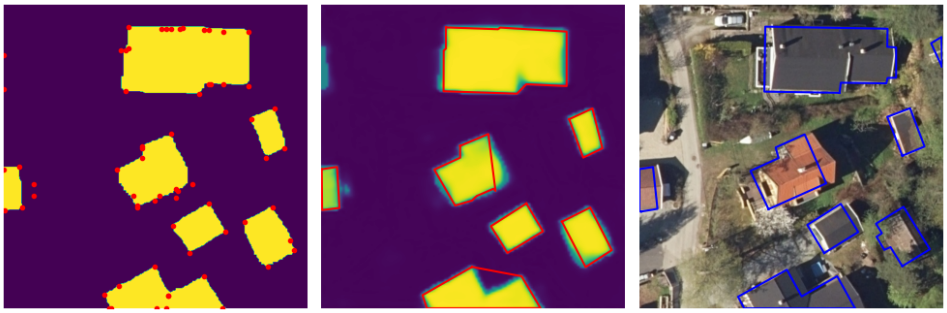

Figure 6.1: Generated vertices and polygons from the example predictions shown in Figure 4.1. The leftmost figure contains the segmentation mask referred to as *Remask* (in Figure 4.1), conditioned on prediction confidence greater than 0.5. The red vertices have confidence greater than $\tau_k = 0.008$, which is the default setting also used in [17]. Those with confidence lower than τ_k are discarded, while NMS is applied to the remaining ones. The center figure contain the polygons produced by MAV-ATTR, overlaid onto the predicted segmentation map without a threshold to highlight ambiguities in the border between foreground and background. The leftmost figure displays the ground truth polygons on top of the input image.

An example of this is illustrated in Figure 6.1, where the center and bottom polygons are missing a vertex. The vertex however is present and has been predicted by the model. However, MAV-ATTR discards it for being too far from the predicted contour, and since the generated polygon is purely drawn between vertices, the method produces a suboptimal result.

A more robust segmentation of the boundary region, and quantification of the uncertainty within this prediction thus has the potential to improve the overall performance of the model. Furthermore, from a user's perspective, quantifying this uncertainty has the advantage to relay the model uncertainty, thereby building trust in the model.

6.2 Stochastically Sampling Segmentation and Vertex Maps

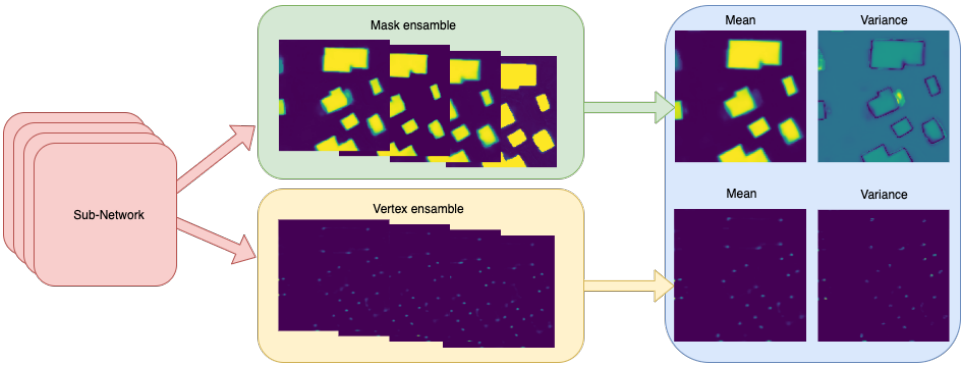

Figure 6.2: Outline the proposed approach to perform Bayesian approximation by performing multiple stochastic forward passes, enabling an ensemble of models. The figure only highlights two predictions, while in fact, Figure 4.1 displays all the outputs which would be combined in this setting.

If we assume that the margin of error between the vertices of the ground truth annotation polygons and the ideal building corners in the image follows a distribution. Then a method such as MCD, described in Section 5.3 would for each stochastic forward pass theoretically sample from this distribution. For a single corner of a building, multiple forward passes with dropout enabled would be thought to be sampling from the Gaussian distribution.

To facilitate more robust outputs of the model we employ the theory discussed in Chapter 5, and propose a modified version of the *Hierarchical Supervision* framework [17], inspired by MCD (as discussed in Section 5.3). This is to allow performing multiple stochastic forward passes, sampling both segmentation and vertex maps from the posterior distribution over the weights of the modified model, using dropout during testing, leveraging an ensemble of predictions. We follow the general setup of [28] by stacking the sampled segmentation and vertex maps, as shown in Figure 6.2, then using their respective depth-wise mean for prediction, and the variance to describe the uncertainty.

Modified framework. During model training, we include dropout layers in both the backbone feature extractor and the *Hierarchical Supervision* architecture. The backbone used is the same as the one that was used by the original authors, which is described in Section 3.4. Within each convolutional block, which there are four of in the largest configuration - built up of convolutional layers and batch normalization operations followed by the RELU activation function - we place a dropout layer with p_{dropout} as the dropout probability. The transitional layers, which are responsible for both upsampling

and downsampling the convolution streams also include a dropout layer with the same dropout probability. The main model described in Section 4.4 which use the feature maps extracted by the backbone are also modified with dropout layers. The authors of [17], in parallel, apply three blocks consisting of 3×3 convolutions followed by batch normalization and ReLU activation function to extract feature maps for the segmentation task, AFM task and vertex task. We include a dropout layer, as before, after each batch normalization layer, before the activation function is applied. To align the segmentation predictor and vertex predictor, the authors then employ ECA to enable cross-level interactions between the parallel tasks. We modify ECA by inserting a dropout layer before the attention activations are computed of the cross-level interactions.

Robust segmentation and vertex maps. Allowing multiple distinct sub-networks to contribute in an ensemble setting, we expect the final segmentation maps to refine low confidence borders, producing a more robust contour, which is less likely to miss vertices during the polygon generation process. The vertices positions are also refined, and as we see from Figure 6.2 the heatmap of vertices can now be considered a better estimator for the Gaussian distribution of each vertex, resulting in a sample mean closer to the true positions of building corners. During inference, we average MCD samples from multiple stochastic forward passes, which is shown to produce more robust predictions than model averaging, for a representative sample size [27, 28].

6.3 Dynamic Uncertainty Based Polygon Refinement

While Section 6.2 presents a refining procedure for the mask predictions using MCD, this chapter addresses the problem of the fixed distance threshold in MAV-ATTR, which discards vertices based on the distance between a vertex and the predicted contour. We present a modified procedure utilizing the uncertainty maps produced from the ensemble of models to replace the fixed threshold with a dynamic one. Subsection 6.3.1 demonstrates the problems involved with the prior method, while Subsection 6.3.2 proposes an uncertainty guided modification to MAV-ATTR.

6.3.1 Sensitivity to Parameters During Inference

In addition to addressing and capture the missing vertices of Figure 6.1, we scrutinize the fixed distance threshold, which MAV-ATTR uses. In their paper

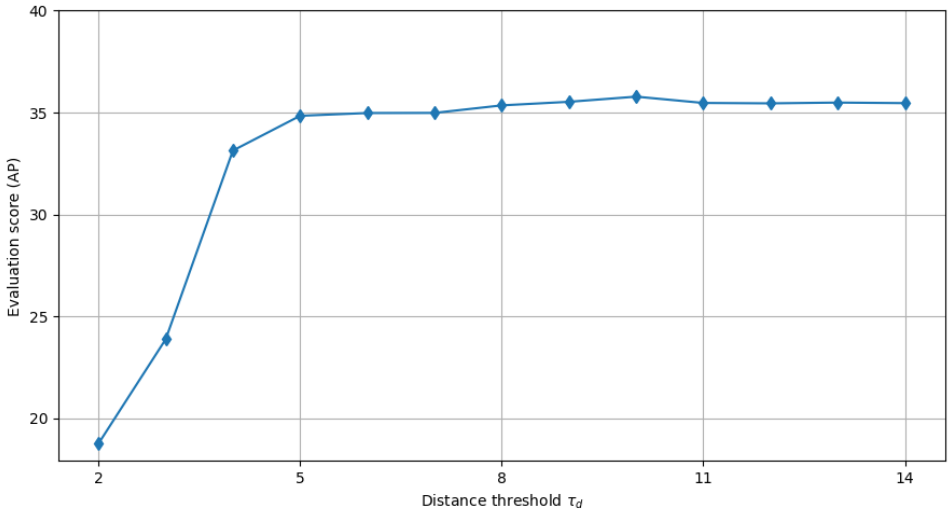

Figure 6.3: AP evaluation metric with different distance thresholds τ_d when evaluating the predictions of a single model. The authors of [17] defaults to using a threshold of $\tau_d = 5$. We observe that for our dataset, a threshold of $\tau_d = 10$ produce even better results.

[17] the authors demonstrate how varying the threshold used in the polygon generation procedure is needed to find the optimal parameters for each dataset. They find that for the *Inria*, *AICrowd* and *Shanghai* datasets a distance threshold of *around* 5 yield the highest accuracies, however the *Open Cities* dataset has an 13% increase in AP when they increase the distance threshold to 11, which they hypothesize is caused by the larger difficulty in the dataset itself. Our own analysis conclude our model benefits from using a distance threshold of 10, as is apparent from Figure 6.3.

The effects of varying the distance threshold Figure 6.4 demonstrates how the distance threshold impacts which vertices are selected. A simple method for improving the final polygons can be implemented by increasing this distance threshold, as shown in Figure 6.4 and Figure 6.3. Increasing the threshold does however require human intervention, and an analysis of performance over a potentially large dataset. Furthermore, the right column of Figure 6.4 shows our persistent example, when the threshold is increased to the optimal one, of $\tau_d = 10$, as demonstrated in Figure 6.3. While the bottom polygon adapts well with the new threshold, the center one ends up including several redundant vertices, while the in situ image in Figure 6.1 confirms that the 6 vertices predicted ideally would be represented using only 2 vertices. Hence, increasing the threshold to some extent also increases the complexity of the polygons as an unfortunate side effect.

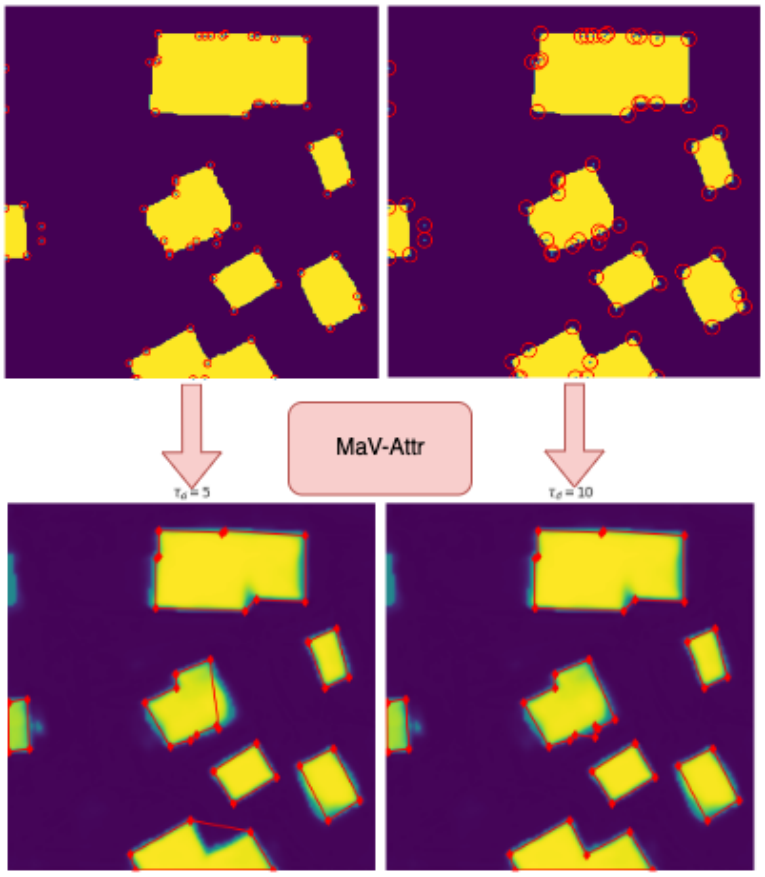

Figure 6.4: Example of vertex selection process in MaV-Attr, a continuation of the example in Figure 6.1 which demonstrates why a vertex is discarded. The illustration on the left displays a distance threshold τ_d of 5. To its right, the threshold is increased to 10. The criteria used in selecting a vertex is a fixed distance between the vertex position and any point along the contour of the segmentation map, which is displayed as the yellow areas.

6.3.2 Dynamic Threshold Selection

We are thus motivated to use the uncertainty maps gained from our proposal in Section 6.2. Although we have uncertainty maps, it is not trivial which one to use or what combination. Many different combinations, leveraging both the uncertainty in the segmentation map and vertex and offset maps, were evaluated, with the goal of devising a method simple enough to inspire other polygon generation algorithms to be able to adopt our proposal. For instance, we attempted to slack the threshold based on the uncertainty in the borders of the segmentation maps, giving the control of dynamic vertex selection to the uncertainty of the segmentation maps. This method was discarded, where we hypothesized that vertex selection using uncertainties need direct feedback

from the predicted vertex heatmap.

Polygon initialization. Following the notation of [17] of MAV-ATTR as outlined in Subsection 4.4.4 we denote the predicted mask $\boldsymbol{S}_{\text{afm}}$, the corresponding vertex heatmap $\boldsymbol{H}$ and offset map $\boldsymbol{O}$ with their respective uncertainty maps $\boldsymbol{U}_S$, $\boldsymbol{U}_H$ and $\boldsymbol{U}_O$. For dynamic threshold selection, we define,

$$\boldsymbol{U}_H = \text{var}\Big(\{\boldsymbol{H}\}_{i=0}^{N} > \tau_v\Big), \tag{6.1}$$

to be the uncertainty of the vertex predictions. Where $\{\boldsymbol{H}\}_{i=0}^{N}$ denotes the stack of both convex and concave vertex map predictions. The variance is applied depth wise, therefore the dimensions of the map is retained, and we are left with a per-pixel variance. τ_v is the same parameter used in the original algorithm, set to $\tau_v = 0.008$, which discards low confidence vertices in the same manner as with the actual predictions. The rest of the initialization procedure of [17], described in Subsection 4.4.4 is done as the original authors presents, while we use the computed vertex uncertainty map $\boldsymbol{U}_H$ during the simplification process.

Polygon simplification. The original simplification process consists of 4 steps, where we modify step 3 (see Subsection 4.4.4) to instead dynamically set the distance threshold for each pixel using their associated uncertainty. Where the original method would discard pixel $\boldsymbol{x}_k^i$ if the distance to the closest vertex $\boldsymbol{v}_{\pi(k,i)} \in V$ is greater than τ_d. We modify this approach to include an additional term $u_{\boldsymbol{v}_j}$, which is defined as,

$$u_{\boldsymbol{v}_j} = u_{v_j[x,y]} = \sum_{m=-\tau_n/2}^{\tau_n/2} \sum_{n=-\tau_n/2}^{\tau_n/2} \boldsymbol{U}_H[x-m, y-n], \tag{6.2}$$

i.e., the $\tau_n \times \tau_n$ neighborhood of uncertainties in $\boldsymbol{U}_H$ is associated with vertex $\boldsymbol{v}_j$, to incorporate the local neighborhood of variance around the chosen vertex. To exaggerate the effects of the neighborhood of uncertainties around the vertex, we linearly scale $u_{\boldsymbol{v}_j}$, to the interval $(0, \tau_s)$. We then discard pixel $\boldsymbol{x}_k^i$ if,

$$\|\boldsymbol{x}_k^i - \boldsymbol{v}_j\|_2 > \tau_d + c_{\boldsymbol{v}_j}. \tag{6.3}$$

This can be interpreted using the statement, given the set of vertices V, the algorithm may relax its distance threshold if the ensemble of models is uncertain

in the position of a vertex $\boldsymbol{v}$. We now in effect have two additional parameters in our modified algorithm. We do however hypothesize that the sensitivity to each parameter is less than when using only a single one for the distance threshold, as our parameters simply would scale the effect of the uncertainty guidance of the original threshold. While a lot of different methods for leveraging the uncertainty maps in the segmentation maps have been explored in development of our method, e.g., by discarding vertices or boundary pixels based on the pixels' certainty, we found that scaling the distance threshold only using the uncertainty of vertices qualitatively produce better results.

We have with this introduced two methods for refining both the predictions of the framework presented in [17], and the polygon generation method by leveraging the uncertainty maps produced by the first proposal. Addressing the need for refined edge and vertex maps through the use of MCD, allows us to rigorously express the uncertainty in both the borders of segmentation maps and the positions of each vertex. The following chapters will present experiments using the proposed methods, with both qualitative and quantitative analysis of our proposals.

Part III

Experiments

7

Experimental Setup

The following part consists of a description of the experiments that have been conducted, with the used setup, datasets and metrics used in measuring the performance of our method and modifications to the framework from [17], presented in Part II. We begin discussing the datasets used during our experiments in Section 7.1, before presenting the evaluation metrics used to measure the performance in Section 7.2, that will be used in Chapter 8 to compare our method with the prior. This chapter is rounded off by discussing the details specific to our implementation and outlining the training process in Section 7.3.

7.1 Datasets

To accurately delineate building footprints, there is a need for vast examples of high-resolution aerial or satellite images with accurate ground truth annotations. There exist common benchmarking datasets which stem from previous or active competitions, such as the *AICrowd* [61], *Inria* [62], *Open Cities AI Challenge* [1] and the *Shanghai* dataset, to name a few. Both the AICrowd and Shanghai dataset are subsets of the more extensive *SpaceNet* datasets and challenge series [63]. The spatial resolution of these datasets are generally

1. Project home page can be found at: https://www.drivendata.org/competitions/60/building-segmentation-disaster-resilience/page/150/

between 1 m to 5 cm, i.e., the datasets have vast differences in spatial resolution, large distributional shifts, varying degrees of occlusions from vegetation, and challenges related to preprocessing and orthorectification of the images. Additionally, the ground truth labels needed to train satisfactory DL models, in general, require extensive effort and oversight from humans, with partially automated techniques with human corrections have been used in some cases [63]. In this work, we focus in particular on two datasets. The first one is a proprietary Norwegian dataset (referred to as the *Field* dataset), which allows us to test the method outside the commonly explored datasets in order to evaluate the "in-the-wild" performance. In addition, we leverage the AICrowd dataset, which is one of the most widely used datasets, for comparing with a common benchmark.

7.1.1 Field Dataset

The main dataset that will be used for our experiments is produced by *Field*[2], and provided for this experiment by the Norwegian Computing Center (NR). It consists of 8000 × 6000 pixel TIFF-format Red, Green, and Blue (RGB) image mosaics, of selected parts of southeastern Akershus county in Norway. The images are captured with a 10 cm spatial resolution, acquired both in the years 2018 and 2020. Being well suited for building detection and change detection, as shown in [22]. The areas captured are both of densely populated, mostly single-family homes, also including larger uninhabited forests and cropland. The images are orthorectified, meaning the pixels are stretched to fit over a Digital Surface Model (DSM) to account for differences in elevation caused by the off-nadir angle they are captured at.

Figure 7.1: Example images of the Norwegian Computing Center (NR) dataset.

For our experiment, we found the 2018 dataset to be adequate, consisting of approximately, 120000 distinct polygons of buildings. We resize the mosaics into 512×512 pixel PNG-formatted images, also cropping their accompanying TFW

2. Link to their website: https://field.group/

files, which contain the geospatial extent of each crop and additional metadata. To construct partitions for training, testing and validation, the dataset is split using an 8-1-1 ratio resulting in a distribution of approximately 30 000, 3 500, 3 500 images respectively. As part of the data preprocessing procedure, we convert the accompanying Shapefile for the 2018 version into three separate annotation files using the Microsoft Common Objects in COntext (MS COCO) format, for each partition. For ease of use, each dataset partition is accompanied by a smaller subset of about 10% the size of the original. We also discard cropped images not containing any ground truth annotations.

Differing from the aforementioned other benchmarking datasets, there is a prominent distributional shift due to the difference in vegetation and more sparse urban development in parts of the dataset. While the other datasets see vegetation more in the form of parks, and the occasional planted cluster of trees, the Field dataset on the other hand, to some extent captures dense wild forests, and long stretches of empty roads and farmland with the sporadic shed.

7.1.2 AICrowd dataset

We are also motivated to test our proposed methods on a common dataset with [17]. Therefore, we include the AICrowd mapping challenge dataset[3] [61]. The dataset is a subset of the SpaceNet dataset [63] that originally contained multispectral data, where the non-RGB channels have been discarded. The images are *pansharpened*, a process where a channel sensitive to a larger spectrum - including visible light - with higher spatial resolution is "colorized" using the RGB channels. This has the effect of efficiently upsampling the spatial resolution of the other channels, and is a common technique in modern optical spacecraft. While the spatial resolution is not explicitly stated, the imaging satellites used - *WorldView* 2 and 3 - are able to capture images in the range of 30 cm to 50 cm spatial resolution [63].

The training dataset includes 280 741 tiles of 300×300 pixels, while the validation set consists of 60 317 tiles, accompanied by MS COCO formatted ground truth labels. Since this is a challenge dataset, the annotations of the test partition has not been released, meaning we use the validation dataset for testing. As discussed in Subsection 7.1.1, we here also have access to a smaller version of each dataset partition to simplify testing.

From Figure 7.2 we see more densely populated suburban homes than in Figure 7.1. There is also less vegetation, except a few buildings occluded by trees.

3. Dataset is available on: https://www.aicrowd.com/challenges/mapping-challenge

Figure 7.2: Examples from the *AICrowd* mapping challenge dataset [61].

We are to some extent are able to notice a difference in the color distribution between the dataset, likely due to being captured by satellites, meaning the photons need to pass through a denser atmosphere than the Field dataset. This is reflected in the data pipeline discussed in section Section 7.3 where we normalize the color channels of the images using the mean and standard deviation of the global dataset per channel and see the standard deviation of the intensity per channel is 50% less for the AICrowd dataset.

7.2 Evaluation Metrics

We will now introduce the evaluation metrics that we use during our experiments. To compare polygons, .

IoU is a simple metric for evaluating the overlap between two sets. In our case of building detection, we use it to compare a detection with the ground truth, effectively measuring the overlap of the two. We define it as,

$$\text{IoU}(A, B) = \frac{A \cap B}{A \cup B}, \tag{7.1}$$

to compare A and B. A reformulation of the above equation could be the footprint of overlap divided by the footprint of union. For a perfect overlap, an Intersection over Union (IOU) of 1 is achieved.

COCO We will rely on the evaluation metric [64], which is the official metric of our AICrowd dataset [61], to be able to compare with existing methods. The detection metrics are Average Precision (AP) and Average Recall (AR). The AP summarizes the precision-recall curve by computing the average precision

across multiple IOU thresholds. We will be using the main metric that averages IOU threshold between 0.5 and 0.95, with increments of 0.05.

$$\begin{aligned} \text{AP} &= \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} \frac{\text{tp}}{\text{tp} + \text{fp}} \\ \text{AR} &= \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} \frac{\text{tp}}{\text{tp} + \text{fn}} \end{aligned} \tag{7.2}$$

From Equation 7.2 $\mathcal{T}$ is the set ouf IOU thresholds.

Complexity Aware IoU Is a metric proposed by [15], that addresses the fact that in the search for better building polygons, pure segmentation models have an advantage in that their extracted polygons have irregular shapes, with redundant vertices. They introduce the Complexity Aware Intersection over Union (C-IOU) that penalizes, not only based on the IOU, but also on the extracted vertices and their quality.

$$\text{C-IoU}(A, B) = \text{IoU}(A, B) \cdot (1 - \frac{||\partial A| - ||\partial B|}{|\partial A| + |\partial|} \tag{7.3}$$

In Equation 7.3 $|\partial A|$ denotes the number of vertices in each region.

Boundary IoU. The Boundary IOU [65] is another metric that penalizes polygons with inconsistent borders. To do this, they only consider the sets of pixels within a distance d, known as the dilation ratio, during their IOU computation. In our experiments, we set it to $d = 0.02$, similar to [17].

$$\text{Boundary IoU}(A, B) = \frac{|(A_d \cap A) \cap (B_d \cap B)|}{|(A_d \cap A) \cup (B_d \cap B)|} \tag{7.4}$$

PoLiS. The PoLiS [66] metric evaluates predicted and ground truth polygons by measuring geometric properties such as vertex and edge comparisons, with an increased sensitivity to small variations in shape and scale. And is an important metric in our later experiment as it fundamentally measures different properties from the IOU based metrics. It measures the average distance between each vertex of ∂A with its closest point b within the boundary ∂B, and is formulated as:

$$\text{PoLiS}(A, B) = \frac{1}{2q} \sum_{a_j \in A} \min_{b \in \partial B} ||a_j - b|| + \frac{1}{2r} \sum_{b_k \in B} \min_{a \in \partial A} ||b_k - a|| \tag{7.5}$$

7.3 Implementation Details

We present the details of our training procedure to provide a reproducible setup. We largely base our proposals on the work done by [17], trying to produce a framework improving on their state-of-the-art model[4].

Backbone. We train a series of models with different configurations before settling with one model per dataset. Similar to the proceeding capstone project of this thesis [67], and [17], the at the time widest variant of the second version of the backbone proposed by [48], termed *HRNetV2-W48*[5]. As its name suggests, the width of the subnetworks in the last three stages are 48. This backbone inputs images of 512×512 as input, explaining the choice of crop sizes used in Subsection 7.1.1, for the Field dataset. The backbones are pretrained on ImageNet [68], meaning the feature extractor will have some familiarity with certain useful representations. The feature maps output from the backbone are downsampled with a factor of 4, meaning 128×128, which we resize to the original image size during inference.

Hyperparameters. Due to the differing input sizes of images, we vary the batch size between the datasets. The model trained on the Field dataset uses a batch size of 4 due to memory constraints, whilst the AICrowd trained model use a batch size of 7. For the remaining details and hyperparameters, we largely follow the same setup as [17]. The Adam optimizer [36], described in Subsection 2.3.2 is used with a base learning rate of 10^{-4}, which is decayed by 10 after 25, 75 and 150 epochs. The AICrowd trained model is trained over 30 epochs, while the Field dataset receives 200 epochs of training. We use a weight decay of 10^{-4}. We also augment the data during training to introduce additional variance. The augmentation strategies used are random rotation, flip, and color jittering. We train the models on a single Nvidia RTX 3090 GPU, and training takes upwards of 200 hours per model. Additional configurations using a cyclic learning rate [69] were tested, however the basic learning rate scheduler gave similar results and was therefore the only one used in producing the final results. Our variant using MCD during inference, in general, use 10 stochastic forward passes during inference. For the distance threshold used in the original version of MAV-ATTR we use $\tau_d = 5$ for the AICrowd dataset, and $\tau_d = 10$ for the Field dataset. During both training and inference, we set the dropout probability in all dropout layers to $p_{\text{dropout}} = 0.1$.

For the modified version of MAV-ATTR, that we propose in Section 6.3 has a

4. The code for HiSup can be found at: https://github.com/SarahwXU/HiSup
5. The authors have also provided wider versions, however with diminishing returns. Code and pretrained models are available at: https://github.com/HRNet/HRNet-Image-Classification

parameter τ_n which denotes the neighborhood to consider uncertainties in for a vertex, and we set this to 2 for both datasets, since this threshold is only applicable to downsampled 128×128 pixels uncertainty maps. We further set the scaling parameter $\tau_s = 2$, hence the effect of the uncertainty guidance is at most 2.

8

Experiments and Results

Building on our proposed methodology in Part II, we conduct in this chapter experiments to evaluate its benefit compared to the state-of-the-art method. Section 8.1 will present results using our improved ensemble strategy described in Chapter 6. The results are directly followed by a discussion and comparison with the prior state-of-the-art. Section 8.2 shifts our focus to using the uncertainty of the model to dynamically relax or enforce the distance threshold, following the method of Section 6.3. The last section of this chapter, Section 8.3, outlines variations to our method, and where we see more research is needed.

8.1 Bayesian Ensemble of Predictions

Table 8.1 and Table 8.2 demonstrate our quantitative results using two classes of models, each trained on one of the datasets used in our work. The large gap in performance between the models trained likely stem from the vast differences between the data. The AICrowd dataset consists of an approximate tenfold more training examples, which may be a significant factor, we further investigate the effects of the data distribution in the later Subsection 8.1.5.

Table 8.1: Evaluation results comparing the benchmark HiSup architecture [17], against using standard dropout with weight scaling, and MCD, for models trained on the Field dataset. All models are trained for 200 epochs. The arrows next to each column label indicates whether a better result is high or low. It is clear that the inclusion of MCD has a significant impact on the robustness.

	AP ↑	AR ↑	AP^b ↑	PoLiS↓	C-IoU↑	IoU↑
[17]	39.6	46.9	21.4	4.170	62.6	74.3
[17] + dropout	36.5	43.1	18.1	4.436	58.9	71.5
[17] + MCD	**40.8**	**48.3**	**22.6**	**3.920**	**63.6**	**74.8**

8.1.1 Results Using Dropout

Table 8.1 and Table 8.2 presents our results when testing a model which has been trained using dropout layers inside the convolutional blocks. Comparing with the baseline, it is clear the performance has deteriorated, which is an unexpected result. The goal with using dropout is to regularize the NN by forcing the network to learn redundant representations, thus preventing co-adaptations where a single neuron is only helpful in the context of several other neurons [38, 39], as we have elaborated on in Subsection 2.7.2. This means that the deterioration seen on the Field dataset results may either indicate that the model is not yet sufficiently trained, or could be the result of a problematic implementation of dropout. We see both alternatives as possible, however the model trained on the Field dataset received 200 epochs of training, where the loss does not significantly decrease during the last 100 epochs. While overtraining also could be a potential issue, the inclusion of dropout layers to a large extent helps with mitigating such issues [38, 39]. Because of this, we are open to the idea that our use of dropout layers should have been limited to a smaller extent, for instance following the *Bayesian SegNet* [28] architecture in constraining dropout layers to the *Central Four Encoder-Decoder*, design idea more closely, by mainly using dropout layers during the compact stages of the backbone bottleneck, as well as only during the convolutional blocks of the HiSup framework. Our model also had dropout layers in the initial convolutional blocks of the prediction heads, which could cause important information to be discarded, although [28] show that their version using dropout in the last unit before the classifier excelled in performance. Another potential configuration that was not tested would be to exclusively have dropout layers in the backbone feature extractor or exclusively in the main framework. However, we were unfortunately not able to test more configurations to properly view these effects.

8.1.2 Results using Monte Carlo Dropout

Table 8.2: Evaluation results comparing the benchmark HiSup architecture [17] with standard dropout and MCD, that have been trained on the AICrowd dataset [61]. Note that the baseline is not directly comparable to the dropout and MCD scores, because the model is trained for 100 epochs. Thus, we refer the reader to Table 8.1 for a proper comparison with the baseline. While the better trained model outperforms both modifications, MCD significantly reduced the gap to the 100 epoch model.

	Epochs	AP ↑	AR ↑	AP^b ↑	PoLiS↓	C-IoU↑	IoU↑
[17]	100	85.3	80.9	66.3	0.738	89.6	94.1
[17] + dropout	30	58.6	60.4	41.7	1.245	70.4	77.5
[17] + MCD	30	69.7	73.2	51.8	1.155	84.4	90.2

While we suspect there to be issues with our implementation of dropout, when activating dropout during testing and averaging the results, the performance in all metrics improve, as seen in Table 8.1 and Table 8.2. Due to not having a comparable baseline for the AICrowd dataset, because the model is trained over 100 epochs, we instead compare our model trained on the AICrowd [61] dataset between the standard dropout model, and the MCD model. We notice a large increase over all metrics, where the main comparison metric AP increase by 11.1%. Additionally, the more difficult boundary sensitive AP^b increase by 10.1% demonstrating the superiority of MCD. For the Field dataset, we are however able to compare results between the baseline, standard dropout and MCD. While we have discussed the decrease in accuracies between the baseline and the standard dropout model, there are clear improvements made when evaluating our MCD modified model using an ensemble of 10 MC simulations. As we will discuss in Subsection 8.1.6, the accuracies of the model trained on the Field dataset will be consistently lower than for the models trained on the AICrowd dataset, which is due to the difference in spatial resolution, occlusions, and inherent differences between the datasets.

Our evaluation use 10 MC samples. This allows the model to sample from an estimate distribution of the true position of vertices from the dataset, as outlined in our proposed methodology Section 6.1. On inspection of Figure 8.1, we observe predictions, where the columns contain the same images, but using a single prediction in the top row, and the average of 10 predictions in the bottom row. By comparing the predictions of (a) with (e), we have the situation where the single prediction (top) has not been able to predict the buildings to the right of the image, however when averaging over 10 predictions, the model is able to correct its mistake since there are other predictions where the building was detected. This is an example where the model becomes more robust, which was our goal in using MCD, thus confirming our hypothesis. The same figure captures an example of the opposite case, where the single

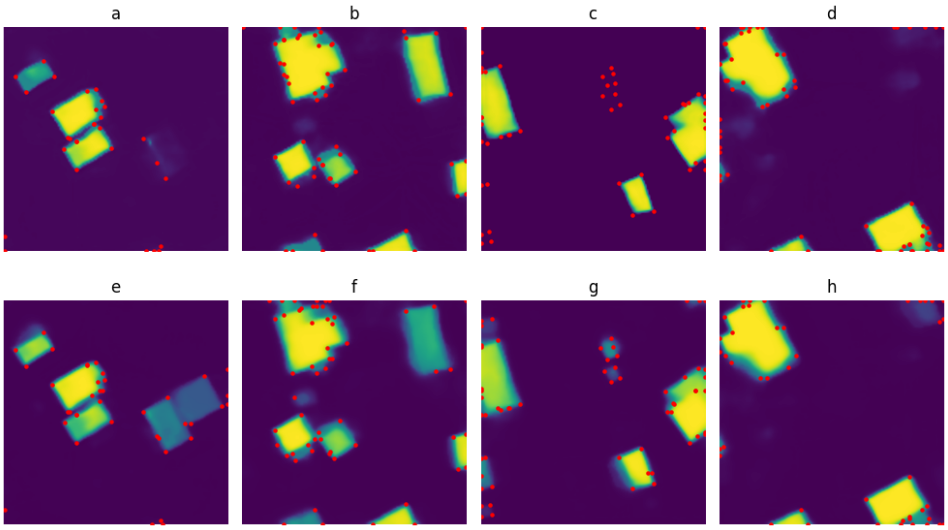

Figure 8.1: Qualitative comparison of predictions using a single prediction versus an ensemble of 10. The top row, images (a) to (d) are using a single prediction of vertices and segmentation masks, while the bottom row average predictions over 10 stochastic forward passes.

prediction has a high confidence for the top right building of (b) in Figure 8.1, while its MCD counterpart is more uncertain, seen by the fainter color. We see both examples are in Figure 8.2 (a) and (b), where the uncertainty is large in both cases. The bottom input image with ground truth labels overlaid in Figure 8.2 of (a) and (b) gives us an interesting case. (b) Reveals there is in fact a building there, however its entire footprint is densely occluded by vegetation and its shadows, to the point where visual inspection nearly is impossible, even by humans. Thus, since some of the ensembles were able to detect it, mean the ensembles as a whole is more robust, as highly uncertain footprints can be leveraged in other settings, such as active learning. For (a), the ground truth labels are only present on the right building, and not on the left building, which is unfortunate. However, the uncertainty is high for the same buildings, which would mean that some of the predictions were not able to detect these buildings, which is peculiar, seeing how unobscured both buildings are in the input image.

The uncertainty of the vertex heatmaps of Figure 8.2 are harder to see, but the interested reader may magnify the images for clarity. The leftmost figure (a) have larger values because the different predictions would have slightly varying confidence levels in the heatmap, and disagree on the centroid position for up to a few pixels, causing the variance to be high. Without our modifications, the offset map would probably account for this, however a downside with our implementation is that vertex maps are averaged before the offset map

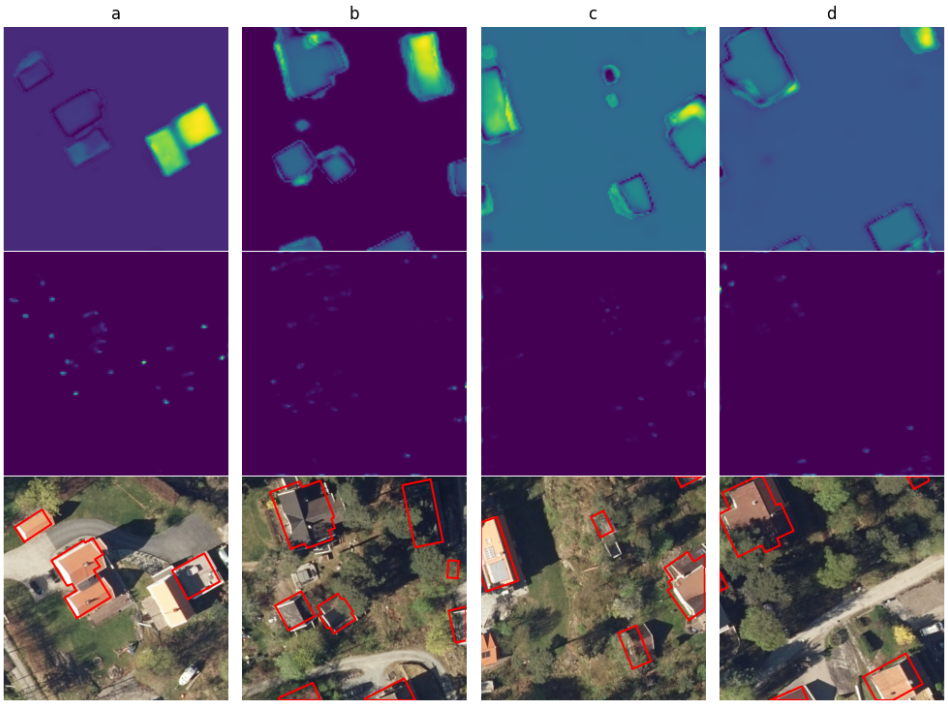

Figure 8.2: Uncertainty maps of predictions from Figure 8.1. The top row shows the uncertainty of the segmentation maps, while the middle row illustrate the uncertainty of the vertices. In the bottom row, we have the input images with the ground truth labels overlaid in red. Due to some segmentation maps being predicted with a difference in the low confidence areas, i.e., the background predictions, we observe differences in the background color of these illustrations. The vertex uncertainty is computed by adding the convex and concave vertex predictions before computing the variance of each pixel. The columns of this figure correspond with the bottom row predictions of Figure 8.1.

is applied. We instead average convex and concave vertex maps separately, in addition to averaging the horizontal and vertical offset maps by themselves, before applying the conversion algorithm between probability maps and vertex vector coordinates, following [17]. Because the feature extractor backbone acts as the *eyes* of our models, when the convex and concave vertex maps from a single prediction has an offset, the offset map is supposed to correct for this. The advantage with this is that the offset map is directly linked with the vertex probability maps. Due to our method averaging of offset maps, we lose the tailored maps for each prediction, which could negatively impact the precision. On the other hand, the offset maps are only able to refine the position of a vertex up to half a pixel at most, and are really only applicable as a method to achieve sub-pixel coordinate representations.

8.1.3 Distance Between Mask and Polygon Predictions

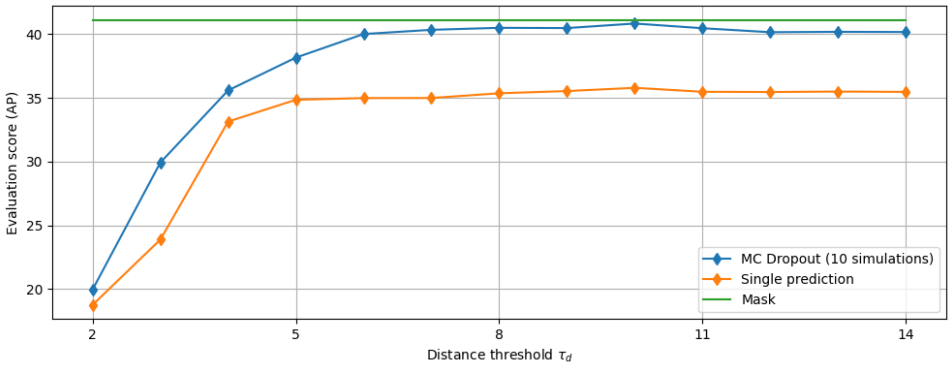

Figure 8.3: Comparison of distance threshold using a single prediction and 10 MCD predictions. This establishes a baseline to evaluate the performance of our proposed dynamic threshold selection algorithm. It is clear that both curves are fairly stable above an arbitrary threshold of 40% in AP during evaluation of the distance thresholds, while having a slight peak at $\tau_d = 10$ which is the selected threshold used in our previous evaluations. The accuracies of the blue curve serves as a benchmark for evaluating our dynamic thresholds selection modification to MAV-ATTR in Section 8.2.

A subject we have yet to discuss, is the difference in achieved accuracies between the predicted mask before a polygon is generated and the generated polygon using MAV-ATTR. Before direct polygon prediction became feasible, buildings were first detected using a framework such as *Mask-RCNN* which was evaluated in [61], that predicts a segmentation map but does not produce polygons by themselves. In evaluating their own model [17] compare with *Mask-RCNN* by performing DP simplification [52] to generate polygons from the predicted building contours, which were *probably* extracted with the Marching Squares algorithm [51], however they do not provide further details. Their own framework, HiSup [17], also initially predict building masks; interestingly, these tend to achieve an even higher accuracy than the vectorized polygon extracted from it, which we see in Figure 8.3. The green line denotes the mask prediction's accuracy. While the performance is better on the AP metric, when we extract a polygon directly - without using the vertex information - the final representation consists of *too* many redundant vertices, which the MAV-ATTR algorithm is able to remove. Figure 8.4 demonstrates our point, where the top row contain polygons generated directly from the mask prediction using DP simplification, while the bottom row use MAV-ATTR. There is a large difference in the amount of vertices used in representing the polygons between the methods, but MAV-ATTR is preferred from downstream tasks due to having a sparser representation of the same polygons, which for large datasets, can translate to enormous storage optimizations. Besides, MAV-ATTR has the additional benefit of regularizing its generated polygons due to finding a minimal representation

of vertices.

However, this observation poses a question, *is the prediction accuracy of the masks the highest achievable, or close to the highest possible accuracy in any version of polygons that* MAV-ATTR *is able to generate*. While our tests have never been able to achieve accuracies that outperform the raw mask predictions, we are not able to conclude with a definite statement. If we however assume the mask predictions to be the absolute maximum that a polygon generation algorithm such as MAV-ATTR is able to achieve - from the same masks - our other proposed method which we will come back to in Section 8.2 should only be able to improve up to the accuracies produced using only the masks.

On inspection of Figure 8.3, the mask prediction achieves an accuracy of 41.11% AP while the best results using 10 MC simulations is reported in Table 8.1 to be 40.8%. Therefore, under the assumption of the previous paragraph, MCD is able to come close to what we *perceive* to be the highest possible accuracies given an unimproved mask.

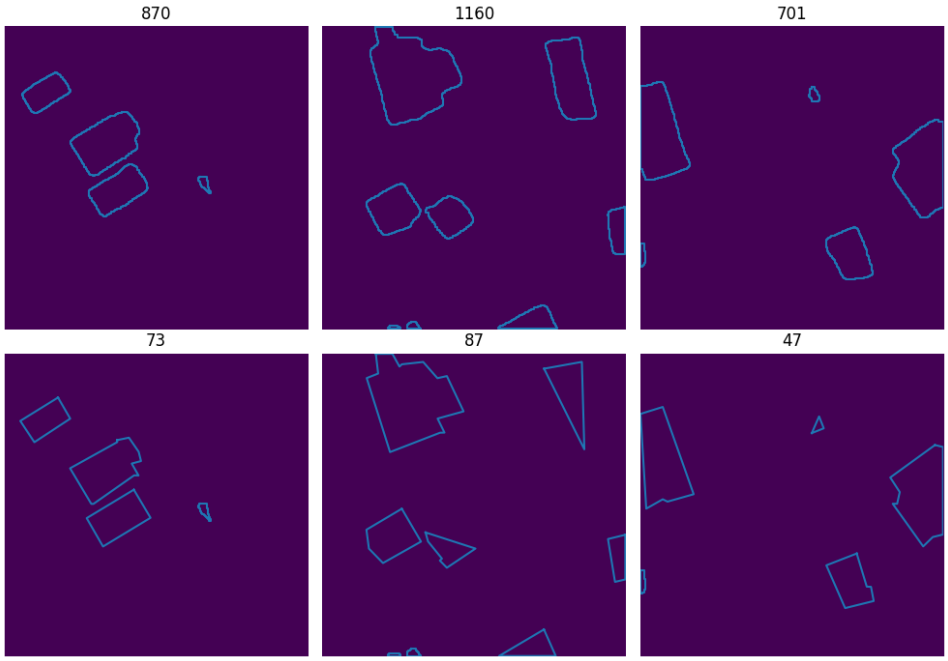

Figure 8.4: Comparison of DP simplification polygons of the predicted mask (top row) against the generated polygon using MAV-ATTR. The numbers above each image show the number of vertices used in representing all polygons in the image.

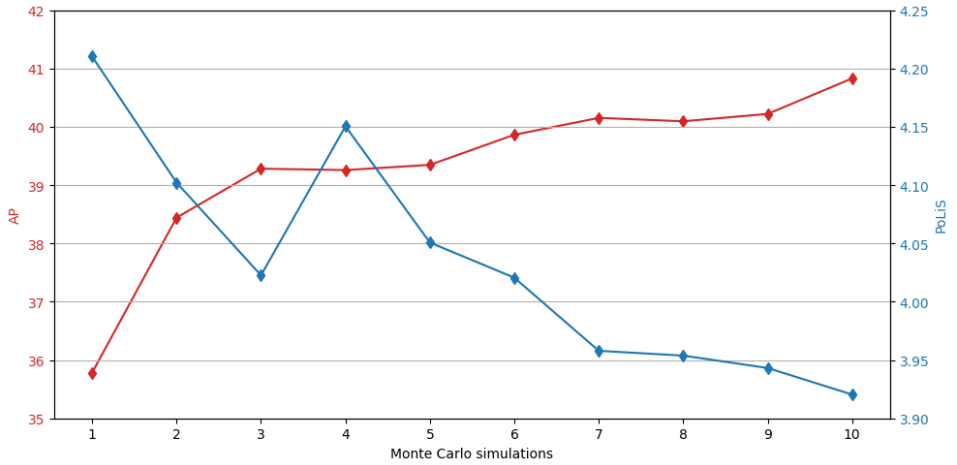

Figure 8.5: Analysis to see how performance is affected of the number of MC simulations used in predictions. The figure show the performance on two metrics, AP (red) and PoLiS [66] (blue).

8.1.4 Effects of Increasing Stochastic Forward Passes

With our proposed method, we perform another experiment analyzing the performance of increasing the number of stochastic forward passes during prediction between one and ten MC simulations. Figure 8.5 reports the AP and PoLiS accuracies of this experiment. The figure only reports results with AP and PoLiS metrics, since they fundamentally measure performance very differently, and more metrics does not provide any additional value to the reader. We see the performance increase is sharpest between one and three ensembles, before the AP flattens out until it increases again after the fifth ensemble predictions. The reported PoLiS [66] accuracies also follow the APs, but there is a large spike when using four simulations for the prediction. It is unclear why the performance in this metric spikes, however it could be the effect of a subnetwork that is consistently producing predictions that significantly differs from the rest.

Our model leveraging MCD consistently performs better than the baseline. However, this performance comes at the cost of ten times longer to predict segmentation, vertex and offset masks. The time it takes to perform a single forward pass will largely depend on the number of parameters in the model and the batch size used, however the number of parameters is constant through all our experiments, and the batch size differs based on the available hardware used, also the dropout layers have a minimal effect on the speed of a forward pass. The authors of [17] report their polygonization routine MAV-ATTR to be the fastest of the frameworks they evaluate, and our method does not affect the speed of MAV-ATTR, because we are supplying the algorithm with an average of

the predictions, meaning the computations are equal to that of their predictions, however the total time it takes to produce a single polygon is the number of MC samples used multiplied with the time it takes for a single detector to produce its prediction.

8.1.5 Effects of Data Distribution

As discussed in Section 7.1 we expect there to be a difference due to a distributional shift caused by differences in urban density, occlusions due to vegetation, and inherent properties in the imaging sensor. We also consider lighting to be a substantial factor, whereupon comparison of Figure 7.1 and Figure 7.2 we see clear color and shadow differences. As briefly mentioned in Subsection 7.1.2 the photons captured by the imaging satellite need to travel through a denser atmosphere than the Field dataset which has been captured by an airborne sensor. Another factor, not touched upon yet, is the uniformity of shadows in images captured by the satellite. The satellite orbits in a *heliosynchronous* fashion - where the lighting conditions are roughly equal at all times, with the obvious exception caused by cloud coverage - causing long shadows stretching across the image to be less prominent than for the airborne images that in general could be captured at any time. We see an example of this in the rightmost image of Figure 7.1 where shadows blend with the northeastern walls of the center building. While we expect the NN to circumvent such conditions given enough training examples, it adds to the uncertainty in the models' prediction.

8.1.6 Ground Range Effects

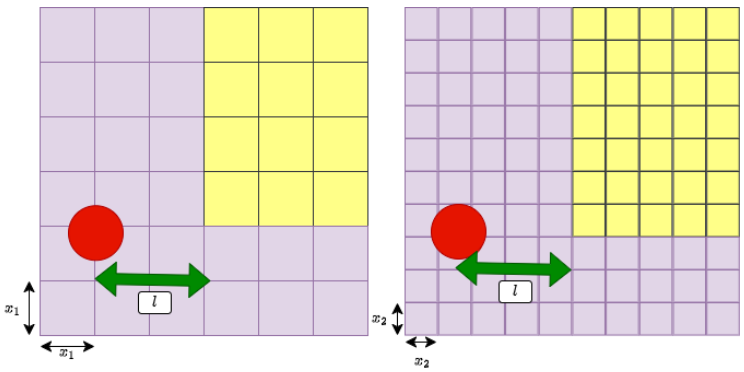

Figure 8.6: Example to illustrate an inherent problem when the spatial resolution increase. The problem arises when we are computing metrics based on distances represented by pixels. This leads to exaggerating the distance in the right figure, where the distances are in fact equal.

In addition to the aforementioned factors discussed, we also emphasize the effect when the spatial resolutions differ between datasets. Figure 8.6 tries to highlight what could possibly play a large factor in the difference of performance, when comparing the model trained on the AICrowd [61] dataset agains the model trained on the Field dataset. While we are uncertain regarding the exact spatial resolution of the AICrowd dataset as mentioned in Subsection 7.1.2, we demonstrate the issues with differing spatial resolutions by assuming that a single pixel of the AICrowd dataset represent 0.5m in the real world. At the same time we know the pixels of the Field dataset to represent 0.1m. Given two samples of a building, much like in Figure 8.6, where the left image has a spatial resolution of e.g., $x_1 = c$, while the right image has $x_2 = c/2$. The red circle denotes a predicted vertex, while the yellow boxes represent binary segmentation maps. The green arrow has a length of $l = 2c$ and are of equal lengths in both images. We count its distance using the number of pixels between the vertex and the mask to be 2 in the left example, and 4 in the right. This means that the polygon generation algorithm MAV-ATTR, which discards or stores a vertex, based on the pixel distance, will have a harder time when the spatial resolution is higher, because the same error in true ground distance is not reflected in the pixel distance which the algorithm uses.

Following our explanation, we assume this explains some of the gap observed between Table 8.1 and Table 8.2. We explain that this is due to the backbone of the framework, which resizes all input images to 128×128 pixels before predicting the positions of masks and vertices, followed by upsampling to the original image size and filtering out redundant vertices on the upsampled image. The filtering process use the fixed distance threshold of $\tau_d = 5$, which we here highlight as a problem when spatial resolutions differ.

The results presented in [17] reflects this to a greater extent. The authors test their models on 4 datasets, AICrowd [61], Inria [62], Open Cities and Shanghai datasets. While their model performs better than all others in the relevant metrics, with similar results to Table 8.2 on the AICrowd dataset, we observe that the datasets with much higher spatial resolution - such as the drone captured Open Cities dataset with 4 cm to 8 cm - consistently score lower than the *low* resolution datasets. Their explanation as to why this dataset is more difficult than AICrowd, is blamed on other properties, namely occlusion [17]. We do however agree with their reasoning, but hope our analysis will highlight other influential reasons.

To build on our analysis, we use the recorded optimal thresholds of each of the datasets evaluated by us, as well as in [17]. By comparing the optimal distance thresholds presented for each model trained on each of the datasets, which is found in Table 8.3.

Table 8.3: Overview of values used in our analysis, including values from datasets of experiments conducted in [17]. Note that the Open Cities dataset has a spatial resolution of 0.04m to 0.08m, which we replace by an assumed average spatial resolution.

Dataset	Optimal τ_d interval	Spatial Resolution [m]	Image size
AICrowd [61]	4-6	0.5	300
Field	9-11	0.1	512
Inria [62]	4-7	0.3	512
Shanghai	4-6	0.3	256
Open Cities	10-13	0.06 (avg)	512

We then use the values of Table 8.3 to transform the spatial resolutions of the original images into the effective spatial resolutions of the downsampled image sizes of 128×128 pixels, which the backbone use for predictions using the formula,

$$\begin{aligned} \text{Ground range size} &= \text{image size} \times \text{spatial resolution} \\ \text{Effective spatial resolution} &= \frac{\text{Ground range size}}{128} \\ \text{Ground range accepted error} &= \text{Effective spatial resolution} \times \tau_d \end{aligned}$$

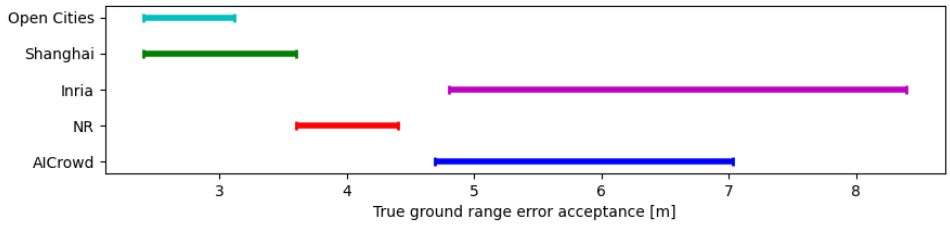

Figure 8.7: Comparison of the true error which MAV-ATTR will accept for models trained on the datasets highlighted. We use a combination of our own results in addition to those reported by [17] for our analysis.

Figure 8.7 shows these results illustrated as intervals. The interesting result is that for the downstream tasks which use the predictions of these models, are more interested in a low *true ground range error* - the actual displacement of the vertices of the polygons - however this is not necessarily reflected in the metrics, as shown in Table 8.2. We conclude this analysis stating that we only can really trust comparisons of models that are trained on the same datasets, and that while metrics such as AP and its variations (see Section 7.2) are useful in quantifying the model performance within a dataset, it does not exclusively reflect the useful value when we compare between datasets.

8.2 Uncertainty Scaled Border Threshold

We now direct our focus from exclusively using MCD to evaluation of our proposed method from Section 6.3. As have been thoroughly discussed and illustrated in the previous section, Section 8.1 the framework benefits from our ensemble strategy. We will in this section show how we attempted to improve what vertices are selected by MAV-ATTR by addressing the fixed distance threshold, replacing it with a dynamically scaled one, using uncertainties which we gain from our previous MCD proposal.

In finding a way to leverage uncertainties in the vertex selection process we met large difficulties due to there being no obvious or intuitive way of using uncertainties to modify thresholds. The proposal presented in Section 6.3 is the result of an extensive analysis that was conducted prior to the one presented here to modify the algorithm. As we will see, we are able to improve the accuracies on some metrics, but the inclusion of MCD from the previous section resulted in the largest improvement.

As have been discussed in Subsection 8.1.6, the choice of distance threshold τ_d significantly differ between datasets. Hence, we expect to see vast differences between the two during our comparison of the results between the datasets. Subsection 8.2.1 presents our findings evaluated on the Field dataset, being followed by Subsection 8.2.2 which perform an evaluation on the AICrowd dataset.

8.2.1 Uncertainty Guidance for Norwegian Homes

Table 8.4: Comparison between MAV-ATTR and our proposed modification leveraging uncertainty evaluated on the Field dataset. We also include the distance threshold which gave the optimal result. The same results are found both in Figure 8.8 and Figure 8.9.

Polygon generation	AP	$AP^{boundary}$	PoLiS	C-IoU	IoU	τ_d
MAV-ATTR [17]	40.7	22.6	3.920	63.6	74.8	10
Dynamic threshold	40.4	22.3	3.901	64.3	74.7	11

Figure 8.8 illustrate the performance over a varying distance threshold using both the original MAV-ATTR algorithm with predictions averaged over ten stochastic forward passes against our proposed method where the base threshold is varied in the same interval. We notice that for smaller distance thresholds, the performance is better than the MCD counterpart, while they are similar towards the optimal value of the algorithm, because of the intervals of thresholds that perform well for earlier values of τ_d. Using these results we may state that

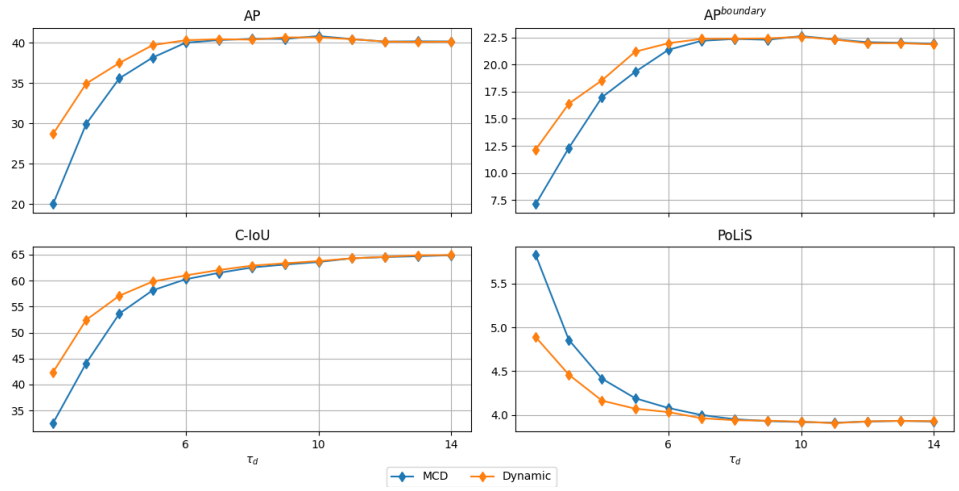

Figure 8.8: Evaluation of our benchmark outlined in Figure 8.3 to our dynamic threshold selection algorithm. The blue line denotes the benchmark using MCD while the orange line show our modification to MAV-ATTR using a dynamic threshold scaled using the uncertainty of each vertex. A zoomed-in version of the relevant thresholds are available in Figure 8.9. These accuracies are from evaluations using the Field dataset.

this initial evaluation demonstrate a greater stability, and less dependence of the distance threshold τ_d, which is a goal of the method.

By displaying only the values between a smaller threshold, as done in Figure 8.9, we are able to more closely inspect the performance of our proposals. We see the AP is similar between the experiments, but the more difficult metric of AP boundary has a higher and more stable performance before the baseline surpass our dynamic algorithm at its optimal threshold. The same is true for both the C-IOU and PoLiS metrics. In fact C-IOU performs consistently similar or better than the benchmark for this dataset. While the PoLiS score is not always better than the benchmark, it outperforms it at its optimal threshold.

We believe that our dynamic threshold selection improves for lower distance thresholds than the baseline, since it chooses vertices based on the uncertainty associated with a vertex. This uncertainty is then in our evaluation scaled between 0 and $\tau_s = 2$, which is then added to the distance threshold τ_d. For consistently similarly uncertain vertices in a prediction, this could simply cause our modified algorithm to collapse into MAV-ATTR but with a scaled, nearly fixed distance threshold. Which has both positive and negative consequences; positive in that when the dynamic threshold does not work, it functions similar to an already established polygon generation algorithm, with good performance. On the other hand; the similarity to MAV-ATTR means the dynamic selection is lost and the robustness and stability seen in Figure 8.9 vanishes when using a

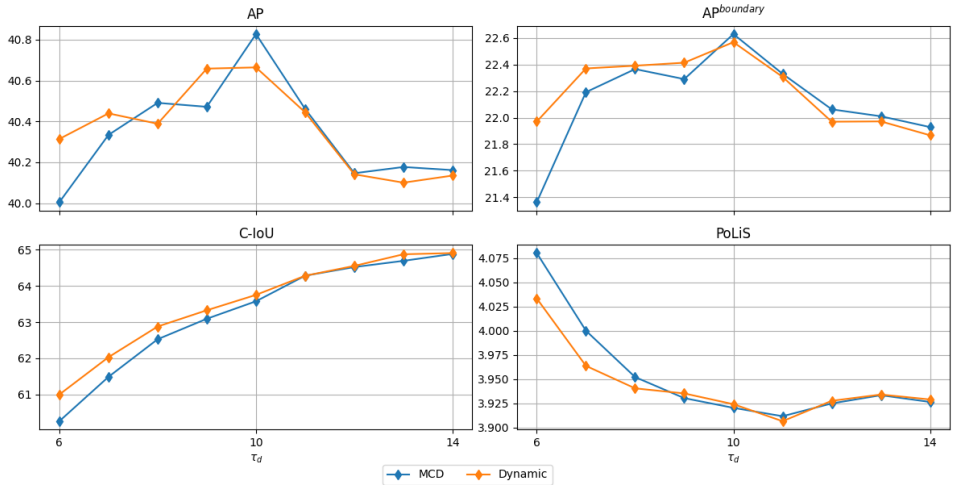

Figure 8.9: The same as Figure 8.8, but the relevant parts are enhanced. The results show that for some thresholds, our proposed dynamic threshold selection performs better than our baseline. Of significance, we notice how the more challenging C-IOU and AP boundary perform better for thresholds below the optimal threshold of the benchmark, PoLiS also perform better for several thresholds.

suboptimal distance threshold.

We perform a qualitative analysis, where we consider two examples where our modifications appears to improve the quality of the polygons, and one example where the modifications fail. Figure 8.10 demonstrate how similar the methods are. For each of the examples, there is only a single vertex that has been modified. in (a) and (b) we have examples where the modifications improve the overall extracted polygons, while (c) demonstrates the polygon of the left building within the yellow circle to incorporate a vertex from the neighboring building. While the successful cases show a sensible alternative polygon, the overall similarity, and preciseness of the original MAV-ATTR algorithm is maintained.

8.2.2 Dynamic Vertex Selection on Large Dataset

Table 8.5: Comparison between MAV-ATTR [17] and our proposed dynamic threshold modification to MAV-ATTR. The results of this table can also be seen in Figure 8.11 and Figure 8.12.

Polygon generation	AP	$AP^{boundary}$	PoLiS	C-IOU	IOU	τ_d
MAV-ATTR [17]	69.7	51.8	1.155	84.4	90.2	5
Dynamic threshold	69.8	51.9	1.152	84.5	90.2	4

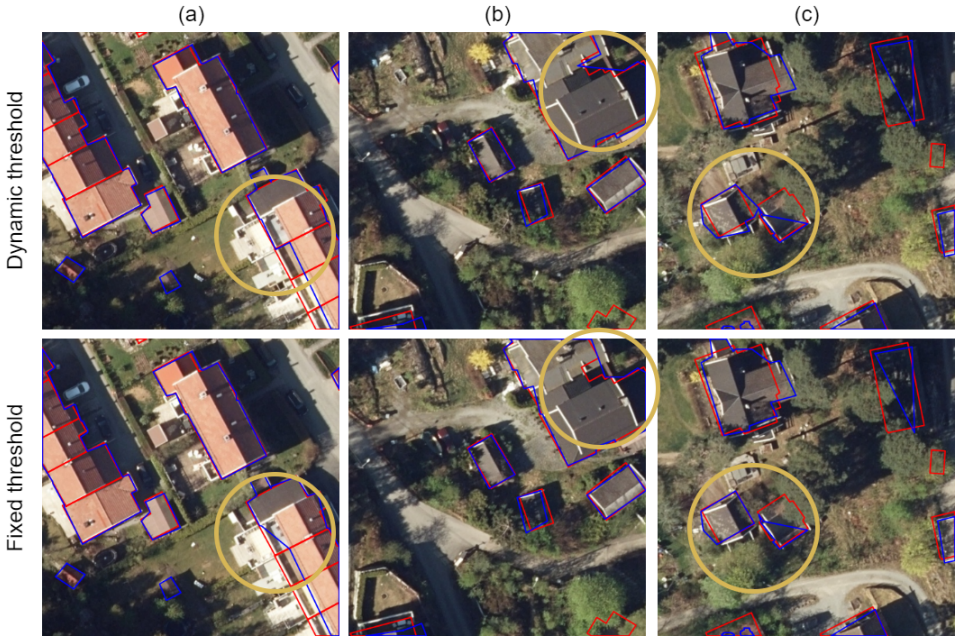

Figure 8.10: Qualitative analysis of our dynamic threshold selection algorithm. The top rows are examples where the dynamic thresholds are used, while the bottom row shows the fixed threshold. The predictions are shown in blue, while the red polygons represent the ground truth annotations. Columns (a) and (b) highlight cases where our method improves the quality of the polygon by being able to select vertices at a greater distance if the model is uncertain in the placemen of the vertex. Column (c) highlights a failure case, where the vertices between two buildings are too close, causing the polygon of the left building to select an additional polygon which should not have been there.

While our previous analysis indicates stable performance reflected on both sides of the *optimal distance threshold* τ_d, the NN trained on the AICrowd dataset [61] only experience this behavior on the lower side of the reflection point, as seen in Figure 8.11. Conversely, our proposal improves robustness and stability below the optimal threshold. Examining Figure 8.12 reveals that uncertainty guidance improves the evaluation metrics by a small amount.

This indicates that the use of uncertainty guidance during the vertex selection process improves the accuracy of the predicted polygons. However, it is important to highlight that while our modifications to MAV-ATTR improve the performance, the gains are miniscule when compared with our initial proposal using MCD. We believe that our results demonstrate that guiding the selection process using uncertainty is able to positively impact performance, although the algorithm itself needs refinement. To further motivate the need for additional modifications, we reiterate that the only uncertainty used is that of

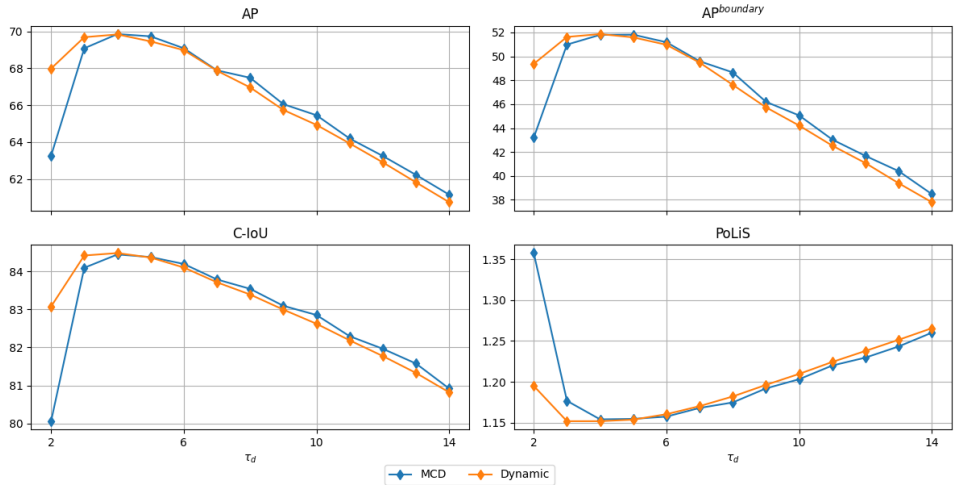

Figure 8.11: Comparison of evaluation metrics on the AICrowd dataset [61] using our dynamic version of MAV-ATTR.

the concave and convex prediction maps. With this and our results in mind, a critique of our implementation is that there is still the possibility to use the uncertainty of the segmentation maps and offset maps, or a combination of these.

Illuminating again the assumptions about the distance between the direct comparison of predicted masks and polygons, that was discussed in Subsection 8.1.3. This assumed upper boundary may limit the algorithms' ability to substantially improve performance, subsequently this motivates the use of uncertainty to improve the predicted segmentation maps as well. During our work, we experimented with alternatives that used the uncertainty map of segmentation to also influence the distance threshold during polygon generation. However, our implementations were unsuccessful in improving the performance on the evaluation metrics. Due to the ambiguous nature of the task, we instead opted for exclusively scrutinizing how vertex uncertainties could be leveraged, to demonstrate the potential use of uncertainties that are byproducts of MCD.

8.3 Future Work

From our analysis of our proposed models, a deep insight into the inner workings of building extraction has been gained. This section will present other methods and extensions of our methods that deserve further work, and potentially could prove beneficial in building detection, and the larger field of object

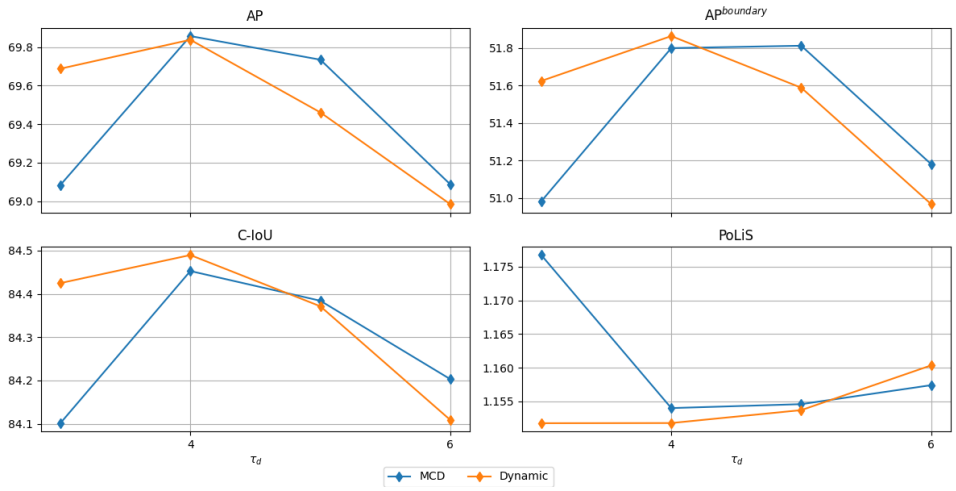

Figure 8.12: Enhanced version of Figure 8.11 that emphasize the area near the optimal threshold distance. We see that our modifications outperforms MAV-ATTR when choosing a distance threshold of four.

detection.

8.3.1 The Need for Higher Quality Segmentation Maps

Aligned with the assumptions made in Subsection 8.1.3 there appears to be an upper bound of the achievable performance of generated polygons. The extracted segmentation map appears to control this upper bound, subsequently causing the algorithms that generate polygons to only approach this boundary, not surpassing it. While this problem does not affect end-to-end learned methods, such as [12, 15, 70], [17] performs an analysis for the algorithmic polygon generation methods; given a segmentation mask, they evaluate four different polygon generation methods, their own, Frame-Field ASM [13], *ASIP*[1] [71], and DP simplification [52]. None of these methods are able to surpass the performance of the segmentation mask, although their own MAV-ATTR is closest.

For most buildings, the segmentation maps are of sufficient quality, however through the use of MCD it is possible to find examples of buildings where several ensembles disagree, for instance as we show in the uncertainty of segmentation in (a) and (b) of Figure 8.2. We believe an interesting approach would be to use ideas from Active Learning (AL) [60], to query an expert that receives e.g., the bounding box of the uncertain buildings, and delineates the image.

1. ASIP refers to the title of their paper: *Approximating shapes in images with low-complexity polygons* [71]

Another approach could employ a refining NN such as the one used by [12] which is given the bounding box and an initial prediction for it to improve on as input. Our uncertainty of segmentation masks show greater uncertainty for buildings that are partially occluded by vegetation, subsequently motivating this approach.

8.3.2 Spatial Resolution Independent Evaluation Metrics

The argument in Subsection 8.1.6 highlights the unfair advantage for datasets that have a lower spatial resolution. The problem arise when our metrics depend on the spatial resolution, where we demonstrate that while extremely high resolution images perform worse than lower resolution images, the in-situ error is much less for high resolution datasets. Downstream tasks such as GIS applications require models that have a high accuracy in real world measurements, motivating the need for evaluation metrics that measure the true error.

8.3.3 Use of Uncertainty Maps From Segmentation and Offset Maps

We present modifications to MAV-ATTR that improve the robustness of the original implementation from [17] by the use of uncertainty maps of the vertex predictions. While the uncertainty of the segmentation maps clearly distinguish difficult walls that are occluded by vegetation, see Figure 8.2, the confidence threshold to produce a binary segmentation map, that we set to $\tau > 0.5$, could be varied to include further away vertices in the final polygon. Through our experiments the vertex predictor in general predict many redundant vertexes which are discarded when the segmentation map is too uncertain.

We believe that a natural next step to this project would be to explore new ways of finding, and leveraging uncertainties in DL, as well as extending the scope past the exclusive use of epistemic uncertainty of the model, to also investigate aleatoric uncertainties inherent in the observations.

9

Conclusion

This study focused on enhancing building detection from aerial and satellite images by leveraging advanced Neural Network techniques and uncertainty estimation methods. In our background theory we explored Neural Networks, Convolutional Neural Networks, Deep Learning feature extractors such as ResNet [43] and HRNet [48], and Monte Carlo Dropout for uncertainty estimation.

We proposed two novel enhancements to existing algorithms. Firstly, we integrated dropout layers into the *Hierarchical Supervision*, which is the current state-of-the-art polygonal building detection algorithm by [17], enabling the use of Monte Carlo Dropout for approximate Bayesian Inference following [7, 30, 29, 26, 28, 27]. Second, we modified the algorithmic Mask-and-Vertices Attraction polygon generation process by incorporating a dynamic threshold guided by uncertainty instead of a fixed threshold.

Our experiments demonstrated that using Monte Carlo Dropout significantly improved performance. Additionally, the dynamic threshold approach, offering a more modest improvement, further enhanced the accuracy of building detection and the stability and robustness to the distance threshold.

While there are similar approaches leveraging uncertainty [7, 30, 28], our approach impacts the field of building detection by demonstrating the use of uncertainty maps in vertex selection to improve accuracies in a polygon generation setting. Enhanced accuracy and reliability of detection algorithms

have the potential to benefit various applications, including urban planning, environmental monitoring, and disaster response efforts.

While our method showed promise, we acknowledge certain limitations. These include constraints related to data quality and occlusion due to vegetation, the ambiguous nature of using uncertainty during polygon generation, and increased computational resources required for implementing ensemble strategies, and uncertainty estimation.

The practical applications of our research are broad. Improved building detection algorithms are valuable for urban planners in designing and managing city infrastructure. Environmental researchers can use these methods for monitoring land use and changes over time, while disaster response teams can benefit from accurate building detection in post-disaster damage assessments [10].

Overall, this research contributes to the advancement of building detection methodologies, offering new insights and practical approaches for enhancing detection accuracy. Our findings pave the way for future innovations and applications in this area of study.

Bibliography

[1] P. Barmpoutis, P. Papaioannou, K. Dimitropoulos, and N. Grammalidis, "A review on early forest fire detection systems using optical remote sensing," *Sensors*, vol. 20, no. 22, p. 6442, 2020.

[2] Y. Wang, Z. Zhang, L. Feng, Y. Ma, and Q. Du, "A new attention-based cnn approach for crop mapping using time series sentinel-2 images," *Computers and electronics in agriculture*, vol. 184, p. 106090, 2021.

[3] T. T. Nguyen, T. D. Hoang, M. T. Pham, T. T. Vu, T. H. Nguyen, Q.-T. Huynh, and J. Jo, "Monitoring agriculture areas with satellite images and deep learning," *Applied Soft Computing*, vol. 95, p. 106565, 2020.

[4] J. Xue, B. Su, *et al.*, "Significant remote sensing vegetation indices: A review of developments and applications," *Journal of sensors*, vol. 2017, 2017.

[5] B. Ayhan, C. Kwan, B. Budavari, L. Kwan, Y. Lu, D. Perez, J. Li, D. Skarlatos, and M. Vlachos, "Vegetation detection using deep learning and conventional methods," *Remote Sensing*, vol. 12, no. 15, p. 2502, 2020.

[6] X. Zhang, L. Han, L. Han, and L. Zhu, "How well do deep learning-based methods for land cover classification and object detection perform on high resolution remote sensing imagery?," *Remote Sensing*, vol. 12, no. 3, p. 417, 2020.

[7] M. Kampffmeyer, A.-B. Salberg, and R. Jenssen, "Semantic segmentation of small objects and modeling of uncertainty in urban remote sensing images using deep convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 1–9, 2016.

[8] G. Grekousis, "Artificial neural networks and deep learning in urban geography: A systematic review and meta-analysis," *Computers, Environment and Urban Systems*, vol. 74, pp. 244–256, 2019.

[9] Q. Yuan, H. Shen, T. Li, Z. Li, S. Li, Y. Jiang, H. Xu, W. Tan, Q. Yang, J. Wang, *et al.*, "Deep learning in environmental remote sensing: Achievements and challenges," *Remote Sensing of Environment*, vol. 241, p. 111716, 2020.

[10] R. Gupta and M. Shah, "Rescuenet: Joint building segmentation and damage assessment from satellite imagery," in *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 4405–4411, IEEE, 2021.

[11] B. Bischke, P. Helber, J. Folz, D. Borth, and A. Dengel, "Multi-task learning for segmentation of building footprints with deep neural networks," in *2019 IEEE International Conference on Image Processing (ICIP)*, pp. 1480–1484, IEEE, 2019.

[12] W. Li, W. Zhao, J. Yu, J. Zheng, C. He, H. Fu, and D. Lin, "Joint semantic–geometric learning for polygonal building segmentation from high-resolution remote sensing images," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 201, pp. 26–37, 2023.

[13] N. Girard, D. Smirnov, J. Solomon, and Y. Tarabalka, "Polygonal building extraction by frame field learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5891–5900, 2021.

[14] Z. Li, J. D. Wegner, and A. Lucchi, "Topological map extraction from overhead images," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1715–1724, 2019.

[15] S. Zorzi, S. Bazrafkan, S. Habenschuss, and F. Fraundorfer, "Polyworld: Polygonal building extraction with graph neural networks in satellite images," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1848–1857, 2022.

[16] V. Iglovikov, S. Seferbekov, A. Buslaev, and A. Shvets, "Ternausnetv2: Fully convolutional network for instance segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 233–237, 2018.

[17] B. Xu, J. Xu, N. Xue, and G.-S. Xia, "Hisup: Accurate polygonal mapping of buildings in satellite imagery with hierarchical supervision," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 198, pp. 284–296, 2023.

[18] X. X. Zhu and R. Bamler, "A sparse image fusion algorithm with application to pan-sharpening," *IEEE transactions on geoscience and remote sensing*, vol. 51, no. 5, pp. 2827–2836, 2012.

[19] S. Müller, D. W. Zaum, *et al.*, "Robust building detection in aerial images," *International Archives of Photogrammetry and Remote Sensing*, vol. 36, no. B2/W24, pp. 143–148, 2005.

[20] B. Sirmacek and C. Unsalan, "Urban-area and building detection using sift keypoints and graph theory," *IEEE transactions on geoscience and remote sensing*, vol. 47, no. 4, pp. 1156–1167, 2009.

[21] H. Mayer, "Automatic object extraction from aerial imagery—a survey focusing on buildings," *Computer vision and image understanding*, vol. 74, no. 2, pp. 138–149, 1999.

[22] A. C. Jensen, "Beyond output-mask comparison: A self-supervised inspired object scoring system for building change detection," in *Northern Lights Deep Learning Conference*, pp. 97–103, PMLR, 2024.

[23] L. Castrejon, K. Kundu, R. Urtasun, and S. Fidler, "Annotating object instances with a polygon-rnn," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5230–5238, 2017.

[24] D. Acuna, H. Ling, A. Kar, and S. Fidler, "Efficient interactive annotation of segmentation datasets with polygon-rnn++," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 859–868, 2018.

[25] H. Ling, J. Gao, A. Kar, W. Chen, and S. Fidler, "Fast interactive object annotation with curve-gcn," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 5257–5266, 2019.

[26] Y. Gal and Z. Ghahramani, "Bayesian convolutional neural networks with bernoulli approximate variational inference," *arXiv preprint arXiv:1506.02158*, 2015.

[27] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*, pp. 1050–1059, PMLR, 2016.

[28] A. Kendall, V. Badrinarayanan, and R. Cipolla, "Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding," *arXiv preprint arXiv:1511.02680*, 2015.

[29] A. Kendall and Y. Gal, "What uncertainties do we need in bayesian deep learning for computer vision?," *Advances in neural information processing systems*, vol. 30, 2017.

[30] S. Hansen, S. Gautam, S. A. Salahuddin, M. Kampffmeyer, and R. Jenssen, "Adnet++: A few-shot learning framework for multi-class medical image volume segmentation with uncertainty-guided feature refinement," *Medical Image Analysis*, vol. 89, p. 102870, 2023.

[31] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[32] E. Alpaydin, *Introduction to machine learning*. MIT press, second ed., 2010.

[33] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[34] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization.," *Journal of machine learning research*, vol. 12, no. 7, 2011.

[35] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[36] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[37] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[38] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[39] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[40] J. Duyck, M. H. Lee, and E. Lei, "Modified dropout for training neural network," *Paper Published online*, vol. 49, p. 50, 2014.

[41] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[42] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[43] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[44] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[45] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[46] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.

[47] O. Ronneberger, P. Fischer, and T. 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*, pp. 234–241, Springer, 2015.

[48] J. Wang, K. Sun, T. Cheng, B. Jiang, C. Deng, Y. Zhao, D. Liu, Y. Mu, M. Tan, X. Wang, *et al.*, "Deep high-resolution representation learning for visual recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 10, pp. 3349–3364, 2020.

[49] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[50] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969, 2017.

[51] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," in *Seminal graphics: pioneering efforts that shaped the field*, pp. 347–353, 1998.

[52] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the

number of points required to represent a digitized line or its caricature," *Cartographica: the international journal for geographic information and geovisualization*, vol. 10, no. 2, pp. 112–122, 1973.

[53] R. Sinkhorn and P. Knopp, "Concerning nonnegative matrices and doubly stochastic matrices," *Pacific Journal of Mathematics*, vol. 21, no. 2, pp. 343–348, 1967.

[54] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *International journal of computer vision*, vol. 1, no. 4, pp. 321–331, 1988.

[55] N. Xue, S. Bai, F. Wang, G.-S. Xia, T. Wu, and L. Zhang, "Learning attraction field representation for robust line segment detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1595–1603, 2019.

[56] N. Xue, S. Bai, F.-D. Wang, G.-S. Xia, T. Wu, L. Zhang, and P. H. Torr, "Learning regional attraction for line segment detection," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 6, pp. 1998–2013, 2019.

[57] Q. Wang, B. Wu, P. Zhu, P. Li, W. Zuo, and Q. Hu, "Eca-net: Efficient channel attention for deep convolutional neural networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11534–11542, 2020.

[58]

[59] J. Arbel, K. Pitas, M. Vladimirova, and V. Fortuin, "A primer on bayesian neural networks: review and debates," *arXiv preprint arXiv:2309.16314*, 2023.

[60] Y. Gal, R. Islam, and Z. Ghahramani, "Deep bayesian active learning with image data," in *International conference on machine learning*, pp. 1183–1192, PMLR, 2017.

[61] S. P. Mohanty, J. Czakon, K. A. Kaczmarek, A. Pyskir, P. Tarasiewicz, S. Kunwar, J. Rohrbach, D. Luo, M. Prasad, S. Fleer, *et al.*, "Deep learning for understanding satellite imagery: An experimental survey," *Frontiers in Artificial Intelligence*, vol. 3, p. 534696, 2020.

[62] E. Maggiori, Y. Tarabalka, G. Charpiat, and P. Alliez, "Can semantic labeling methods generalize to any city? the inria aerial image labeling benchmark," in *2017 IEEE International geoscience and remote sensing*

symposium (IGARSS), pp. 3226–3229, IEEE, 2017.

[63] A. Van Etten, D. Lindenbaum, and T. M. Bacastow, "Spacenet: A remote sensing dataset and challenge series," *arXiv preprint arXiv:1807.01232*, 2018.

[64] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pp. 740–755, Springer, 2014.

[65] B. Cheng, R. Girshick, P. Dollár, A. C. Berg, and A. Kirillov, "Boundary iou: Improving object-centric image segmentation evaluation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 15334–15342, 2021.

[66] J. Avbelj, R. Müller, and R. Bamler, "A metric for polygon comparison and building extraction evaluation," *IEEE Geoscience and Remote Sensing Letters*, vol. 12, no. 1, pp. 170–174, 2014.

[67] C. Salomonsen, "Polygonal building extraction from aerial and satellite images," *UiT Capstone Project*, 2023.

[68] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[69] L. N. Smith, "Cyclical learning rates for training neural networks," in *2017 IEEE winter conference on applications of computer vision (WACV)*, pp. 464–472, IEEE, 2017.

[70] S. Zorzi and F. Fraundorfer, "Re: Polyworld-a graph neural network for polygonal scene parsing," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 16762–16771, 2023.

[71] M. Li, F. Lafarge, and R. Marlet, "Approximating shapes in images with low-complexity polygons," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8633–8641, 2020.